Improved Algorithms for Linear Inequalities with Two Variables per Inequality

Edith Cohen* Nimrod Megiddo†

Revised: August, 1993

Abstract

We show[1] that a system of m linear inequalities with n variables, where each inequality involves at most two variables, can be solved in $\tilde{O}(mn^2)$ time[2] deterministically, and in $\tilde{O}(n^3 + mn)$ expected time using randomization. Parallel implementations of these algorithms run in $\tilde{O}(n)$ time, where the deterministic algorithm uses $\tilde{O}(mn)$ processors and the randomized algorithm uses $\tilde{O}(n^2 + m)$ processors. The bounds significantly improve over previous algorithms. Our randomized algorithm is based on novel insights into the structure of the problem.

*AT&T Bell Labs, Murray Hill, NJ 07974. Research done while the author was at Stanford University and IBM Almaden Research Center. Research partially supported by ONR-N00014-91-C-0026 and by NSF PYI Grant CCR-8858097, matching funds from AT&T and DEC.

†IBM Research, Almaden Research Center, San Jose, CA 95120-6099, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel. Research partially supported by ONR-N00014-91-C-0026.

[1] A preliminary version of the results appeared in [6].

[2] We denote $\tilde{O}(f) = O(f \operatorname{polylog} n \operatorname{polylog} m)$.

1. Introduction

In this paper we consider the following class of linear systems:

Definition 1.1 A *TVPI* system is a system of linear inequalities where each inequality involves at most two variables (i.e., a system of the form $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$, where $\boldsymbol{A} \in R^{m \times n}$ is a matrix, $\boldsymbol{b} \in R^m$ is a real vector, and each row of $\boldsymbol{A}$ contains at most two nonzero entries). We denote the number of inequalities by m , and the variables by $x_1, \ldots, x_n$. We denote by $F = \{\boldsymbol{x} \in R^n | \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$ the set of feasible solutions.
A TVPI system is called *monotone* if the two nonzero entries in each row have opposite signs. See Figure 1 for an example of a TVPI system.

Our goal is to either find a point that satisfies all the inequalities or conclude that no such point exists. The structure of TVPI systems enables us to obtain specialized algorithms that are faster than known algorithms for solving general linear systems. The algorithms given here can also be adapted to find a solution that minimizes or maximizes a specific variable. We summarize previous work on solving TVPI systems. Shostak [17] characterized the set of solutions of TVPI systems and gave an algorithm that is exponential in the worst case. Nelson [16] gave an $n^{O(\log n)}$ algorithm. Aspvall and Shiloach [3] and Aspvall [2] proposed algorithms that perform $O(mn^3 I)$ and $O(mn^2 I)$ arithmetic operations, respectively, where I is the size of the binary encoding of the problem. Megiddo [15] proposed the first strongly polynomial time algorithm for the problem, which performs $O(mn^3 \log m)$ operations. The parallel implementation of Megiddo's algorithm runs in $O(n^3 \log m)$ time using $O(m)$ processors.

The algorithms presented here improve the sequential and parallel time bounds. We give an $O\left(mn^2(\log m + \log^2 n)\right)$ time deterministic algorithm, which has a parallel implementation that runs in $O\left(n(\log m + \log^2 n)\right)$ time using $O(mn)$ processors. An additional improvement is obtained through using randomization: we give an algorithm that runs in $O\left(n^3 \log n + mn(\log^5 n + \log m \log^3 n)\right)$ expected time. A parallel implementation runs in $O\left(n(\log^5 n + \log^3 n \log m)\right)$ expected time using $O(n^2 + m)$ processors. The space requirement of the algorithms presented here is $O(n^2 + m)$. The effort involved in translating these algorithms into actual programs is about the same as in the previously known algorithms, and the hidden constants in the time bounds are still reasonable. Subsequent to our work, Hochbaum and Naor [12] found a new $O(mn^2 \log m)$ deterministic algorithm for the problem. Their algorithm, however, runs in $O(n^2 \log m)$ time in parallel, and it does not seem possible to combine it with the randomized approach to yield algorithms with better expected time.

Section 2 gives the preliminaries. We give a characterization of polyhedra defined by TVPI systems. We represent a TVPI system by a directed multigraph with $2n$ vertices and $2m$ edges. The edges have linear functions of a single variable as "weights", and the weight of a path is interpreted as compositions of the weights along the path's edges. We review a result of Shostak [17] that relates weights of directed cycles and paths in this multigraph to bounds on the feasible region. We also discuss some subproblems and show how to solve them by performing a "Bellman-Ford shortest-path" type computation on the representing multigraph where initial numerical values are assigned to vertices and "adding" an edge-weight is

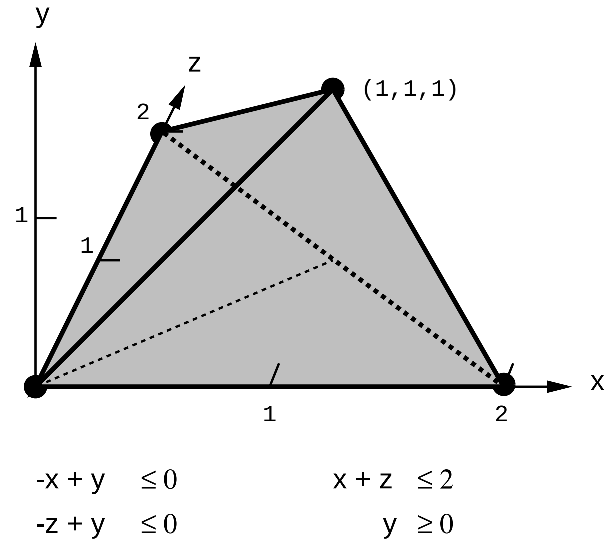

Figure 1: An example of a TVPI system with 3 variables

replaced by substituting the current value at the vertex in the linear function comprising the edge-weight (hence, by an $O(mn)$ sequential algorithm, or, by an $O(n)$ time parallel algorithm using $O(m)$ processors). In particular we discuss the subproblem of *locating a value*: for a numerical value ξ and $1 \leq i \leq n$, decide whether a TVPI system with the addition of a constraint of the form $x_i \geq \xi$ or $x_i \leq \xi$ remains feasible. An $O(mn)$ algorithm for locating a value was given by Aspvall and Shiloach [3].

In Section 3 we introduce a framework for solving TVPI systems that amounts to performing a "Floyd-Warshall all-pairs shortest-path" type computation where resolving comparisons is replaced by locating values. This formulation enables us to reduce solving TVPI systems to performing $\tilde{O}(m + n^3)$ operations for the Floyd-Warshall type computation and in addition solving a polylogarithmic number of instances of the more general subproblem of *locating a pool of values*: for numerical values $\xi_1, \ldots, \xi_n$ choose bounds $s_1, \ldots, s_n$, where $s_i \in \{x_i \geq \xi_i, x_i \leq \xi_i\}$, such that the system with the additional constraints $s_1, \ldots, s_n$ remains feasible. Obviously, a pool of values can be located by n sequential applications of an algorithm that locates a single value. This yields an $O(mn^2)$ time sequential algorithm that in parallel, runs in $O(n^2)$ time using $O(m)$ processors. The latter yields a $\tilde{O}(mn^2)$ deterministic algorithm for solving TVPI systems.

In Section 4 we present better algorithms for locating a pool. We first give an algorithm that improves the parallel running time to $O(n)$ using $O(mn)$ processors. This algorithm enables us to obtain a $\tilde{O}(n)$ parallel deterministic running time (using $O(mn)$ processors) for solving TVPI systems. Section 4 also contains an overview of a two stage $\tilde{O}(mn)$ time randomized algorithm. In parallel, the randomized algorithm runs in $\tilde{O}(n)$ time and uses $O(m)$ processors. This algorithm enables us to obtain the stated randomized bounds for solving TVPI systems. The randomized algorithm constitutes the most novel and the technically most difficult part of this work. Sections 5 and 6 are concerned with the details of the two stages.

In Section 7 we discuss the special structure of monotone systems. These systems are of special interest since the linear programming duals of generalized transshipment problems are monotone. In [4, 7] the authors exploited this relationship and presented algorithms for generalized network flow problems (see also Adler and Cosares [1]).

Section 8 contains concluding remarks and a discussion of open problems.

2. Preliminaries

Consider two two-variable inequalities that share a common variable. In some cases, these inequalities can be combined to yield a new two-variable inequality that is implied by the previous two. For example, the inequalities $x_1 \leq 2x_2 - 4$ and $x_2 \leq 4x_3 + 5$ can be combined to yield $x_1 \leq 8x_3 + 6$. The inequalities $x_1 \leq 2x_2 - 4$ and $x_2 \geq 4x_3 + 5$, however, can not be combined in a similar fashion. The notion of combining inequalities was first discussed by Shostak [17] and used in all succeeding works. A TVPI system was represented by a multigraph where nodes correspond to variables and inequalities correspond to edges. Each set of inequalities that can be combined as described above to yield a new inequality forms an *admissible* path in this multigraph [17]. We find it more convenient to consider a slightly different graph (the *associated graph* of a system), in which there is an exact correspondence between directed paths and new inequalities that can be obtained as above. cycles in the associated graph yield bounds on variables. Obviously, the new inequalities and bounds that are obtained by combining inequalities are implied by the original system. Shostak proved that the converse is also true: the tightest bounds on variables that can be obtained from simple cycles and paths define the bounding box of the feasible region. The current paper, like much previous work (see [3, 2, 15, 17]), makes use of Shostak's result.

2.1. The associated graph

We represent a TVPI system by a set of n intervals and a directed multigraph with $2n$ nodes and $2m$ edges. The graph is a natural representation of the system, where variables correspond to vertices and two-variable inequalities to edges between vertices of the participating variables. A formal definition of the associated graph follows.

Definition 2.1 Suppose we are given a TVPI system as in Definition 1.1. Without loss of generality, assume that the inequalities with a single variable (*bounds*) are summarized in the form of intervals $S_i = [a_i, b_i]$ ($-\infty \leq a_i \leq b_i \leq \infty$, $i = 1, \ldots, n$). We consider a directed multigraph $G = (V, E)$ as follows. For each variable x_i, there are two vertices in G associated with x_i, namely, $V = \overline{V} \cup \underline{V}$ where $\overline{V} = \{\overline{v}_i | \ i = 1, \ldots, n\}$ and $\underline{V} = \{\underline{v}_i | \ i = 1, \ldots, n\}$. For $u \in V$, we define $u^{-1} \in V$ as follows. If $u = \underline{v}_i \in \underline{V}$, then $u^{-1} = \overline{v}_i$, and if $u = \overline{v}_i \in \overline{V}$, then $u^{-1} = \underline{v}_i$. The edges of G are associated with the inequalities that involve exactly two variables. Note that each such inequality can be written in the form

$$\gamma x_i \leq \alpha x_j + \beta$$

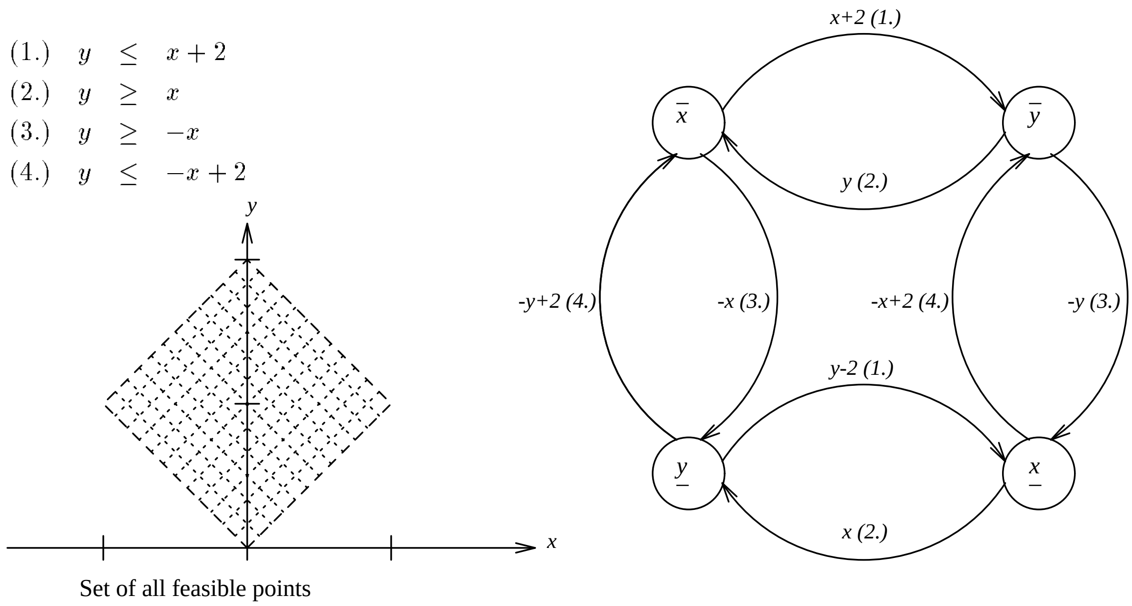

Figure 2: An example of a system of inequalities and the associated graph

where either $\gamma = 1$ and $\alpha \neq 0$ or $\gamma = -1$ and $\alpha > 0$. Each such inequality is represented by two edges, where each edge e is labeled with a certain linear function f_e as follows:

i. If $\alpha > 0$ and $\gamma = 1$, we have an edge $e = (\overline{v}_j, \overline{v}_i)$ labeled $f_e(x) = \alpha x + \beta$ and an edge $e^{-1} = (\underline{v}_i, \underline{v}_j)$ labeled $f_{e^{-1}}(x) = \frac{1}{\alpha}x - \frac{\beta}{\alpha}$.

ii. If $\alpha < 0$ and $\gamma = 1$, we have an edge $e = (\underline{v}_j, \overline{v}_i)$ labeled $f_e(x) = \alpha x + \beta$ and an edge $e^{-1} = (\underline{v}_i, \overline{v}_j)$ labeled $f_{e^{-1}}(x) = \frac{1}{\alpha}x - \frac{\beta}{\alpha}$.

iii. If $\alpha > 0$ and $\gamma = -1$, we have an edge $e = (\overline{v}_j, \underline{v}_i)$ labeled $f_e(x) = -\alpha x - \beta$ and an edge $e^{-1} = (\overline{v}_i, \underline{v}_j)$ labeled $f_{e^{-1}}(x) = -\frac{1}{\alpha}x - \frac{\beta}{\alpha}$.

See Figure 2 for an example of such a system and the associated graph. See Figure 3 for the associated graph of the system shown in Figure 1. We assume throughout this paper that a TVPI system is given by the associated graph and set of intervals.

For any one-to-one function f let f^{-1} denote the inverse function. In particular, if $f(x) = \alpha x + \beta$ and $\alpha \neq 0$, then $f^{-1}(x) = \frac{1}{\alpha}x - \frac{\beta}{\alpha}$. Note that for all $e \in E$, $f_{e^{-1}} = f_e^{-1}$.

Let G be as in Definition 2.1. A linear function associated with an edge (u, w) corresponds to an inequality in the original system: suppose that $u \in \{\underline{v}_i, \overline{v}_i\}$; if $w = \overline{v}_j$ the inequality is $x_j \leq f_e(x_i)$, and if $w = \underline{v}_j$ the inequality is $x_j \geq f_e(x_i)$. We define the linear function associated with a directed path p from u to w. The corresponding two variable inequality that results from treating the path as an edge (u, w), is implied by the original system.

Definition 2.2 Let $p = (e_1, \ldots, e_k)$ be a (directed) path in G.

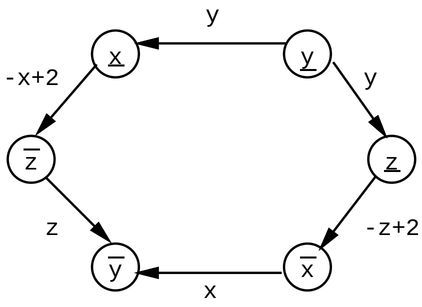

Figure 3: The associated graph of the TVPI system of Figure 1

i. For any path p, we define a linear function f_p, where $f_p = f_{e_k} \circ \cdots \circ f_{e_1}$. Note that if both ends of p lie either in $\overline{V}$ or in $\underline{V}$ then f_p is increasing. Otherwise, f_p is decreasing.

ii. We denote by p^{-1} the path $p = (e_k^{-1}, \ldots, e_1^{-1})$. Note that $f_{p^{-1}} = f_p^{-1}$ and hence the two paths p, p^{-1} correspond to the same inequality.

Consider, for example, the directed path from $\underline{y}$ to $\overline{z}$ in the graph of Figure 3. The linear function associated with this path is $f = 2 - y$, and the corresponding inequality is $z \leq 2 - y$.

In particular, Definition 2.2 applies to cycles (closed paths). Cycles play a special role since they give a relation that involves a single variable, from which a bound on this variable can be deduced. This is formalized in the following definition.

Definition 2.3 We refer to a path that starts and ends at vertices from $\{v, v^{-1}\}$ as a *cycle*. A path from v to v is a *closed cycle*, and to a path from v to v^{-1} is an *open cycle*.

i. Let c be a cycle starting at one of the vertices $\overline{v}_i$ and $\underline{v}_i$, and ending at a vertex $\overline{v}_i$ (resp., $\underline{v}_i$). Let f_c be the associated linear function. The bound on x_i implied by the cycle c follows from the inequality $x \leq f_c(x)$ (resp., $x \geq f_c(x)$). Obviously, the implied inequality must hold for all feasible points. Note that if a cycle c starts at $\overline{v}_i$ and ends at $\underline{v}_i$ (resp., starts at $\underline{v}_i$ and ends at $\overline{v}_i$), then f_c is decreasing. Hence, c implies a lower bound (resp., an upper bound) on x_i.

ii. We say that the cycle c *contradicts* a value ξ_i of $\overline{v}_i$ (resp., $\underline{v}_i$) if the bound implied by c is of the form $x_i \geq \alpha$ (resp., $x_i \leq \alpha$) whereas $\xi_i < \alpha$ (resp., $\xi_i > \alpha$).

Consider, as an example, the graph of Figure 2 and the cycle that consists of the path from $\underline{x}$ to $\overline{x}$. The function associated with this cycle is $f = -x + 2$ and the bound implied by the cycle is $x \leq -x + 2$ (i.e., $x \leq 1$).

2.2. Pushing bounds along edges

We maintain numerical values associated with the vertices of G: the value associated with $\overline{v}_i$ (resp., $\underline{v}_i$) is denoted by $\overline{x}_i$ (resp., $\underline{x}_i$). Intuitively, $\overline{x}_i$ and $\underline{x}_i$ correspond to upper and lower bounds, respectively, on the value of the variable x_i. We say that ξ is *tighter* than μ as a value of $\overline{x}_i$ (resp., $\underline{x}_i$), if $\xi < \mu$ (resp., $\xi > \mu$). We introduce the following operation that considers an edge $e = (u, v)$ and updates the bound associated with v, as implied by the inequality associated with e and the bound associated with u.

Definition 2.4 A *push* through an edge e is defined as the following operation:

i. If $e = (\overline{v}_j, \overline{v}_i)$, $\overline{x}_i \leftarrow \min\{\overline{x}_i, f_e(\overline{x}_j)\}$.

ii. If $e = (\underline{v}_j, \underline{v}_i)$, $\underline{x}_i \leftarrow \max\{\underline{x}_i, f_e(\underline{x}_j)\}$.

iii. If $e = (\overline{v}_j, \underline{v}_i)$, $\underline{x}_i \leftarrow \max\{\underline{x}_i, f_e(\overline{x}_j)\}$.

iv. If $e = (\underline{v}_j, \overline{v}_i)$, $\overline{x}_i \leftarrow \min\{\overline{x}_i, f_e(\underline{x}_j)\}$.

A push is said to be *essential* if it actually modifies the value of $\underline{x}_i$ or $\overline{x}_i$.

Proposition 2.5 *Suppose $\underline{x}_i$ and $\overline{x}_i$ $(i = 1, \ldots, n)$ are initialized to some values and let X be the set of vectors $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ that (i) satisfy all the given inequalities and (ii) $\underline{x}_i \le x_i \le \overline{x}_i$ $(i = 1, \ldots, n)$. The set X is invariant under pushes.*

Proof: Consider any $\boldsymbol{x} \in X$ and a push through an edge $e = (\overline{v}_j, \overline{v}_i)$. (The arguments for the other cases are similar.) Since $\boldsymbol{x}$ is a feasible solution we have $x_i \le f_e(x_j)$. Note that f_e is an increasing function when both ends of e lie either in $\overline{V}$ or in $\underline{V}$. Hence, from $x_j \le \overline{x}_j$, it follows that $x_i \le f_e(\overline{x}_j)$. Since $x_i \le \overline{x}_i$, we have $x_i \le \min\{\overline{x}_i, f_e(\overline{x}_j)\}$. ∎

It is easy to see that a vector $\boldsymbol{x}$ solves a given TVPI system if and only if (i) $x_i \in S_i$ $(i = 1, \ldots, n)$ and (ii) the set of values $\overline{x}_i = \underline{x}_i = x_i$ $(i = 1, \ldots, n)$ is invariant under pushes.

We define a new operation that amounts to assigning simultaneously at all vertices, the tightest values that could be obtained by applying pushes (see Definition 2.4).

Definition 2.6 Consider the graph G with some set of values at the vertices, $\underline{x}_i = \underline{x}_i^k$, $\overline{x}_i = \overline{x}_i^k$ $(i = 1, \ldots, n)$. A *push phase* on G is assigning, at the respective vertices, the set of values $\underline{x}_i^{k+1}$, $\overline{x}_i^{k+1}$ $(i = 1, \ldots, n)$ defined as follows:

$$\begin{aligned}
\overline{x}_i^{k+1} \quad &\leftarrow \quad \min\left\{\overline{x}_i^k, \min_{e \in \mathrm{in}\{\overline{v}_i\}} \left\{f_e(\overline{x}_j^k) \text{ if } e = (\overline{v}_j, \overline{v}_i), \text{ or } f_e(\underline{x}_j^k) \text{ if } e = (\underline{v}_j, \overline{v}_i)\right\}\right\} \\
\underline{x}_i^{k+1} \quad &\leftarrow \quad \max\left\{\underline{x}_i^k, \max_{e \in \mathrm{in}\{\underline{v}_i\}} \left\{f_e(\overline{x}_j^k) \text{ if } e = (\overline{v}_j, \underline{v}_i), \text{ or } f_e(\underline{x}_j^k) \text{ if } e = (\underline{v}_j, \underline{v}_i)\right\}\right\}
\end{aligned}$$

Performing a *push phase* is computationally similar to a single phase of the Bellman-Ford shortest path algorithm. It takes $O(m)$ time sequentially; $O(\log\log n)$ time using $O(m)$ processors or $O(1)$ expected time using $O(m)$ processors by a randomized algorithm on a CRCW PRAM; and $O(\log m)$ deterministic time using $O(m)$ processors on a CREW PRAM. (see [13].)

We use push phases as a basic operation in algorithms presented later. When repeated applications of push phases are performed, the initial set of values is denoted by $\underline{x}_i = \underline{x}_i^0$, $\overline{x}_i = \overline{x}_i^0$ $(i = 1, \ldots, n)$ and the respective values of $\underline{x}_i$ and $\overline{x}_i$ after the termination of the kth push phase are denoted by $\underline{x}_i^k$ and $\overline{x}_i^k$.

Definition 2.7 Suppose that a sequence of push phases is performed. Consider pairs consisting of a value and a vertex of the form $(\underline{x}_j^k, \underline{v}_j)$ or $(\overline{x}_j^k, \overline{v}_j)$ (where $k \geq 0$ and $1 \leq j \leq n$). The *predecessor* of a pair (ξ, v) is the pair (ξ', v') such that $e = (v', v)$ is the edge through which the essential push determined the value ξ, and $\xi' = f_e^{-1}(\xi)$ is the corresponding value at v'. If the predecessor is not uniquely defined we choose arbitrarily among the qualified pairs. A pair (ξ, v) does not have a predecessor if and only if ξ is the initial value at v.
The *essential path* of a pair (ξ, v) as above consists of the sequence of pairs $(\xi_1, v_1), \ldots, (\xi_\ell, v_\ell)$ and the corresponding sequence of edges $e_1, \ldots, e_{\ell-1}$ such that (i) (ξ_i, v_i) is the predecessor of (ξ_{i+1}, v_{i+1}) and e_i is the edge through which the essential push occurs $(i = 1, \ldots, \ell - 1)$, (ii) $(\xi_\ell, v_\ell) = (\xi, v)$, and (iii) (ξ_1, v_1) does not have a predecessor.

It is easy to see that the essential paths that correspond to $(\underline{x}_i^\ell, \underline{v}_i)$ and $(\overline{x}_i^\ell, \overline{v}_i)$ $(i = 1, \ldots, n)$ can be found within the same time bounds of performing ℓ push phases, simply by keeping track of all essential push operations.

We extend the notion of a push through an edge to directed paths. Let $p = (e_1, \ldots, e_k)$ be a (directed) path in G. A *push along* p is defined to be the composition of k successive pushes, through the edges $e_1, \ldots, e_k$. It is easy to see that for every $\ell \geq 0$ and a vertex v, after a sequence of ℓ push phases, the value at v is exactly the tightest between the initial value at v and the values obtained by pushing corresponding initial values at vertices along paths of size at most k that are going into v.

2.3. Properties of the feasible region

We discuss the relation between the bounds of the interval of feasible values of a variable and the tightest bounds implied by directed cycles. We first give the following definitions.

Definition 2.8

i. For each variable x_i, let $[\underline{x}_i^*, \overline{x}_i^*]$ be the set of values in $[a_i, b_i]$ that are not contradicted by any simple cycle. Note that the two cycles c and c^{-1} imply the same bound. It follows that in order to find $\overline{x}_i^*$ (resp., $\underline{x}_i^*$), it suffices to consider cycles ending at $\overline{v}_i$ (resp., $\underline{v}_i$).

ii. For $1 \leq i \leq n$ denote by $[x_i^{\min}, x_i^{\max}] \subseteq [\underline{x}_i^*, \overline{x}_i^*]$ the largest interval such that (1) for all $1 \leq j \leq n$ and for all simple paths p from a vertex $\overline{v}_j$ (resp., $\underline{v}_j$) to $\overline{v}_i$, we have $x_i^{\max} \leq f_p(\overline{x}_j^*)$ (resp., $x_i^{\max} \leq f_p(\underline{x}_j^*)$) and, (2) for all $1 \leq j \leq n$ and for all simple paths p from a vertex $\overline{v}_j$ (resp., $\underline{v}_j$) to $\underline{v}_i$ we have $x_i^{\min} \geq f_p(\overline{x}_j^*)$ (resp., $x_i^{\min} \geq f_p(\underline{x}_j^*)$).

iii. An interval $I \subset R$ is *infeasible* with respect to the variable x_i if $I \cap [x_i^{\min}, x_i^{\max}] = \emptyset$. A value $\overline{x}_i$ (resp., $\underline{x}_i$) is *infeasible* if $\overline{x}_i < x^{\min}$ (resp., $\underline{x}_i > x^{\max}$).

iv. A value $\overline{x}_i$ (resp., $\underline{x}_i$) is *consistent with an interval* $[a, b]$ if $\overline{x}_i \geq b$ (resp., $\underline{x}_i \leq a$). A value $\overline{x}_i$ (resp., $\underline{x}_i$) is *consistent* if $\overline{x}_i \geq x_i^{\max}$ (resp., $\underline{x}_i \leq x_i^{\min}$).

Remark 2.9 When the values $\overline{x}_i^*$ and $\underline{x}_i^*$ $(i = 1, \ldots, n)$ are given, we can compute $x_i^{\min}$ and $x_i^{\max}$ as follows. Initialize the values at the vertices to be $\overline{x}_i^0 = \overline{x}_i^*$ and $\underline{x}_i^0 = \underline{x}_i^*$ $(i = 1, \ldots, n)$. Perform $2n$ push phases. It follows from Definition 2.8 part ii that $x_i^{\min} = \underline{x}_i^n$ and $x_i^{\max} = \overline{x}_i^n$. This procedure runs in $O(mn)$ time and is used in Megiddo's algorithm [15].

For example, the TVPI system of Figure 2 has $[\underline{x}^*, \overline{x}^*] = [-1, 1] = [x^{\min}, x^{\max}]$, and $[\underline{y}^*, \overline{y}^*] = [0, 2] = [y^{\min}, y^{\max}]$. The TVPI system of Figure 3 has $[\underline{x}^*, \overline{x}^*] = [\underline{z}^*, \overline{z}^*] = [-\infty, +\infty]$, $[\underline{y}^*, \overline{y}^*] = [y^{\min}, y^{\max}] = [0, 1]$, and $[x^{\min}, x^{\max}] = [z^{\min}, z^{\max}] = [0, 2]$.

The following key observation is due to Shostak:

Proposition 2.10 [17] *If the system is feasible, then for all $1 \leq i \leq n$, the interval $[x_i^{\min}, x_i^{\max}]$ is the projection of the set of solutions on the x_i-axis. Otherwise, for some $1 \leq i \leq n$, $x_i^{\min} > x_i^{\max}$*

The proof follows from considering the possible structure of minimal sets of inequalities that imply a bound on a variable. Proposition 2.10 and the procedure described in Remark 2.9 assert that in order to find a feasible solution it suffices to consider all *simple* cycles in the associated graph. Shostak presented an exponential time algorithm that essentially examines all directed simple cycles [17].

The following corollary states that if two bounds are feasible separately but not simultaneously, then there exists a simple path that implies an inequality which asserts that the bounds are not simultaneously feasible.

Corollary 2.11 *Let S be a set of TVPI constraints, and $s_1 \in \{x_i \geq \alpha, x_i \leq \alpha\}$ $s_2 \in \{x_j \geq \beta, x_j \leq \beta\}$ be two bounds. Let $u_i = \overline{v}_i$ if s_1 is an upper bound, and $u_i = \underline{v}_i$ otherwise; and let $u_j = \overline{v}_j$ if s_2 is an upper bound, and $u_j = \underline{v}_j$ otherwise. Suppose $S \cup \{s_1\}$, and $S \cup \{s_2\}$ are feasible systems, but $S \cup \{s_1, s_2\}$ is not feasible. Under the above conditions the following holds. There exists a simple path p from u_i to u_j^{-1} such that (i) $f_p(\alpha) < \beta$, if s_2 is a lower bound, and (ii) $f_p(\alpha) > \beta$, if s_2 is an upper bound. Moreover, if p is the tightest (simple) path from u_i to u_j^{-1} relative to $x_i = \alpha$, then if s_2 is an upper (resp., lower) bound, $x_j = f_p(\alpha)$ is the smallest (resp., largest) value of x_j subject to $S \cup \{s_1\}$.*

Proof: Suppose that both s_1, s_2 are upper bounds (the proof for the other cases is similar). Consider $a_j \equiv x_j^{\min}$ under the system $S \cup \{s_1\}$ and $a'_j \equiv x_j^{\min}$ under the system S. We have $a'_j \leq \beta$ and $a_j > \beta$. By definition, $x_j^{\min}$ is either implied (a) by a cycle or a cycle and a path (i.e., implied only by inequalities that involve exactly two variables), or (b) a (possibly empty) directed simple path (i.e., by a single variable inequality). Since the system S contains the same two variable inequalities as $S \cup \{s_1\}$, a_j is determined by a system of type (b) consisting of the bound s_1 and a simple path p. ∎

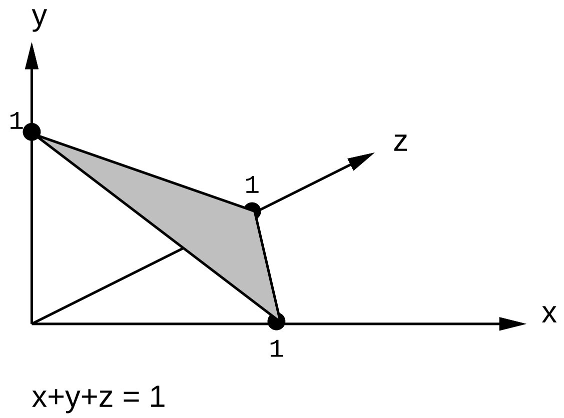

Figure 4: An example of a non-TVPI polyhedron

2.4. Characterizations of TVPI polyhedra

Proposition 2.10 relates the bounding box of the feasible region to the associated graph. We give several characterizations for polyhedra that comprise sets of solutions of TVPI systems (*TVPI polyhedra*). Polyhedra that can not be expressed as such are *non-TVPI polyhedra.*

Proposition 2.12 *Consider a polyhedron $P \subset R^n$. The following statements are equivalent:*

i. *The polyhedron P is a TVPI polyhedron.*

ii. *For a bounded convex set $P' = P \cap B'$, where B' is a box, denote $a_i = \min_{\boldsymbol{x} \in P'} x_i$, $b_i = \max_{\boldsymbol{x} \in P'} x_i$ $(i = 1, \ldots, n)$, and denote by $B = \mathsf{X}_{i=1}^n [a_i, b_i]$*[3] *the tightest bounding box for P'. Every P' as above is such that $\frac{1}{2}\boldsymbol{a} + \frac{1}{2}\boldsymbol{b} \in P'$.*

iii. *Every set S of bounds (i.e., conditions of the form $x_i \geq \alpha$, $x_i \leq \alpha$, $x_i = \alpha$) has the following property. Denote by $B_S \in R^n$ the set of all feasible vectors for S. $P \cap B_S = \emptyset$ if and only if there exists a set $S' \subset S$, where $|S'| \leq 2$ such that $P \cap B_{S'} = \emptyset$.*

iv. *Same as property iii, but S contains only equalities.*

In analysis done later in this paper we make use of the fact that property i implies iii. The other characterizations are presented for the sake of interest and completeness.

Proof: We first show that property i implies properties ii–iv.

i $\Rightarrow$ ii Was proved by Lueker, Megiddo, and Ramachandran [14].

[3] $\mathsf{X}_{i=1}^n [a_i, b_i]$ denotes the Cartesian product of the intervals $[a_i, b_i]$ $(i = 1, \ldots, n)$.

i $\Rightarrow$ iii Suppose P is a TVPI polyhedron, let S_P be a set of TVPI constraints that define P and let $P_i = [\hat{x}_i^{\min}, \hat{x}_i^{\max}]$ $(i = 1, \ldots, n)$ be the corresponding projections. Let S be a set of bounds such that the combined TVPI system $S_P \cup S$ is infeasible. Note that the associated graphs of S_P and $S_P \cup S$ are identical. Denote by $x_i^{\min}, x_i^{\max}$ $(i = 1, \ldots, n)$ the bounds defined by the system $S \cup S_P$. By definition, each of $x_i^{\min}, x_i^{\max}$ $(i = 1, \ldots, n)$ that does not coincide with the respective end point of the interval P_i, is determined by a single bound from S and a simple path. Proposition 2.10 implies that the system $S_P \cup S$ is infeasible if and only if for some variable x_i, $x_i^{\min} > x_i^{\max}$. Hence, at least one of the two bounds $x_i^{\min}, x_i^{\max}$ does not coincide with an end point of P_i. Consider the bounds $S' \subset S_P \cup S$ $(|S'| \leq 2)$ that determine $x_i^{\min} > x_i^{\max}$. Obviously, the system $S_P \cup S'$ is infeasible.

iii $\Rightarrow$ iv Obvious.

It remains to prove that either one of properties ii–iv implies property i. Note that properties i–iv are invariant under scalings and translations of coordinates.

Figure 4 provides an example of a non-TVPI polyhedron. It is easy to see that for this polyhedron property ii does not hold (consider the bounding box $0 \leq x \leq 1$, $0 \leq y \leq 1$, $0 \leq z \leq 1$), and property iv does not hold (consider $S = \{x = 0, y = 0, z = 0\}$).

Assume that P is a non-TVPI polyhedron. If P is full dimensional, there exists at least one face F such that the equation of the hyperplane H constituting the affine hull of F involves more than two variables. Without loss of generality (by permuting and scaling coordinates) we can assume that the equation is $\sum_{i=1}^{\ell} x_i = c$ (where $\ell \geq 3$ and $c \in R$), and for all points $\boldsymbol{x} \in P$, $\sum_{i=1}^{\ell} x_i \leq c$. Note that $P \cap H$ is full-dimensional in H. Define $\hat{\boldsymbol{x}}$ to be some point in $\operatorname{rel\,int} P \cap H$. If P is not full dimensional, consider a set of equations defining the affine hull of P. Consider a representation of P by the equalities $x_i = f_i(x_1, \ldots, x_{n'})$ $(i \geq n')$ for some $n' < n$ and a set of inequalities on $x_1, \ldots, x_{n'}$. Denote by $P' \subset R^{n'}$ the polyhedron defined by these inequalities. The representation is chosen such that P' is full-dimensional. If one of the equations involves more than two variables assume (by permuting and scaling coordinates) that it has the form $x_\ell = -\sum_{i=1}^{\ell-1} x_i + c$. Define $(\hat{x}_1, \ldots, \hat{x}_\ell)$ to be some point in $\operatorname{rel\,int} P'$. If none of the equations involves more than two variables, then at least one face F of P' involves more than two variables. Using similar considerations to the ones used above, we assume without loss of generality that the equation defining the affine hull of F is $\sum_{i=1}^{\ell} x_i = c$, and for all points $\boldsymbol{x} \in P$, $\sum_{i=1}^{\ell} x_i \leq c$. Note that $P' \cap H$ is full-dimensional in H. Define $\hat{\boldsymbol{x}}$ to be some point in $\operatorname{rel\,int} P' \cap H$.

iv $\Rightarrow$ i By definition of $\hat{\boldsymbol{x}}$, there exists a small enough $\epsilon \geq 0$ such that for $i = 1, \ldots, \ell$, there exists $\boldsymbol{x}^i \in P$ such that $x_i^i = \hat{x}_i - (\ell - 1)\epsilon$, and $x_j^i = \hat{x}_j + \epsilon$ $(j = 1, \ldots, i-1, i+1, \ldots, \ell)$. Consider the set of equations $S = \{x_j = \hat{x}_j + \epsilon \mid j = 1, \ldots, \ell\}$. Obviously, there is no point in P that satisfies S. On the other hand, there is a point that satisfies every subset of $\ell - 1 \geq 2$ constraints. Hence, property iv does not hold for P.

ii $\Rightarrow$ i By definition of $\hat{\boldsymbol{x}}$, there exists a small enough $\epsilon \geq 0$ such that for $i = 1, \ldots, \ell$, there exists $\boldsymbol{x}^i \in P$ such that $x_i^i = \hat{x}_i + (\ell - 1)\epsilon$, and $x_j^i = \hat{x}_j - \epsilon$ $(j = 1, \ldots, i-1, i+1, \ldots, \ell)$.

Consider the box B defined by the intervals $[\hat{x}_j - \epsilon, \hat{x}_j + (\ell - 1)\epsilon]$ $(i = 1, \ldots, \ell)$. Since $\boldsymbol{x}^i \in B \cap P$, B is a bounding box for $B \cap P$. However, there is no $\boldsymbol{x} \in P$ for which $x_j = \hat{x}_j + (\ell - 2)\epsilon/2$ $(j = 1, \ldots, \ell)$. It follows that property ii does not hold for P.

∎

2.5. Classifying and locating values

We discuss two procedures that are used later in the paper and are based on repetitive applications of push phases and examining essential paths.

One of the procedures solves the following problem.

Problem 2.13 [Locate a value] For a given TVPI system, a number ξ, and a variable x_i, locate ξ with respect to the interval $[x_i^{\min}, x_i^{\max}]$, that is, decide whether (i) $\xi_i < x_i^{\min}$, (ii) $\xi_i > x_i^{\max}$, or (iii) $\xi_i \in [x_i^{\min}, x_i^{\max}]$.

We refer to solving Problem 2.13 as *locating* the value ξ_i. Aspvall and Shiloach [2, 3] proved the following:

Proposition 2.14 *Problem 2.13 can be solved in $O(mn)$ operations.*

Aspvall and Shiloach utilized this result to solve TVPI systems: by solving instances of Problem 2.13 their algorithm conducts a binary search that finds a point $\xi_i \in [x_i^{\min}, x_i^{\max}]$ $(1 \le i \le n)$. They showed that finding such a point ξ_i can be done in $O(mnI)$ operations where I is the number of bits in the binary representation of the input. This yielded their $O(mn^2 I)$ algorithm for TVPI systems: the algorithm performs n steps, where in step i a point $\xi_i \in [x_i^{\min}, x_i^{\max}]$ is found, and the equation $x_i = \xi_i$ is added to the system. If the original system is feasible, then $\boldsymbol{\xi} \in F$.

In Section 3 we present a $\tilde{O}(mn^2)$ deterministic algorithm for solving TVPI systems, that is based on locating values in the time bounds stated in Proposition 2.14. The material contained in the remainder of the current section is needed for achieving the randomized sequential bound of $\tilde{O}(n^3 + mn)$ and parallel time bounds of $\tilde{O}(n)$, and hence, it is not needed for the reading of Section 3. In the rest of the section we first characterize the possible structures of minimal subsystems that certify the infeasibility of values (*"certificates"*). We then classify infeasible values according to the types of their certificates. We also present a generic algorithm that performs a sequence of push phases for a given set of initial values. This algorithm maintains essential paths and tests for cycles in these paths. We apply this algorithm to solve both the value location problem and the *reveal* problem that is defined later.

Certificates of infeasibility. Suppose an interval $I = [a, b]$ is infeasible with respect to x_i (equivalently, either $[-\infty, b]$ or $[a, \infty]$ is infeasible). It follows from Proposition 2.10 that there exists a vertex w, a simple path p from $u \in \{\underline{v}_i, \overline{v}_i\}$ to w, and a disjoint simple cycle c starting at w such that the bound on w implied by c and the path p produce a bound on x_i that does not intersect I. Note that the path p and the cycle c comprise a minimal subsystem that implies the infeasibility of I. Also note that the cycle c may be empty, in which case, the infeasibility follows from the path p and a bound on the variable associated with w. We refer to such a system as a *certificate* for I. If c is a closed cycle we say that the certificate is *closed*. If c is open or empty, we say that the certificate is *open*. When we refer to a certificate, we interchangeably mean the set of edges $E' = p \cup c$, the set E'^{-1} of the reversed edges, or the corresponding set of inequalities.

Classifying values. We classify values of variables according to the types of their certificates. Consider an interval I with respect to x_i. If I has a closed certificate we say that I is *strongly infeasible*. If only open certificates exist, we say that I is *weakly infeasible*. Otherwise, if no certificate exists, $I \cap [\boldsymbol{x}_i^{\min}, x_i^{\max}] \neq \emptyset$ and we say that I is feasible. In particular these definitions apply when I consists of a single point. For $1 \leq i \leq n$, we denote by $[x_i^{\min *}, x_i^{\max *}]$ the set of all feasible and weakly infeasible values of x_i. Note that $x_i^{\min *}$, $x_i^{\max *}$ $(1 \leq i \leq n)$ are independent of the single variable inequalities.

We motivate distinguishing strongly and weakly infeasible values. We shall see that weakly infeasible values can be identified by applying push phases (at most $2n$ times) until for some $1 \leq i \leq n$ we have $\underline{x}_i > \overline{x}_i$. Strongly infeasible values are trickier to identify. Merely applying push phases is not enough. We need to keep track of essential paths, detect cycles in these paths, and compute the corresponding bounds.

We extend the definitions of the concepts above from intervals to values at vertices of G: A property (having a closed/open certificate, weak/strong infeasibility) holds for a value ξ at a vertex $\underline{v}_i$ (resp., $\overline{v}_i$) if it holds for the interval $[\xi, \infty]$ (resp., $[-\infty, \xi]$) with respect to x_i.

We present two corollaries of Propositions 2.5 and 2.10. Consider a feasible TVPI system and the associated graph. Suppose a sequence of push phases is performed.

Corollary 2.15 *If the values at the vertices are initially consistent (see Definition 2.8), they remain consistent after any number of push operations.*

Corollary 2.16 *Consider a vertex $u = \underline{v}_i$ (resp., $u = \overline{v}_i$) and a value $\xi = \underline{x}_i^\ell$ (resp., $\xi = \overline{x}_i^\ell$) for some ℓ. Let (ξ_j, v_j) $(j = 1, \ldots, k \leq \ell)$, where $(\xi_k, v_k) \equiv (\xi, u)$, be the pairs comprising the essential path (see Definition 2.6) associated with (ξ, u). If ξ is infeasible (resp., strongly infeasible) at u, then each of the pairs (ξ_j, v_j) $(1 \leq j < k)$ is such that ξ_j is an infeasible (resp., strongly infeasible) value of v_j.*

Consider an essential path that contains a closed cycle (a vertex appears more than once). The following proposition enables us to extract information from the cycle. It states that either all values following the start pair of the cycle are consistent (the path does not add information) or all values preceding the last pair of the cycle are strongly infeasible (an infeasible value is detected). Moreover, by considering the updating cycle we can determine which of the two situations occurs.

Proposition 2.17 *Let ξ be a value at $u = \overline{v}$ (resp., $u = \underline{v}$). Suppose a closed cycle c, that starts and ends at u, is such that $f_c(\xi) < \xi$ (resp., $f_c(\xi) > \xi$). Then, either (i) ξ is strongly infeasible at u and the cycle c is a closed certificate, or (ii) if $F \neq \emptyset$, ξ is consistent at u.*

Proof: The bound implied by the cycle c is $f_c(y) \geq y$ (resp., $f_c(y) \leq y$). It suffices to show that it holds either (i) only for values y such that $y > \xi$ (resp., $y < \xi$), and hence ξ is strongly infeasible, or (ii) only for values y such that $y \leq \xi$ (resp., $y \geq \xi$) and hence, ξ is consistent. The function $f_c(y) - y$ is linear and thus can change sign only once. It cannot be a nonpositive (resp., nonnegative) constant, since this contradicts $f_c(\xi)$ being tighter than ξ. If the function is a negative (resp., positive) constant, the cycle implies that the system is infeasible (and therefore ξ is contradicted by the cycle). Otherwise, consider the value y^* such that $f_c(y^*) = y^*$. The bound implied by c is either $[-\infty, y^*]$ or $[y^*, \infty]$. Since ξ does not satisfy the bound inequality, it is not in this interval. ∎

The following algorithm is applied to a set of values $\underline{x}_i^0$, $\overline{x}_i^0$ (for $1 \leq i \leq n$). The algorithm performs a sequence of push phases, and keeps track of essential paths. When the algorithm detects a non simple essential path, it either terminates or discards the path. If the algorithm terminates as a result of such a path, a closed certificate is found for one of the initial values. A vertex v is *active* at a particular point in the execution of the algorithm if the current value at v resulted from an essential path that is not discarded.

Algorithm 2.18 [Generic applications of push phases]

i. For $i = 1, \ldots, n$, initialize the values at $\underline{v}_i$ and $\overline{v}_i$ as $\underline{x}_i^0$ and $\overline{x}_i^0$, respectively.

ii. For $k = 1, \ldots, 2n$ do steps iii–vii. If the algorithm did not terminate, conclude that none of the initial values is strongly infeasible, and stop.

iii. Perform a push phase. Denote the values at the vertices after the kth push phase by $\underline{x}_j^k$, $\overline{x}_j^k$ $(1 \leq k \leq 2n)$.
Keep track of all essential push operations that result from active vertices. If there are no such essential pushes or no active vertices, stop and conclude that none of the initial values is strongly infeasible.
(We assume that ambiguities about the edge that carried the essential push are resolved consistently according to some ordering on the edges.)

iv. If $k \neq 2^j$ for all integers j and $k \neq 2n$, go to step iii (next iteration).

v. Optional: If for some j, $\underline{x}_j^k > \overline{x}_j^k$, then stop.

vi. For each active vertex v, let p_v be the essential path that corresponds to the last tightening of the value at v.
If p_v contains a closed cycle (i.e., some vertex appears in more than one pair), execute the following subroutine (step vii). After considering all vertices, go to step iii (next iteration).

vii. **Subroutine: Examine p_v.**
Consider the last simple closed cycle c on p_v. Denote by u the vertex where c starts and ends. Denote by ξ the value associated with u at its first occurrence on c. Check whether the bound implied by c contradicts ξ. If there is no contradiction, ξ is consistent (see Proposition 2.17), discard all essential paths originating from the first pair (u, ξ) of the cycle c (all correspond to consistent values according to Corollary 2.15); return. Otherwise, if c contradicts ξ, ξ is strongly infeasible. It follows from Corollary 2.16 that so are all values along the path p_v prior to the pair $(u, f_c(\xi))$ (the last occurrence of u on p_v). In particular, the initial value at the first vertex w of p_v is strongly infeasible. Terminate the execution of the algorithm.

Complexity. The underlying computation of the algorithm is the Bellman-Ford single-source shortest path computation [8], where we maintain information needed to construct the paths and test for cycles.
The algorithm terminates after $\ell \leq 2n$ iterations, in which case, it requires $O(m\ell)$ sequential time, and $O(\ell)$ expected time (or, $O(\ell \log\log m)$ deterministic time) using $O(m\ell)$ work on a CRCW PRAM.

If the algorithm terminates at step vii, it finds a closed certificate for the initial value at w. If the algorithm terminates during the ℓth iteration, the closed certificate found is of size $\theta(\ell)$.

Proposition 2.19 *Consider an execution of Algorithm 2.18, where step v is skipped.*

i. At least one of the initial values is strongly infeasible if and only if the algorithm terminates at step vii.

ii. Suppose that exactly one of the initial values, ξ at the vertex w, is strongly infeasible. The algorithm terminates at step vii with the same closed certificate regardless of the initial values at other vertices.

Proof: We first prove part i. The "if" direction is immediate. Suppose an initial value ξ at a vertex w has a closed certificate consisting of a path p of length $|p|$ to u_0 and a cycle c of length r that starts and ends at u_0. After at most $|p|$ iterations, the value at u_0 is at least as tight as $f_p(\xi)$ and hence, has a closed certificate consisting of c. Similarly, after $|p| + r$ iterations the same holds for all the vertices of c. Let $e_0, \ldots, e_r$ and $u_0, \ldots, u_r$, respectively, be the edges and vertices of c. We claim that all iterations for which $k \geq |p|$ result in an update of at least one of the values at $u_0, \ldots, u_r$. Assume the claim is true.

Assume to the contrary that the algorithm did not terminate at step vii. Therefore, it performs $2n$ iterations. It follows that at least one of the paths p_{u_j} $(1 \leq j \leq r)$ is of length $2n$, and hence, contains a closed cycle. Since the value at u_j is strongly infeasible, it follows from Corollary 2.16 that w or another vertex with a strongly infeasible initial value is the first vertex of p_{u_j}. It follows from Proposition 2.17 that each cycle in p_{u_j} contradicts the value at the start vertex. Hence, the algorithm terminates at step vii.

What remains is to prove the claim. Assume the contrary. Let $\xi_0, \ldots, \xi_r$ be the respective values at $u_0, \ldots, u_r$. In particular, none of the edges e_k $(0 \leq k \leq r)$ had an essential push. Hence, for all $j = 1, \ldots, r$, $f_{e_j}(\xi_j)$ is not tighter than $\xi_{j+1 \bmod r}$. It follows that $f_{e_1 \cdots e_r}(\xi_0) = f_c(\xi_0)$ is no more tight than ξ_0. This is a contradiction for c being a closed certificate for ξ_0.

We prove part ii. Consider an initialization where ξ at w is the only strongly infeasible value. It follows from part i that the algorithm terminates in step vii, and finds an essential path p that starts at w and terminates in a closed cycle. All values along the path are strongly infeasible. Suppose that an initial path starting at a vertex other than w updates a value at a vertex in p. It follows from Corollary 2.16 that the updated value can not be tighter or as tight as the value resulting from p. The proof follows. ∎

Definition 2.20 Suppose ξ is a strongly infeasible value at a vertex w. Consider an execution of Algorithm 2.18 where step v is skipped and ξ at w is the only strongly infeasible initial value (see Proposition 2.19 part ii). We refer to the unique closed certificate found by such an execution as *the certificate* of ξ.

Remark 2.21 Consider a run of the algorithm where step v is performed.

i. Suppose that the algorithm terminates at step v, where it detects $\underline{x}_j^k > \overline{x}_j^k$. Consider the initial pairs (u, ξ) and (w, μ) of the two essential paths that determined $(\underline{v}_j, \underline{x}_j^k)$ and $(\overline{v}_j, \overline{x}_j^k)$. It is not necessarily true that one of the initial pairs is infeasible. We can conclude, however, that the two corresponding bounds can not be satisfied simultaneously by a feasible vector.

ii. Consider a run of the algorithm where at least one of the initial values has an open certificate of size ℓ that contains a nontrivial cycle. The following is immediate: the algorithm terminates at either step vii or step v within ℓ iterations.

Locating a value. We apply Algorithm 2.18 to solve Problem 2.13.

Consider a value ξ_i. We show how to decide whether or not $\xi_i > x_i^{\max}$ (the case $\xi_i < x_i^{\min}$ is similar).

Algorithm 2.22 [Locate a value with respect to $x_i^{\max}$]
Perform a run of Algorithm 2.18, where step v is enabled, for the following input of values: $\underline{x}_j^0 = a_j$ for $j \neq i$, $\overline{x}_j^0 = b_j$ for $j = 1, \ldots, n$, and $\underline{x}_i^0 = \xi_i$.
Conclude as follows:

- If the algorithm stopped at step iii (no active vertices), determine that $\xi_i \leq x_i^{\max}$.
- If the algorithms stopped at step v, consider the essential paths associated with $\underline{x}_j^k > \overline{x}_j^k$. If in neither path, the initial pair is $(\underline{v}_i, \xi_i)$, conclude that the system is infeasible. Otherwise, conclude that $\xi_i > x_i^{\max}$.
- Suppose the algorithm terminated at step vii. If $w = \underline{v}_i$, determine that $\xi_i > x_i^{\max}$. Otherwise, if $w \neq \underline{v}_i$, the system is infeasible.
- If the algorithm terminated at step ii, determine that $\xi_i \leq x_i^{\max}$.

The correctness follows immediately from Proposition 2.19 part i, and Remark 2.21.

Reveal a strongly infeasible value. We discuss applying Algorithm 2.18 to solve the following problem.

Problem 2.23 [Reveal a strongly infeasible value]
Given are values ξ_i, $a_i \leq \xi_i \leq b_i$ $(i \in I)$ for the respective variables x_i $(i \in I)$, where $I \subset \{1, \ldots, n\}$. Do one of the following:

i. Conclude that all the values are feasible or weakly infeasible.

ii. Find a strongly infeasible value ξ_j and a closed certificate.

Algorithm 2.24 [Reveal]
Perform a run of Algorithm 2.18, where step v is skipped, for the input values: $\overline{x}_i = \underline{x}_i = \xi_i$ $(i \in I)$, and $\overline{x}_i = b_i$, $\underline{x}_i = a_i$ $(i \notin I)$.

The following is a corollary of Proposition 2.19:

Corollary 2.25 *Algorithm 2.24 solves Problem 2.23. If none of the initial values is strongly infeasible, the algorithm requires $O(mn)$ operations. Otherwise, the algorithm terminates after $\ell \leq 2n$ iterations with a closed certificate of size $\theta(\ell)$. If exactly one of the values ξ_i $(i \in I)$ is strongly infeasible, the algorithm terminates with the same closed certificate regardless of the other values.*

3. The basic algorithm

In this section we present a framework for solving TVPI systems. This framework allows us to state an algorithm for TVPI systems in terms of solving instances of Problem 2.13 (locating single values). The framework is stated in Subsection 3.1, and the correctness proof is given in Subsection 3.2. In Subsection 3.3 we reduce solving a TVPI system to locating $O(n(\log^2 n + \log m))$ values. Since locating a single value requires $O(mn)$ time ($O(n \log\log m)$ time using $O(mn)$ work on a CRCW PRAM), we obtain an $O(mn^2(\log^2 n + \log m))$ deterministic algorithm for solving TVPI systems (that runs in $\tilde{O}(n^2)$ time using $O(mn^2(\log^2 n + \log m))$ work on a CRCW PRAM). In Subsection 3.4 we introduce the problem of *locating a pool of values*. A key for further improvements is the reduction of solving a TVPI system to locating $O(\log^2 n + \log m)$ pools of values. A pool of values can be located naively by sequentially locating n single values. In Section 4 we present faster parallel and sequential algorithms for locating a pool, that yield better algorithms for solving TVPI systems.

3.1. The framework

We first describe an idea introduced by Megiddo [15], which is the key in obtaining strongly polynomial time bounds. Consider the associated graph of some TVPI system. Every directed path in the associated graph corresponds to a two-variable inequality. Consider two inequalities that correspond to two paths between the same pair of vertices (v_i, v_j), where $v_i \in \{\overline{v}_i, \underline{v}_i\}$. These inequalities are linear, hence, there exists a number a such that for all $x_i \leq a$ one of the inequalities implies the other, and vice versa for $x_i \geq a$. If we focus only on feasible points $\boldsymbol{x}$ for which $x_i \geq a$ (similarly $x_i \leq a$), one of the inequalities is redundant. The algorithm eliminates paths and simultaneously restricts the feasible region. When "comparing" two paths, the decision about which one to eliminate is done as follows. First, the number a, as above, is computed. The redundant path is determined by locating a with respect to feasible values of x_i (see Problem 2.13). Megiddo applied the above idea in an algorithm that basically performed n single-source shortest path Bellman-Ford type computations, where comparisons amount to locating values. The framework presented here is based on performing an all-pairs shortest path Floyd-Warshall [8] type computation that allows us to apply further ideas.

The following definition formalizes the concept of comparing paths.

Definition 3.1

i. Suppose that p_1 and p_2 are two directed paths from u to v, and from v to w, respectively. Denote by $p_1 p_2$ the path from u to w obtained by concatenating p_1 and p_2.

ii. Suppose that p_1 and p_2 are two paths from u to w. Let $I \subset R$ be an interval. We say that the path p_1 is at least as *tight* as p_2 relative to I (denote it by $p_1 \prec_I p_2$) if either $w \in \underline{V}$ and $f_{p_1}(\xi) \geq f_{p_2}(\xi)$ for all $\xi \in I$, or $w \in \overline{V}$ and $f_{p_1}(\xi) \leq f_{p_2}(\xi)$ for all $\xi \in I$.

iii. Suppose $p_1, p_2, \ldots, p_k$ are paths from u to w. If for some i, $p_i \prec_I p_j$ for all j, we write $p_i = \min_{\prec_I}\{p_1, p_2, \ldots, p_k\}$. Note that when I is a single point, $\min_{\prec_I}$ is well-defined.

The algorithm maintains a set of intervals S_i $(1 \leq i \leq n)$, and a path p_{uw} from u to w for every pair of vertices (u, w). The algorithm runs in $\lceil \log_2 2n \rceil$ phases. During each phase, the paths and the intervals are considered for possible updates. Denote by S_i^k $(1 \leq i \leq n)$ and

p^k_{uw} ($\{u,w\} \subseteq V$) the intervals and paths, respectively, at the beginning of the kth phase. The algorithm has the following properties: (i) The set $\mathsf{X}_{i=1}^n S_i^k$ contains a feasible point, and (ii) the path p^k_{uw} is the tightest path from u to w, of length at most 2^k, relative to the interval S_i^k (where $u \in \{\overline{v}_i, \underline{v}_i\}$).

The algorithm is based on solving instances of the following problem:

Problem 3.2 Given are a graph G and a set of intervals $S_1, \ldots, S_n$ as in Definition 2.1. For every ordered pair $(u,v) \in V \times V$ of vertices we are given $p^1_{uv}, \ldots, p^{k_{uv}}_{uv}$, a collection of directed paths from u to v in G. The goal is to find a set of n intervals $I_1, \ldots, I_n$ and select a path $p^*_{uv} \in \{p^1_{uv}, \ldots, p^{k_{uv}}_{uv}\}$ for every pair (u,v) of vertices, where

i. $F \neq \emptyset \Rightarrow (\mathsf{X}_{i=1}^n I_i) \cap F \neq \emptyset$, and

ii. $u \in \{\underline{v}_i, \overline{v}_i\} \Rightarrow p^*_{uv} = \min_{\prec_{I_i}} \{p^1_{uv}, \ldots, p^{k_{uv}}_{uv}\}$.

We later suggest algorithms for Problem 3.2. The following algorithm uses it as a subroutine. It first initializes the tightest-paths matrix by selecting the tightest edge out of every set of multiple edges (step ii). The rest of the algorithm consists of $\lceil \log 2n \rceil$ update phases of the tightest-paths matrix (step iv). During each phase, the algorithm considers n^2 sets of n paths (one for each pair of vertices). It then selects the tightest path in each set. In the last step, the tightest path matrix is used to compute a feasible vector.

Algorithm 3.3 [Solve TVPI systems]

i. [Initialization] Construct S and G as in Definition 2.1.

ii. [Initialize tightest paths matrix] For each pair of vertices consider the set of paths of length 1, i.e., all the multiple edges. Solve Problem 3.2 relative to these paths. For $(u,v) \in V \times V$: $p^0_{uv} \leftarrow p^*_{uv}$. For $i = 1, \ldots, n$: $S_i \leftarrow S_i \cap I_i$, $S_i^0 \leftarrow S_i$.

iii. For $k = 1, \ldots, \lceil \log_2 2n \rceil$, execute step iv. To continue, go to step v.

iv. For each pair $(u,v) \in V \times V$, consider the set of paths

$$\left\{p^{k-1}_{uv}\right\} \cup \left\{p^{k-1}_{uw} p^{k-1}_{wv} | w \in V \setminus \{u,v\}\right\} .$$

Solve Problem 3.2 relative to these sets of paths.
For $(u,v) \in V \times V$: $p^k_{uv} \leftarrow p^*_{uv}$. For $i = 1, \ldots, n$: $S_i \leftarrow S_i \cap I_i$, $S_i^k \leftarrow S_i$.

v. Denote $p_{\underline{i}\underline{j}} \equiv p^{\lceil \log 2n \rceil}_{\underline{v}_i, \underline{v}_j}$, $p_{\underline{i}\overline{j}} \equiv p^{\lceil \log 2n \rceil}_{\underline{v}_i, \overline{v}_j}$, $p_{\overline{i}\underline{j}} \equiv p^{\lceil \log 2n \rceil}_{\overline{v}_i, \underline{v}_j}$, $p_{\overline{i}\overline{j}} \equiv p^{\lceil \log 2n \rceil}_{\overline{v}_i, \overline{v}_j}$ (for $1 \leq i \leq n$, $1 \leq j \leq n$).
For $i = 1, \ldots, n$:
$S_i \leftarrow S_i \cap \{x | f_{p_{\underline{i}\underline{i}}}(x) \leq x\}$, $S_i \leftarrow S_i \cap \{x | f_{p_{\underline{i}\overline{i}}}(x) \geq x\}$, $S_i \leftarrow S_i \cap \{x | f_{p_{\overline{i}\underline{i}}}(x) \leq x\}$.

vi. For $i = 1, \ldots, n$, compute intervals $S'_i = [a'_i, b'_i]$ as follows (where $[a_j, b_j] = S_j$, for $1 \leq j \leq n$):

$$a'_i \leftarrow \max\left\{a_i, \max_j f_{p_{\overline{j}\underline{i}}}(b_j), \max_j f_{p_{\underline{j}\underline{i}}}(a_j)\right\}$$

$$b'_i \leftarrow \min\left\{b_i, \min_j f_{p_{\overline{j}\overline{i}}}(b_j), \min_j f_{p_{\underline{j}\overline{i}}}(a_j)\right\}$$

vii. Compute a feasible solution $\hat{\boldsymbol{x}}$ as follows, for $i = 1, \ldots, n$:

$$\hat{x}_i \in \left[\max\left\{a'_i, \max_{j<i} f_{p_{\overline{j}\underline{i}}}(\hat{x}_j), \max_{j<i} f_{p_{\underline{j}\underline{i}}}(\hat{x}_j)\right\}, \min\left\{b'_i, \min_{j<i} f_{p_{\overline{j}\overline{i}}}(\hat{x}_j), \min_{j<i} f_{p_{\underline{j}\overline{i}}}(\hat{x}_j)\right\}\right]$$

3.2. Correctness

We prove the following:

i. For $1 \leq i \leq n$, $S_i^0 \supset S_i^1 \supset \cdots \supset S_i^{\lceil \log_2 2n \rceil} \supset S'_i$.

ii. If $\times_{i=1}^n S_i^0$ contains feasible points, then so do $\times_{i=1}^n S_i^k$ (for all $k > 0$) and $\times_{i=1}^n S'_i$.

iii. p_{uw}^k ($u \in \{\underline{v}_i, \overline{v}_i\}$, $w \in V$) is the tightest path (relative to S_i^k) from u to w, of length less than or equal to 2^k.

iv. If the system is feasible, the process described in step vii results in a feasible vector $\hat{\boldsymbol{x}}$.

Claims i and ii follow directly from the statement of the algorithm.

We prove Claim iii. Consider paths p_1, p_2 between the same pair of vertices. Assume they originate at $\{\underline{v}_j, \overline{v}_j\}$. It follows from the definition of $\prec$ that if $p_1 \prec p_2$ relative to S_j^k, then $p_1 \prec p_2$ relative to S_j^{k+1}. Consider paths of length at most 2^{k+1} from u to v. It suffices to show that when $p_{uv} = p_{uw}p_{wv}$ and $p'_{uv} = p'_{uw}p'_{wv}$, if $p_{uw} \prec_I p'_{uw}$ and $p_{wv} \prec_I p'_{wv}$, then $p_{uv} \prec_I p'_{uv}$. The latter is straightforward.

We prove claim iv. We first show that $S'_i \subset [x_i^{\min}, x_i^{\max}]$ ($i = 1, \ldots, n$). We claim that $S_j^{\lceil \log 2n \rceil} \subset [\underline{x}_j^*, \overline{x}_j^*]$. If the latter holds, it follows from Proposition 2.10 that $S'_i \subset [x_i^{\min}, x_i^{\max}]$. Assume the contrary, that is, for some $1 \leq j \leq n$, $S_j^{\lceil \log 2n \rceil} \not\subset [\underline{x}_j^*, \overline{x}_j^*]$. Consider a point $\eta \in S_j^{\lceil \log 2n \rceil} \setminus [\underline{x}_j^*, \overline{x}_j^*]$. Assume that $\eta > \overline{x}_j^*$ (the case where $\eta < \underline{x}_j^*$ is similar). It follows that there exists a simple cycle c, such that either of the following is true:

i. The cycle c, starts and ends at $\overline{v}_j$, and is such that $f_c(\eta) < \eta$.

ii. The cycle c, starts at $\underline{v}_j$ and ends at $\overline{v}_j$, and is such that $f_c(\eta) < \eta$.

Denote $c' = p_{\overline{jj}}$ (for case (i)) and $c' = p_{\underline{j}\bar{j}}$ (for case (ii)). It follows from claim iii that the cycle c' is the tightest simple cycle relative to $S_j^{\lceil \log 2n \rceil}$. Therefore, $f_{c'}(\eta) \leq f_c(\eta)$. On the other hand, due to step v of the algorithm $f_{c'}(\eta) \geq \eta$. This is a contradiction. Thus we have shown that $S'_i \subset [x_i^{\min}, x_i^{\max}]$ $(i = 1, \ldots, n)$.

We conclude the proof of Claim iv. Consider the computation performed in step vii. For $i = 1, \ldots, n$ consider the set of intervals $\hat{S}_\ell^i$ $(1 \leq \ell \leq n)$ defined as follows. If $\ell < i$, $\hat{S}_\ell^i = \{\hat{x}_\ell\}$. Otherwise,

$$\hat{S}_\ell^i = \left[\max\left\{a'_\ell, \max_{j<i} f_{p_{\bar{j}\underline{\ell}}}(\hat{x}_j), \max_{j<i} f_{p_{\underline{j}\underline{\ell}}}(\hat{x}_j)\right\}, \min\left\{b'_\ell, \min_{j<i} f_{p_{\overline{j\ell}}}(\hat{x}_j), \min_{j<i} f_{p_{\underline{j}\bar{\ell}}}(\hat{x}_j)\right\}\right] .$$

We claim that for $i = 1, \ldots, n$ the following holds. For each $\mu \in \hat{S}_j^i$ there exists a feasible vector $\boldsymbol{\xi}$ such that $\xi_j = \mu$ and $\xi_\ell \in \hat{S}_\ell^i$ $(\ell \neq j)$. To conclude the proof of property iv, it suffices to prove the claim. We prove the claim by induction on i.
In the base case $\hat{S}_\ell^1 = S'_\ell$ $(1 \leq \ell \leq n)$. We show that for every $\xi_j \in S'_j$ there exists a feasible solution $\boldsymbol{x}$ such that $x_j = \xi_j$ and $x_\ell \in S'_\ell$ for $\ell \neq j$. Assume the contrary. It follows from Proposition 2.12 that there exist two bounds $s_1 \in \{x_j \geq \xi_j, x_j \leq \xi_j\}$ and $s_2 \in \{x_\ell \leq b'_\ell, x_\ell \geq a'_\ell\}$ such that the system subject to $x_k \in S'_k$ $(1 \leq k \leq n)$ with the additional bounds s_1, s_2 is infeasible. Assume that s_1 is $x_j \geq \xi_j$ and s_2 is $x_\ell \leq b'_\ell$ (the other cases are treated similarly). It follows from Corollary 2.11 that there exists a simple path p from $\overline{v}_\ell$ to $\overline{v}_j$ such that $f_p(b'_\ell) < \xi_j$. In contrast, it follows from property iii and the computation of step vi that $f_{p'}(b'_\ell) \geq b'_j$ for all paths p' from $\overline{v}_\ell$ to $\overline{v}_j$, hence a contradiction.
We prove the correctness of the induction step. Consider the step that determines $\hat{x}_i$. Assume that the claim is true for previous steps. The induction hypothesis asserts that subject to the constraints $x_\ell \in \hat{S}_\ell^i$ $(1 \leq \ell \leq n)$, we have $[x_\ell^{\min}, x_\ell^{\max}] = \hat{S}_\ell^i \equiv [\hat{a}_\ell^i, \hat{b}_\ell^i]$. For $1 \leq \ell \leq n$, let $\tilde{x}_\ell^{\min}$, $\tilde{x}_\ell^{\max}$ be the respective values of $x_\ell^{\min}$, $x_\ell^{\max}$ subject to $x_\ell \in \hat{S}_\ell^i$ $(1 \leq \ell \leq n)$ and the additional constraint $x_i = \hat{x}_i$. We need to show that for $1 \leq \ell \leq n$, $[\tilde{x}_\ell^{\min}, \tilde{x}_\ell^{\max}] = \hat{S}_\ell^{i+1}$. The direction $\hat{S}_\ell^{i+1} \supset [\tilde{x}_\ell^{\min}, \tilde{x}_\ell^{\max}]$ is obvious. Consider $\tilde{x}_j^{\max}$ (the arguments for $\tilde{x}_j^{\min}$ are similar). If $\tilde{x}_j^{\max} = \hat{b}_j^i$ we are done. Otherwise, it follows from Proposition 2.10 that there exists a simple path p from $v_i \in \{\overline{v}_i, \underline{v}_i\}$ to $\underline{v}_j$ such that $f_p(\hat{x}_i) = \tilde{x}_j^{\max}$. It follows from property iii that either $p' = p_{\bar{i}\underline{j}}$ or $p' = p_{\underline{ij}}$ is as tight as p. Hence, $f_{p'}(\hat{x}_i) = \tilde{x}_j^{\max} = \hat{b}_j^{i+1}$.

Remark 3.4 Note that step vii can be replaced by choosing $\hat{x}_i = (a'_i + b'_i)/2$, $1 \leq i \leq n$. This follows from the fact that for the system subject to $x_i \in S'_i$ $1 \leq i \leq n$, we have $[x_i^{\min}, x_i^{\max}] = S'_i$ $(1 \leq i \leq n)$, and from Proposition 2.12 part ii.

3.3. Complexity of the naive implementation

The complexity is dominated by the calls to an algorithm for Problem 3.2. We present a naive algorithm for Problem 3.2 that is based on locating single values (see Problem 2.13). The algorithm consists of n sequential stages as follows. At the ith stage the interval I_i is computed and a tightest path is found for each of the $O(n)$ sets of paths that emanate from either one of $\{\underline{v}_i, \overline{v}_i\}$. In the proceeding stages we consider the system with the additional constraint $x_i \in I_i$.

For $1 \leq i \leq n$, denote by $k^{(i)} = \sum_u (k_{\overline{v}_i u} + k_{\underline{v}_i u})$, the number of paths emanating from $\{\underline{v}_i, \overline{v}_i\}$. We discuss stage i. We consider $O(n)$ sets of paths with the goal of choosing a tightest path in each set. This is done in $O(\log k^{(i)})$ iterations, where in each iteration the total number of paths that need to be considered reduces from r to $n + 3(r-n)/4$. Initially, $r = k^{(i)}$. After the stage terminates, $r = n$ and each set contains a single path (the tightest path). Each iteration is as follows. First, we pair up paths that belong to the same pair of vertices. Each such pair corresponds to a comparison between the two paths that needs to be resolved. For each pair, we compute the intersection of the two linear functions that correspond to the two paths. Each "comparison" amounts to locating the intersection point with respect to the interval $[x_i^{\min}, x_i^{\max}]$. We solve one instance of Problem 2.13 to locate the median of these $r/2$ intersections. By doing this, half the comparisons are resolved, and the number of remaining paths is at most $n + 3(r-n)/4$.

The following proposition is immediate. The parallel bounds follow from known parallel selection algorithms (see e.g., [13]).

Proposition 3.5 *Stage i can be performed by an $O(k^{(i)} + mn\log k^{(i)})$ algorithm. A parallel algorithm runs in $O(n\log m \log\log m \log k^{(i)})$ determinisitc time (alternatively, by allowing randomization, in $O(n\log k^{(i)})$ expected time) using $O(mn\log k^{(i)} + k^{(i)})$ work on a CRCW PRAM.*

We discuss the resulting complexity of Algorithm 3.3. In step ii, the total number of paths ($\sum_{i=1}^n k^{(i)}$) is the number of edges in the graph. Hence the number of operations is $O(mn^2\log m)$. In step iv, $k^{(i)} = O(n^2)$ $(1 \leq i \leq n)$. Hence, the number of operations in each execution of step iv is $O(n^3 + mn^2\log n)$. Step iv is performed $\lceil \log_2 2n \rceil$ times. It follows that the total number of operations is $O\left(mn^2(\log m + \log^2 n)\right)$. On a CRCW PRAM the algorithm runs in $O\left(n^2(\log m + \log^2 n)\log m \log\log m\right)$ time (alternatively, by allowing randomization, in $O\left(n^2(\log m + \log^2 n)\right)$ expected time), using $O(mn^2(\log m + \log^2 n)$ work.

3.4. Solving TVPI systems by locating pools of values

Consider the "naive" algorithm for solving Problem 3.2. The algorithm consists of n stages that are performed sequentially. We present a different algorithm that performs the stages "concurrently", where iterations of different stages are interleaving. The algorithm consists of $O(\max_{1\leq i\leq n}\log k^{(i)})$ phases, where at phase i all the ith iterations of the n stages are performed.

An iteration of stage $1 \leq j \leq n$ amounts to locating a value of x_j. Performing a phase amounts to solving an instance of the following problem:

Problem 3.6 [Locate a pool of values] Given are a graph G and a set of intervals $S_1, \ldots, S_n$ as in Definition 2.1. Also given are a set of values $\xi_i \in S_i$ $(i \in I)$ for the corresponding variables x_i $(i \in I)$, where $I \subset \{1, \ldots, n\}$. The goal is to find a set of intervals J_i $(i \in I)$, such that (i) $F \neq \emptyset \Rightarrow F \cap \{\boldsymbol{x} \in R^n | \bigwedge_{i\in I}\ x_i \in J_i\} \neq \emptyset$, and (ii) $\xi_i \notin \operatorname{interior} J_i$ $(i \in I)$.

We give a more elaborate description of the algorithm that reduces Problem 3.2 to locating pools of values. The correctness is straightforward. The algorithms performs interleaving executions of the "n" stages. This is done in $O(\log \max_i k^{(i)})$ phases. Denote by $I \subset \{1, \ldots, n\}$ the set of stages that did not terminate at the current phase (initially, $I = \{1, \ldots, n\}$). Phase j is as follows:

i. For each $\ell \in I$: compute the value ξ_ℓ of x_ℓ arising from an iteration of stage ℓ. This computation takes $O(n + (3/4)^j \sum_{i \in I} k^{(i)})$ time, and takes, with optimal speedup, $O(\log \max_i k^{(i)} \log\log \max_i k^{(i)})$ deterministic time and constant expected time on a CRCW PARM.

ii. Locate the pool (solve Problem 3.6) ξ_ℓ ($\ell \in I$). Let J_ℓ ($\ell \in I$) be the intervals constituting the solution.

iii. For $\ell \in I$, $S_\ell \leftarrow S_\ell \cap J_\ell$.

It follows that Problem 3.2 is reduced to solving $O(\max_i \log k^{(i)})$ instances of Problem 3.6 and $O(n \max_i \log k^{(i)} + \sum_{1 \leq i \leq n} k^{(i)})$ additional time. The problem of solving TVPI systems is therefore reduced to solving $O(\log^2 n + \log m)$ instances of Problem 3.6 and $O(m + n^3 \log n)$ additional computation. Note that, in parallel, the additional computation can be performed in $O(\log m \log\log m + \log^2 n \log\log n)$ time deterministically and in $O(\log n)$ expected time with optimal speedup.

Problem 3.6 can be solved naively by sequentially solving $|I|$ instances of Problem 2.13 (locating a single value). This requires mn^2 time, and $\tilde{O}(n^2)$ time in parallel. In Section 4 we give algorithms that improve over this bound.

4. Algorithms for locating a pool

The problem of solving TVPI systems was reduced to locating $O(\log^2 n + \log m)$ pools (Problem 3.6) and $O(n^3 \log n + m)$ additional computation.

In Section 3.4 we showed that a pool can be located in mn^2 time, and $\tilde{O}(n^2)$ time in parallel. In this section we discuss two algorithms for locating a pool. In Subsection 4.1 we present an algorithm that runs in $O(mn^2)$ sequential time, and $\tilde{O}(n)$ time in parallel with optimal speedup. In Subsection 4.2 we overview a randomized $O(mn \log^3 n)$ expected time algorithm. A parallel implementation runs in $\tilde{O}(n)$ expected time using $O(m)$ processors. The details of the randomized algorithm are given in Sections 5 and 6.

4.1. $\tilde{O}(n)$ time using $O(n^2 m)$ work

In this subsection we prove the following.

Proposition 4.1 *Problem 3.6 can be solved on a CRCW PRAM in $O(n \log\log m)$ time (alternatively, by allowing randomization, in $O(n)$ expected time) using $O(mn|I|)$ work.*

Corollary 4.2 *Algorithm 3.3 has a parallel implementation on a CRCW PRAM that runs in* $O\left(n \log\log m(\log^2 n + \log m)\right)$ *time (*$O\left(n(\log^2 n + \log m)\right)$ *expected time if randomization is allowed) and performs* $O(mn(\log^2 n + \log m))$ *work.*

The following algorithm solves Problem 3.6 within the time bounds stated in Proposition 4.1.

Algorithm 4.3 [locate a pool]

i. In parallel, locate (solve Problem 2.13) each of the values ξ_i $(i \in I)$.
For $i \in I$ do as follows:

- If $\xi_i \geq x_i^{\max}$, determine $J_i \leftarrow \{z | z \leq \xi_i\}$.
- If $\xi_i \leq x_i^{\min}$, determine $J_i \leftarrow \{z | z \geq \xi_i\}$.

Let $I' \subset I$ be the set of indices such that $x_i^{\min} < \xi_i < x_i^{\max}$ $(i \in I')$.

ii. For each $i \in I'$ perform the following computation:
Initialize the values at the vertices of G to $\overline{x}_i = \underline{x}_i = \xi_i$ and $\overline{x}_j = \infty$, $\underline{x}_j = -\infty$ $(j \neq i)$. Apply $2n$ push phases. For $1 \leq j \leq n$ denote the final values at the nodes $\underline{v}_j$, $\overline{v}_j$, respectively, by $\underline{x}_j^i, \overline{x}_j^i$.

iii. Construct a graph H as follows: The graph H has $|I'|$ nodes w_i $(i \in I')$. There is an edge between w_i and w_j if and only if $\xi_j < \underline{x}_j^i$ or $\xi_j > \overline{x}_j^i$.

iv. Compute a maximal independent set (corresponds to nodes $I^+ \subset I'$) in H.
Choose intervals J_i $(i \in I')$ as follows:
If $i \in I^+$, $J_i \leftarrow \{\xi_i\}$.
If $i \in I' \setminus I^+$, then:

- If $\min_{j \in I^+} \overline{x}_i^j < \xi_i$, $J_i \leftarrow \{z | z \leq \xi_i\}$.
- If $\max_{j \in I^+} \underline{x}_i^j > \xi_i$, $J_i \leftarrow \{z | z \geq \xi_i\}$.

We prove the correctness of the algorithm. It follows from Corollary 2.11 that $F \cap \{\boldsymbol{x} | x_i = \xi_i \wedge x_j = \xi_j\} \neq \emptyset$ if and only if $\underline{x}_j^i \leq \xi_j \leq \overline{x}_j^i$. Hence, the graph H captures the dependencies between pairs of values. Proposition 2.12 (see equivalence of properties i and iv) implies that a set of single variable equations is feasible if and only if every pair is feasible. Hence, the set of intervals J_i $(i \in I')$ solves Problem 3.6.

4.2. Overview of a $\tilde{O}(mn)$ algorithm

In this subsection we present an overview of a faster randomized algorithm for locating a pool of values. A single value can be located in $O(mn)$ operations by using Algorithm 2.22. In the previous section we solved Problem 3.2 more efficiently by inferring from locating a value of one variable about other values of the same variable. To locate a set I of values of different variables, however, we still needed $O(mn|I|)$ operations. The randomized approach presented in this section enables us to infer about values of other variables as well, and in particular, locate a pool of values within $\tilde{O}(mn)$ operations.

We prove the following:

Theorem 4.4 *Problem 3.6 can be solved sequentially in an expected number of $O(mn\log^3 n)$ operations, and on a CRCW PRAM in $O(n\log^3 n)$ expected time using $O(m)$ processors.*

Consequently, TVPI systems can be solved in an expected number of

$$O\left(n^3 \log n + mn(\log^5 n + \log m \log^3 n)\right)$$

operations.

Consider a pool of values $\xi_1, \ldots, \xi_n$ for the corresponding variables. These values are classified into 3 groups as follows (see Subsection 2.5):

i. $\xi_i \in [x_i^{\min}, x_i^{\max}]$ (ξ_i is feasible)

ii. $\xi_i \notin [x_i^{\min}, x_i^{\max}]$, but $\xi_i \in [x_i^{\min *}, x_i^{\max *}]$ (ξ_i is weakly infeasible)

iii. $\xi_i \notin [x_i^{\min *}, x_i^{\max *}]$ (ξ_i is strongly infeasible)

We propose a two stage algorithm for Problem 3.6. In the first stage the algorithm locates all the strongly infeasible values ξ_i ($i \in I'$), and determines whether $\xi_i < x_i^{\min *}$ or $\xi_i > x_i^{\max *}$. The respective intervals are determined to be $J_i = [-\infty, \xi_i]$ if $\xi_i > x_i^{\max *}$, and $J_i = [\xi_i, \infty]$ if $\xi_i < x_i^{\min *}$. In the second stage the algorithm solves an easier special case of Problem 3.6 where the values are guaranteed to be either feasible or weakly infeasible. In Section 5 we present an algorithm that solves the first stage and in Section 6 we present an algorithm for the second stage. These algorithms run in $O(mn\log^3 n)$ time, and in $O(n\log^3 n)$ parallel time using $O(m)$ processors on a CRCW PRAM.

By combining the above results we obtain algorithms for Problem 3.6 with the running times stated in Theorem 4.4.

5. Locating the strongly infeasible values

We present an algorithm for the following problem:

Problem 5.1 [Determine the strongly infeasible values] Given are values ξ_i ($i \in I$) for the corresponding variables x_i ($i \in I$), where $I \subset \{1, \ldots, n\}$. The goal is to determine for each $i \in I$ whether $\xi_i < x_i^{\min *}$, $\xi_i > x_i^{\max *}$, or $\xi_i \in [x_i^{\min *}, x_i^{\max *}]$.

We prove the following:

Proposition 5.2 *Problem 5.1 can be solved (i) sequentially, in an expected number of $O(mn \log^3 n)$ operations, and (ii) on a CRCW PRAM, in $O(n \log^3 n)$ expected time, using $O(m)$ processors.*

For purposes of analysis we classify the strongly infeasible values of variables according to the sizes of their certificates (see Definition 2.20):

Definition 5.3 A strongly infeasible value ξ_i of a variable x_i is *ℓ-big* if the certificate is of size at most ℓ. A value is $(\ell_1, \ell_2]$-big if it is ℓ_2-big but not ℓ_1-big. We interchangeably refer to strongly infeasible values as *big*. Note that all strongly infeasible values are $2n$-big.

We explain the motivation for this classification. The algorithm that determines all the big values is based on a tradeoff between the following two properties. These properties are stated more formally and proved later. The first one "favors" big values with small certificates: For any ℓ-big value ξ it takes $O(m\ell)$ operations (by using "reveal") to find the certificate. Hence, a decision procedure that locates an ℓ-big value requires only $O(m\ell)$ operations. The second property favors values with large certificates: We introduce the procedure "check" that considers a big value ξ along with an associated certificate E'. The "check" procedure can on average locate many other big values whose certificates intersect E'.

5.1. The algorithm

The description of the algorithm includes calls to the following three procedures:

- The first procedure "reveal" (Problem 2.23) considers a set of k values, and either finds a big value that belongs to the set, or concludes that all these values are feasible or weakly infeasible. Algorithm 2.24 solves "reveal". Recall (see Proposition 2.25) that the algorithm takes $O(mn)$ time. If the input set contains big values, the algorithm terminates after $O(m\ell)$ operations (where $\ell \leq n$) with a closed certificate of size $\theta(\ell)$. If the original set contained exactly one big value ξ_j, the certificate found by the algorithm is the certificate of ξ_j.

- The second procedure "check" considers an ℓ-big value and the corresponding certificate. The "check" procedure uses the certificate to produce upper and lower bounds on the feasible regions of other variables. These bounds enable us to make decisions regarding other values in the pool. An $O(m\ell)$ time "check" algorithm is given in Subsection 5.3. In Subsection 5.4 we discuss properties of the "check" algorithm. We show that at least half of the $(\ell, 2\ell]$-big values in the pool have the following property: When a "check" is applied to any of them, it determines at least $\ell/(6n) - 1$ of the other $(\ell, 2\ell]$-big values in the pool.

- The third procedure considers the set of unlocated values I and computes a crude estimate r^* to the number of big values r. The estimate r^* is such that $1/2 \leq r^*/r \leq 2$

with probability at least 1/2. In Subsection 5.5 we present an "estimate" algorithm that performs $O((\log\log n)^2)$ calls to the "reveal" procedure. The time complexity is dominated by these calls and hence is $O(mn(\log\log n)^2)$.

The following algorithm determines all big values in a pool of $|I| = \hat{n}$ values. The algorithm performs iterations where each iteration determines some of the big values. The set I and the number $\hat{n}$ are updated accordingly. Denote by $r \leq \hat{n}$ the (unknown) number of big values in I. Appropriate values for the constants C_1, C_2, and C are given later.

Algorithm 5.4 [Determine all the big values]

◇ Loop A: Steps i-ix

i. Apply the "reveal" procedure to the set I. If there are no big values, stop.

ii. Compute an estimate r^* for r such that with probability at least 1/2, $1/2 \leq r^*/r \leq 2$. Reset the following two counters: $tt \leftarrow 0$ [number of operations]; $tb \leftarrow 0$ [number of big values discarded] (in the current iteration of Loop A).

◇ Loop B: Steps iii-ix

iii. If either (i) $tb \geq r^*/4$ (successful iteration of Loop A), or (ii) $tt \geq Cmn\log^2 n$ (unsuccessful iteration of Loop A), go to Step i (next iteration).

iv. $b \leftarrow 0$ [the number of big values discarded in the current iteration of Loop B].

v. If $r^* \geq \hat{n}/4$, $s \leftarrow 1$. Otherwise, $s \leftarrow \lfloor \hat{n}/r^* \rfloor$. Choose $k = \lceil C_1 \log n \rceil$ random samples $S_1, \ldots, S_k$ of size s (with replacements) from the pool I.

vi. Execute "concurrently" k runs of "reveal" (see Algorithm 2.24) applied to the sets $S_1, \ldots, S_k$ as follows:

vii. Initialize copy i according to the set of values S_i. Set $\ell \leftarrow 0$ [current phase number]. Set $K \leftarrow \{1, \ldots, k\}$ [set of "active" runs].

◇ Loop C: Steps viii-ix

viii. Perform an additional iteration (push phase) to the runs in K. set $\ell \leftarrow \ell + 1$. Let $K' \subset K$ be the (possibly empty) subset of the runs for which certificates are found. Apply "check" operations to these certificates, and discard from I all the big values that are determined. Increment b and tb accordingly, and set $K \leftarrow K \setminus K'$.

ix. If either $\ell = 2n+1$ or $b \geq \lceil C_2 \ell r^*/(n \log n) \rceil$ then $tt \leftarrow tt + C_1 \ell m \log n$, go to Step iii [next iteration of Loop B]. Otherwise, go to Step viii.

In Subsection 5.2 we prove the following:

Proposition 5.5 *There exist constants C_1,C_2, and C as follows. If the estimate r^* is such that $1/2 \le r^*/r \le 2$, then with probability at least 1/2, the current iteration of Loop A terminates after determining $r^*/4$ big values (the iteration is successful).*

It follows that with probability at least 1/4, each iteration of Loop A determines at least 1/8 of the big values in the pool. Hence, the expected number of iterations performed until all big values are determined is $O(\log n)$. Each iteration of Loop A runs in $O(mn\log^2 n)$ time. Hence, the expected time in which Algorithm 5.4 terminates is $O(mn\log^3 n)$. This concludes the proof of Proposition 5.2.

Remark 5.6 Note that the computation of the estimate r^* in step ii can be replaced by running Loop A with $r^* = 2^1, 2^2, \ldots, 2^{\lfloor \log \hat{n} \rfloor}$. This results in an additional $\log n$ factor in the worst-case time bound.

5.2. Probability for a successful iteration of Loop A

In this subsection we prove Proposition 5.5.

Denote by r_i $(i = 1, \ldots, \lceil \log 2n \rceil)$ the number of $(2^{i-1}, 2^i]$-big values in I.

Definition 5.7 A sample $S \subset I$ of values from the pool is *good* if the following three conditions hold:

i. The sample contains exactly one big value ξ_j. Denote the size of the certificate of ξ_j by $2^{i-1} < \ell \le 2^i$.

ii. ξ_j determines (using "check") at least $2^{i-1}r_i/(6n)$ big values in I (including ξ_j itself).

iii. $r_i \ge r/(4\lceil \log 2n \rceil)$.

We claim that property (iii) holds for at least 3/4 of the big values. Let

$$L = \{1 \le i \le \lceil \log 2n \rceil \mid r_i < r/(4\lceil \log 2n \rceil)\} \ .$$

We need to show that $\sum_{i\in L} r_i < r/4$. The claim follows from the inequality

$$\sum_{i\in L} r_i/r < \sum_{i\in L} 1/(4\lceil \log 2n \rceil) = |L|/(4\lceil \log 2n \rceil) < 1/4 \ .$$

It follows from Theorem 5.17 in Subsection 5.4 that property (ii) holds for at least half the $(2^{i-1}, 2^i]$-big values (for all i). Hence, properties (ii) and (iii) hold for at least 3/8 of the big values.

The following proposition motivates the definition of good samples:

Proposition 5.8 *Consider an application of the reveal algorithm to a good sample, followed by a check to the certificate found. There exists a number $\ell = 2^i$ $(1 \le i \le \lceil \log 2n \rceil)$, such that by using $O(m\ell)$ operations we can locate at least $\lceil \ell r/(24n\lceil \log 2n \rceil) \rceil$ big values.*

Proof: Denote by ξ_j the big value in the sample, and let $\ell = 2^i$ be such that the size of the certificate of ξ_j is in $(2^{i-1}, 2^i]$. Consider an application of the reveal algorithm to the sample. It follows from Proposition 2.19 part ii that the certificate is found within $O(m\ell)$ operations. A check procedure applied to the certificate locates at least $2^{i-1}r_i/(6n) - 1$ additional big values from the pool. Note that $r_i \geq r/(4\lceil \log 2n \rceil)$. Hence, at least $\ell r/(24n\lceil \log 2n \rceil)$ big values are located. ▮

Proposition 5.9 *Consider an integer $s \geq 1$ such that (i) $s = 1$ if $r \geq \hat{n}/2$ and in some cases, when $r \geq \hat{n}/16$, and (ii) otherwise, $c = rs/\hat{n}$ is such that $0.25 \leq c \leq 2$. Consider a random sample $S \subset I$ (with replacements) of s elements. In some cases, and only when $s - 1 \geq 4$, the random sample contains $s - 1$ elements. There exists a constant ρ_1 such that with probability at least ρ_1, S contains exactly one big value.*

Proof: If $|S| = s$, the probability that S contains exactly one big value is

$$P = s\frac{r}{\hat{n}}\left(1 - \frac{r}{\hat{n}}\right)^{s-1} .$$

If $s = 1$, then $16r \geq \hat{n}$ and hence $P \geq 1/16$. Otherwise, $2r < \hat{n}$, hence $c/s < 1/2$. We obtain that

$$P \geq c\left((1 - c/s)^{s/c}\right)^c (1 - c/s)^{-1} \geq c/4^c \geq 1/32 .$$

If $|S| = s - 1$, we have

$$P = (s-1)\frac{r}{\hat{n}}\left(1 - \frac{r}{\hat{n}}\right)^{s-2} \geq ((s-1)/s)c/4^c \geq (4/5)(1/32) .$$

▮

Consider an iteration of Loop A. During iterations of Loop B, the number r of big values in the set I decreases, but the value of the estimate r^* is not updated. The iteration of Loop A terminates when $r^*/4$ big values are discovered. Therefore, if initially $1/2 \leq r^*/r \leq 2$, then during the iterations of Loop B we have $1/2 \leq r^*/r \leq 4$.

Corollary 5.10 *Suppose that $1/2 \leq r^*/r \leq 4$, and consider a random sample $S \subset I$ (with replacements) of size s, where $s = 1$ when $r^* \geq \hat{n}/4$ and $s = \lfloor \hat{n}/r^* \rfloor$ otherwise. The sample S is good with probability $\rho \geq 3\rho_1/8$.*

Corollary 5.11 *Let $C_1 = -1/\ln(1-\rho)$. Consider $\lceil C_1 \log n \rceil$ randomly chosen samples (with replacements) of size s. If $1/2 \leq r^*/r \leq 4$, the probability that none of the samples is good is smaller than $1/n$.*

Proof: The probability that no sample is good is $p = (1-\rho)^{\lceil C_1 \log n \rceil}$. It follows that $p \leq 1/n$ when $C_1 \geq -1/\ln(1-\rho)$. ▮

Assume that initially $1/2 \leq r^*/r \leq 2$. Choose $C_2 = 1/96$. We compute the expected number of steps performed until $r^*/4$ big values are located.

Definition 5.12 An iteration of Loop B is *beneficial* if $b \geq \lceil C_2 \ell r^*/(n \log n) \rceil$ when the iteration terminates (at step ix).

Proposition 5.13 *Consider an iteration of Loop B. If at least one of the samples is good, the iteration is beneficial.*

Proof: The proof follows from Proposition 5.8 and the fact that for $r^* \leq 4r$,

$$\lceil C_2 \ell r^*/(n \log n) \rceil \leq \lceil \ell r/(24n \lceil \log 2n \rceil) \rceil \ .$$

∎

The counter tt is such that at the end of each iteration of Loop B, the running time so far of the current iteration of Loop A is $O(tt)$. A beneficial iteration increases tt in $C_1 m \ell \log n$ units (runs in time $O(C_1 m \ell \log n)$), and each unbeneficial iteration by $C_1 mn \log n$ units ($O(C_1 mn \log n)$ time). It follows from Proposition 5.13 and Corollary 5.11 that an expected number of $(n-1)/n$ of all iterations of Loop B are beneficial. Hence, the expected number of operations performed by unbeneficial iterations is not bigger than the expected number of operations performed by beneficial iterations.

To conclude the proof of Proposition 5.5, it suffices to count the expected number of operations performed by beneficial iterations (increase of the counter tt) until at least $r^*/4$ big values are located.

Denote by t_i ($i = 1, \ldots, \lceil \log n \rceil$) the number of beneficial iterations that ended within $(2^{i-1}, 2^i]$ phases. The total expected increase in the value of tt by beneficial iterations is

$$\Delta_{tt} \leq C_1 \sum_{i=1}^{\lceil \log n \rceil} t_i 2^i m \log n \ .$$

The expected final value of tt (number of operations by beneficial and unbeneficial iterations) is $tt \leq 2\Delta_{tt}$. At all iterations but the last we have

$$\sum_{i=1}^{\lceil \log n \rceil} (t_i C_2 2^{i-1} r^*/(n \log n)) \leq r^*/4 \ .$$

Hence, after the last iteration, $\sum_{i=1}^{\lceil \log n \rceil} t_i 2^i \leq n \log n/(2C_2) + n$. Therefore,

$$\Delta_{tt} \leq ((C_1 + C_2)/(2C_2)) mn \log^2 n \ .$$

Thus, the expected final value of tt is

$$tt \leq ((C_1 + C_2)/C_2) mn \log^2 n \ .$$

We choose $C = 2(C_1 + C_2)/C_2$. It follows that with probability $1/2$, at least $r^*/4$ of the big values are located before the value of tt reaches $Cmn \log^2 n$. This concludes the proof of proposition 5.5.

5.3. The "check" procedure

In this subsection we give an algorithm for performing "checks." The "check" algorithm is applied to a $(2^{k-1}, 2^k]$-big value ξ_i and the certificate ξ_i, and provides information from which it can be concluded that certain other values are also big.

We explain how a closed certificate for a value of one variable is used to obtain information about other variables:

Remark 5.14 Suppose ξ_i is a big value of x_i. The certificate G_i consists of a path from v_i to a vertex u and a cycle that starts and ends at u, such that $v_i = \underline{v}_i$ if $\xi_i > x_i^{\max *}$, and $v_i = \overline{v}_i$ if $\xi_i < x_i^{\min *}$. For a vertex w in G_i, define $\xi_i(w) = f_p(\xi_i)$, where p is the path in G_i from v_i to w. Define $\xi_i(v_i) = \xi_i$. It follows from Corollary 2.16 that for every vertex w of G_i, the value $\hat{\xi}_i(w)$ at w is strongly infeasible.

The "check" algorithm is described below. The input is a $(2^{k-1}, 2^k]$-big value ξ_i of x_i along with the certificate G_i.

Algorithm 5.15 [check]

i. For each vertex $v \in G_i$, compute the big value $\xi(v)$ (as in Remark 5.14).

ii. Initialize the values of $\overline{x}_1, \ldots, \overline{x}_n$ and $\underline{x}_1, \ldots, \underline{x}_n$ as follows. If $\overline{v}_j \in G_i$, set $\underline{x}_j = \xi_i(\overline{v}_j)$; otherwise, if $\overline{v}_j \notin G_i$, set $\underline{x}_j = -\infty$. If $\underline{v}_j \in G_i$, set $\overline{x}_j = \xi_i(\underline{v}_j)$; otherwise, if $\underline{v}_j \notin G_i$, set $\overline{x}_j = \infty$.

iii. Perform 2^k push phases.

iv. Make conclusions as follows. If for some value η we have $\eta \geq \overline{x}_j$ (resp., $\eta \leq \underline{x}_j$), conclude that $\eta > x_j^{\max *}$ (resp., $\eta < x_j^{\min *}$).

The initial values at the vertices are consistent (see Remark 5.14), and hence (see Corollary 2.15), the final values are consistent. It follows that the conclusions made by the "check" algorithm are correct.

Definition 5.16 If in step iv of Algorithm 5.15 we deduce that a value η is big, we say that ξ_i 2^k*-locates* η.

5.4. Properties of the "check" procedure

This subsection establishes the following theorem:

Theorem 5.17 *Consider a collection of b $(2^{k-1}, 2^k]$-big values. At least half the values have the following property. If a "check" is applied to any of them, it results in 2^k-locating at least $b2^{k-1}/(6n) - 1$ other values from the collection.*

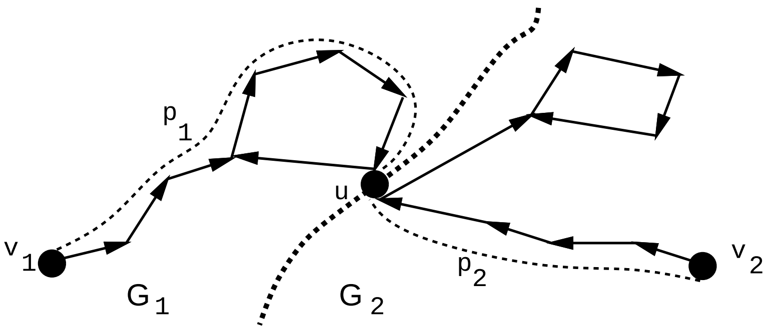

Figure 5: Dependence of intersecting certificates

The following proposition analyzes the dependencies among values of different variables.

Proposition 5.18 *Let ξ_i and ξ_j be big values of x_i and x_j, respectively. Let the respective certificates G_i, G_j, and the nodes v_i, v_j be as in Remark 5.14. Suppose that $|G_i| \le \ell$, $|G_j| \le \ell$ for some ℓ. If the sets of vertices participating in G_i and G_j intersect, at least one of the values ξ_i and ξ_j is ℓ-located by the other.*

Proof: Suppose $u \in \underline{V}$ (similar for $u \in \overline{V}$) participates both in G_i and in G_j. Let p_i be the directed path in G_i from v_i to u and let p_j be a directed path in G_j from v_j to u. (see Figure 5 for an illustration, where $i = 1$ and $j = 2$). We claim that if $f_{p_i}(\xi_i) \le f_{p_j}(\xi_j)$ then ξ_i ℓ-locates ξ_j; if $f_{p_j}(\xi_j) \le f_{p_i}(\xi_i)$ ξ_j ℓ-locates ξ_i. Assume that $f_{p_i}(\xi_i) \le f_{p_j}(\xi_j)$, and consider an application of "check" to ξ_i. The "check" algorithm determines that $f_{p_i}(\xi_i)$ is a strongly infeasible value of u. The algorithm initializes u^{-1} to the consistent value $f_{p_i}(\xi_i)$ and applies ℓ push phases. Consider the path p_j^{-1} from u^{-1} to v_j^{-1}. Since $|p_j^{-1}| \le \ell$, the value at v_j^{-1} after ℓ push phases must be at least as tight as $\xi_j^* \equiv f_{p_j^{-1}}(f_{p_i}(\xi_i))$. To conclude the proof, note that ξ_j^* is at least as tight as ξ_j at v_j^{-1}, since $\xi_j = f_{p_j^{-1}}(f_{p_j}(\xi_j))$ and we assumed that $f_{p_j}(\xi_j)$ is no more tight than $f_{p_i}(\xi_i)$ at u^{-1}. Hence, ξ_i ℓ-locates ξ_j. ∎

Definition 5.19 Consider a pool of b $(2^{k-1}, 2^k]$-big values. The *influence* graph of this pool is defined as follows. Each big value in the pool corresponds to a vertex in the influence graph. If one value 2^k-locates another, then there is a directed edge from the vertex of the former to the vertex of the latter.

Theorem 5.17 is an immediate corollary of the following theorem:

Theorem 5.20 *In the influence graph of a pool of b $(2^{k-1}, 2^k]$-big values, at least half of the vertices have an out-degree greater than $\frac{1}{6}b2^{k-1}/n - 1$.*

We first prove two propositions that are used in the proof of the theorem.

Proposition 5.21 *In the intersection graph of b subsets of $\{1,\ldots,n\}$ with cardinality ℓ, the sum of the degrees is at least $b^2\ell/n - b$.*

Proof: Let s_i $(i = 1,\ldots,n)$ be the number of subsets containing i. Obviously, $\sum_{i=1}^n s_i = b\ell$. The sum of the degrees plus the number of vertices (subsets) is at least $\sum_{i=1}^n s_i^2/\ell$. This expression is minimized when $s_i \in \{\lfloor b\ell/n\rfloor, \lceil b\ell/n\rceil\}$ $(i = 1,\ldots,n)$. It follows that the sum of degrees is at most $\lceil b^2\ell/n\rceil - b$. ∎

Proposition 5.22 *In any orientation of the edges of the intersection graph of b subsets of $\{1,\ldots,n\}$ with cardinality ℓ, at least half the vertices have out-degrees greater than $\frac{1}{6}b\ell/n-1$.*

Proof: It follows from Proposition 5.21 that the sum of out-degrees (i.e., the total number of edges) in any subgraph induced by b' vertices is at least $\frac{1}{2}(b'^2\ell/n - b')$. Suppose, to the contrary, that there exists a subset U of cardinality $|U| = b' = b/2$ where the out-degree of each vertex is less than or equal to $\frac{1}{6}b\ell/n - 1$. The total number of edges (that equals the sum of out-degrees) in the subgraph induced by these vertices is $\frac{1}{3}b'^2\ell/n - b'$, hence a contradiction since it is less that $\frac{1}{2}(b'^2\ell/n - b')$. ∎

Proof of Theorem 5.20: The cardinality of a certificate of a $(2^{k-1}, 2^k]$-big value is greater than or equal to $\ell = 2^{k-1}$. It follows from Proposition 5.18 that if two big values have intersecting certificates, then in the influence graph there is an edge between the corresponding vertices. Hence, the influence graph contains an intersection graph of b subsets of $\{1,\ldots,n\}$ of cardinality ℓ. Thus, the proof follows from Proposition 5.22. ∎

5.5. Estimating the number of big values

In this subsection we prove the following proposition:

Proposition 5.23 *An estimate r^* such that $1/2 \le r^*/r \le 2$ with probability at least $1/2$, can be found using $O((\log\log n)^2)$ "reveal" operations.*

The problem of computing an estimate can be stated in the following terms. We are given a collection of $\hat{n}$ stones, where r of them are radioactive. The number r is not known. We are equipped with a primitive gauge that when exposed to a set of stones can determine if at least one of them is radioactive (i.e., "reveal" operation). A *good estimate* to the number of radioactive stones is a number r^* such that $1/2 \le r^*/r \le 2$. The goal is to compute a good estimate that is correct with probability at least $1/2$, by using at most $O((\log\log\hat{n})^2)$ gauge readings. We first give some useful propositions.

Proposition 5.24 *Denote by $p_0(s)$ the probability that a sample (with replacements) of size s contains no radioactive stones and let $c \equiv rs/\hat{n}$. Under these conditions, (i) $c < \ln(1/p_0(s))$, and (ii) if $r/\hat{n} \le 0.5$, $c \ge \log(1/p_0(s))/\log 4$.*

Proof:

$$p_0(s) = \left(1 - \frac{r}{\hat{n}}\right)^s = \left(\left(1 - \frac{r}{\hat{n}}\right)^{\frac{\hat{n}}{r}}\right)^c$$

It follows that $p_0(s) < e^{-c}$ and if $r/\hat{n} \leq 0.5$, $p_0 \geq 4^{-c}$. ∎

Corollary 5.25 *Suppose $p_0(s)$ is known. We can conclude as follows:*

i. If $0.136 \leq p_0(s) \leq 0.5$, then $1/2 \leq c \leq 2$.

ii. If $0.146 \leq p_0(s) \leq 0.49$, then $0.514 \leq c \leq 1.92$.

iii. If $p_0(s) \leq 0.156$, then $c \geq 1.34$.

iv. If $p_0(s) \geq 0.48$, then $c \leq 0.74$.

Proposition 5.26 *Let B' be the r.v. that corresponds to the number of non-radioactive samples out of $B = 5000(\lceil \log\log \hat{n} \rceil + 1)$ random samples of size s.*

$$Prob\,\{|B'/B - p_0(s)| \geq 0.01\} \leq 1/(2\lceil \log\log \hat{n} \rceil + 2)\ .$$

Proof: B' is the number of successes of B Bernoulli trials with success probability $p_0(s)$, and hence, has a binomial distribution. Therefore B' has expected value $p_0(s)B$ and variance $Bp_0(s)(1 - p_0(s))$ (see [11] for background). It follows from Chebyshev inequality that $Prob\,\{|B' - p_0(s)B| \geq 0.01B\} \leq 2500B/B^2 = 2500/B = 1/(2\lceil \log\log \hat{n} \rceil + 2)$. ∎

Denote by $\boldsymbol{p}'_0(s)$ the r.v. B'/B, and by $p'_0(s)$ a value of $\boldsymbol{p}'_0(s)$.

Below we state an algorithm that computes the estimate r^*. For simplicity, we assume that $\hat{n}$ is a power of 2 (the general case can be handled by a small modification of the algorithm). The algorithm is based on performing a binary search on the $\log \hat{n} - 1$ possible estimate values $2^1, 2^2, \ldots, 2^{\log \hat{n} - 1}$.

Algorithm 5.27 [Compute an estimate]

i. If $p'_0(2) \leq 0.35$, stop and return the estimate $r^* = 2^{\log \hat{n} - 1}$.

ii. Initialize $a = 1$, $b = \lfloor \log \hat{n} \rfloor$. Repeat step iii until $a = b$. Return $r^* = \hat{n}/2^a$.

iii. Set $j = \lceil (a + b)/2 \rceil$, and $s = 2^j$.

- If $0.146 \leq p'_0(s) \leq 0.49$, stop and return the estimate $r^* = \hat{n}/s$.
- If $p'_0(s) < 0.146$, set $b = j$ and go to step iii.
- If $p'_0(s) > 0.49$, set $a = j$ and go to step iii.

We show that with probability at least 0.5, $1/2 \leq r^*/r \leq 2$. Consider the first step. If $r \geq \hat{n}/2$, $p_0(2) \leq 1/4$. Hence, $Prob\{p_0'(2) \leq 0.35\} \geq 1 - 1/(2\lceil \log\log \hat{n} \rceil + 2)$. If $r < \hat{n}/4$, $p_0(2) > 9/16$. Hence, $Prob\{p_0'(2) < 0.35\} \leq 1/(2\lceil \log\log \hat{n} \rceil + 2)$. It follows that if $r \geq \hat{n}/2$, the algorithm stops with a good estimate with probability at least $1 - 1/(2\lceil \log\log \hat{n} \rceil + 2)$. If $r \leq \hat{n}/4$ the probability that the algorithm stops at step i with a bad estimate is smaller than $1/(2\lceil \log\log \hat{n} \rceil + 2)$.

The algorithm performs at most $\lceil \log\log \hat{n} \rceil$ iterations of step iii. Each iteration terminates with a further restriction of the set $a, a+1, \ldots, b$. Consider a single iteration of step iii. Suppose that at the beginning of the iteration, at least one of the values $2^a, 2^{a+1}, \ldots, 2^b$ constitutes a good estimate. It follows from Proposition 5.26 that with probability $(1 - 1/(2\lceil \log\log \hat{n} \rceil + 2)$, $|p_0'(s) - p_0(s)| \leq 0.01$. Assume that this is the case. Note that for $r^* = \hat{n}/s$ we have $c = r/r^*$. Therefore, it follows from Corollary 5.25 that when the iteration ends the modified interval $a, a+1, \ldots, b$ is such that at least one of $2^a, 2^{a+1}, \ldots, 2^b$ is a good estimate.

Therefore, the probability that when the algorithm terminates 2^a is a good estimate is at least

$$(1 - 1/(2\lceil \log\log \hat{n} \rceil + 2))^{\lceil \log\log \hat{n} \rceil + 1} \geq 0.5 \ .$$

The algorithm performs $O(\log\log \hat{n})$ iterations, where each iteration computes an appropriate value of p_0'. Each iteration requires $O(\log\log \hat{n})$ "gauge readings" (see Proposition 5.26). Hence, the total number of "gauge readings" performed by the algorithm is $O((\log\log \hat{n})^2)$. This concludes the proof of Proposition 5.23.

6. Locating a pool of weakly infeasible values and feasible values

In this section we present an algorithm that asserts the following:

Theorem 6.1 *Problem 3.6, where the values ξ_i ($i \in I$) are guaranteed to be either feasible or weakly infeasible, can be solved (i) sequentially, in $O(mn \log^3 n)$ expected time, and (ii) on a CRCW PRAM, in $O(n \log^3 n)$ expected time, using $O(m)$ processors.*

The following is a key property of feasible and weakly infeasible values:

Proposition 6.2 *Let ξ_i and ξ_j be feasible or weakly infeasible values of x_i and x_j, respectively. If there exists a path p from $v_j \in \{\underline{v}_j, \overline{v}_j\}$ to $v_i \in \{\underline{v}_i, \overline{v}_i\}$ such that $f_p(\xi_j)$ is at least as tight as ξ_i at the vertex v_i, there exists such a simple path p'.*

Proof: We assume that $v_j = \overline{v}_j$, $v_i = \overline{v}_i$ (the proof for the other cases is similar). Consider the path p from v_j to v_i. Suppose a vertex u occurs more than once on p. Consider two of the occurrences of u. Let $p = p_1 p_2 p_3$, where p_1 is the prefix of p until the first occurrence and p_3 is the suffix of p starting from the second occurrence of u. If $f_{p_1 p_2}(\xi_j)$ is not tighter than $f_{p_1}(\xi_j)$ at u, the path $p' = p_1p_3$ is at least as tight as p. Consider iterating the above process. It follows that there exists a path $\hat{p}$ from v_j to v_i that is at least as tight as p with respect to $x_j = \xi_j$, and when a vertex appears more than once, the value at subsequent occurrences is always tighter. We claim that each vertex may occur at most once in $\hat{p}$. Assume the contrary, and consider the first cycle c along $\hat{p}$. Let u be the vertex where c starts and ends, and let $p' = p_1c$ be the corresponding prefix of $\hat{p}$. It follows from Proposition 2.17 that either $f_{p_1}(\xi_j)$ is strongly infeasible at u, or $f_{p_1}(\xi_j)$ is consistent at u. If $f_{p_1}(\xi_j)$ is strongly infeasible, then p' is a closed certificate for $[-\infty, \xi_j]$ with respect to x_j and we get a contradiction to the assumption that ξ_j is not strongly infeasible. Otherwise, $f_{p_1}(\xi_j)$ is consistent at u. It follows that for every prefix p_2 of $\hat{p}$ that contains p_1 and ends at a vertex u', the value $f_{p_2}(\xi_j)$ is consistent at u'. Consider the last cycle c' along $\hat{p}$, and let $p' = c'p_3$ be the corresponding suffix of $\hat{p}$. It follows that p' is a closed certificate for $[f_{\hat{p}}(\xi_j), \infty]$ at v_i^{-1}. Hence, since $\xi_i \geq f_{\hat{p}}(\xi_j)$, p' is a closed certificate for $[\xi_i, \infty]$. This is a contradiction to ξ_i not being strongly infeasible. ∎

We present two procedures that are subroutines of the algorithm for locating a pool. The first procedure is applied to a subset of the values and returns a partial solution. The second procedure extends a partial solution as much as possible without restricting the feasible region further. We discuss the first procedure in detail. The input is a subset of values $\xi_i \in S_i$ $(i \in I')$, where $I' \subset I$. The procedure determines intervals J_i $(i \in I'')$, where $I'' \subset I'$ is a subset of the values in the input set. The intervals J_i $(i \in I'')$ constitute a partial solution for Problem 3.6, that is, there exists a set of intervals J_i $(i \in I \setminus I'')$ such that J_i $(i \in I)$ is a solution of Problem 3.6.

Algorithm 6.3 [Find a partial solution]

i. Initialize the values at the vertices of G as follows:
$\overline{x}_i = \underline{x}_i = \xi_i$ $(i \in I')$, $\overline{x}_i = b_i$, $\underline{x}_i = a_i$ $(i \notin I')$,

ii. Apply $2n$ push phases.

iii. For $i \in I'$ do as follows:

- If $\xi_i = \overline{x}_i = \underline{x}_i$, set $J_i = \{\xi_i\}$.
- If $\xi_i = \overline{x}_i < \underline{x}_i$, set $J_i = [\xi_i, \infty]$.
- If $\xi_i = \underline{x}_i > \overline{x}_i$, set $J_i = [-\infty, \xi_i]$.
- Otherwise, J_i is not determined.

Let $I'' \subset I'$ be the set of all $i \in I'$ for which J_i is determined. For $i \in I''$ do: $S_i \leftarrow S_i \cap J_i$.

Denote by F^* the feasible region prior to applying Algorithm 6.3. Consider an application of the algorithm. We show that if $F^* \neq \emptyset$ then the system remains feasible, that is, $F \neq \emptyset$. The following proposition proves a necessary condition for correctness: If $F^* \neq \emptyset$, then for all $i \in I''$, $F^* \cap \{\boldsymbol{x} | x_i \in J_i\} \neq \emptyset$.

Proposition 6.4 *Suppose $\underline{x}_i = \xi_i$ (resp., $\overline{x}_i = \xi_i$) is weakly infeasible, and the certificate does not consist only from a bound and a simple path. There exists a simple path p from $\underline{v}_i$ to $\overline{v}_i$ (resp., $\overline{v}_i$ to $\underline{v}_i$), such that $f_p(\xi_i) < \xi_i$ (resp., $f_p(\xi_i) > \xi_i$).*

Proof: It suffices to prove the existence of a not necessarily simple path p with the above properties. The existence of a simple path would follow from Proposition 6.2. We assume $\underline{x}_i = \xi_i$ is weakly infeasible (the other case is proved similarly). There exists $1 \leq j \leq n$ and two simple paths emanating from $\underline{v}_i$, p_1 to $\overline{v}_j$ and p_2 to $\underline{v}_j$ such that $f_{p_1}(\xi_i) < f_{p_2}(\xi_i)$. Consider the path $p = p_1 p_2^{-1}$ from $\underline{v}_i$ to $\overline{v}_i$. We claim that $f_p(\xi_i) < \xi_i$. To prove the claim note that $f_p(\xi_i) = f_{p_2^{-1}} \circ f_{p_1}(\xi_i)$. Since the end vertices of p_2^{-1} both lie in $\underline{V}$, the function $f_{p_2^{-1}}$ is monotone increasing. Hence, $f_{p_2^{-1}} \circ f_{p_1}(\xi_i) < f_{p_2^{-1}} \circ f_{p_2}(\xi_i) = \xi_i$. ∎

We show that the intervals J_i $(i \in I'')$ constitute a partial solution:

Proposition 6.5 $F = F^* \cap \{\boldsymbol{x} | \bigwedge_{i \in I''} \; x_i \in J_i\} \neq \emptyset$.

Proof: We first claim that J_i is feasible for all $i \in I''$. Consider $i \in I''$. If ξ_i is feasible the claim is immediate. Otherwise, assume to the contrary that $J_i = [\xi_i, \infty]$ (similar for $[-\infty, \xi_i]$) is weakly infeasible. If the certificate consists of a bound in S_j $(j \in I)$ and a simple path then, when the algorithm terminates, we have $\overline{x}_i < \xi_i$ (since the initialization is with values at least as tight as S_j $(j \in I)$). This is a contradiction. Otherwise, the certificate contains an open cycle. The contradiction follows from Proposition 6.4. This concludes the proof of the claim. In order to prove the proposition, Proposition 2.12 part iii asserts that it suffices to show that for every pair $\{i, j\} \in I''$, $F^* \cap \{\boldsymbol{x} | x_i \in J_i, \; x_j \in J_j\} \neq \emptyset$. The latter follows from Corollary 2.11. ∎

We discuss the procedure that extends a partial solution. Suppose we applied Algorithm 6.3 and as a result, for $i \in I''$, we updated the intervals S_i and determined J_i. The following algorithm may determine the interval J_i for some additional values $i \in I \setminus I''$.

Algorithm 6.6 [Extend a partial solution]

i. Initialize the values at the vertices of G as follows. $\underline{x}_i = a_i$ and $\overline{x}_i = b_i$ $(i \in I)$.

ii. Apply $2n$ push phases.

iii. For $i \in I \setminus I''$ conclude as follows:

- If $\overline{x}_i \leq \xi_i$, set $J_i = \{-\infty, \xi_i\}$.
- If $\underline{x}_i \geq \xi_i$, set $J_i = \{\xi_i, \infty\}$.

Let $\hat{I}$ be the set of all $i \in I \setminus I''$ such that J_i was determined.
For $i \in \hat{I}$ do: $S_i \leftarrow S_i \cap J_i$.

Remark 6.7 Proposition 2.5 asserts that the feasible region is not changed by the updates of the intervals S_i done by Algorithm 6.6. In particular, if the feasible region was non empty prior to applying the algorithm it remains non empty.

6.1. The algorithm

The following algorithm considers feasible and weakly infeasible values ξ_i $(i \in I)$, and solves Problem 3.6 for these values. The solution consists of a set of intervals J_i $(i \in I)$. The algorithm performs iterations, where in each iteration some intervals J_i $(i \in I' \subset I)$ are determined. The intervals J_i $(i \in I')$ are incorporated into the system, we set $I \leftarrow I \setminus I'$, and the problem is reduced to locating the values ξ_i $(i \in I)$.

Algorithm 6.8 [Locate a pool]

i. Repeat the following until $I = \emptyset$.

ii. For $\ell = 1, 2, 4, \ldots, 2^{\lfloor \log |I| \rfloor}$, execute steps iii–v:

iii. Choose (uniformly at random) ℓ elements from I. Let $I' \subset I$ be the set of chosen elements.

iv. Apply Algorithm 6.3 to I'. The algorithm determines J_i for $i \in I'' \subset I'$.

v. Apply Algorithm 6.6 and determine J_i for additional values $\hat{I} \in I$. Set $I \leftarrow I \setminus \{I'' \cup \hat{I}\}$.

It follows from Proposition 6.5 and Remark 6.7 that if the problem was feasible, it remains feasible at any point during the execution of the algorithm. Hence, when the algorithm terminates, the intervals J_i $(i \in I)$ constitute a solution for Problem 3.6.

6.2. Complexity

For values ξ_i $(i \in I)$, consider a solution for Problem 3.6, that is, intervals J_i $(i \in I)$ such that $J_i \in \{[-\infty, \xi_i], [\xi_i, \infty]\}$ and $F \cap \{\boldsymbol{x} | \bigwedge_{i \in I} x_i \in J_i\} \neq \emptyset$.
We designate vertices v_i $(i \in I)$ according to the choice of intervals J_i $(i \in I)$: If $J_i = [-\infty, \xi_i]$, $v_i = \overline{v}_i$. If $J_i = [\xi_i, \infty]$, $v_i = \underline{v}_i$.

We define a relation on the set I, with respect to the solution J_i $(i \in I)$:

Definition 6.9 Let $i \in I$, $j \in I$ be two elements.

i. We say that j *locates* i if there exists a path p from v_j to v_i such that $f_p(\xi_j)$ is at least as tight as ξ_i on v_i.

ii. We say that i *interferes* with j if there exists a path p from v_i^{-1} to v_j^{-1} such that $f_p(\xi_i)$ is tighter than ξ_j on v_j^{-1}.

iii. We define the relation $\prec$ on I as follows. $j \prec i$ if and only if i interferes with j. $j \preceq i$ if and only if j locates i. (Note that $j \prec i$ implies $j \preceq i$, and $j \preceq i \wedge i \not\preceq j$ implies $j \prec i$.)

iv. For an element $i \in I$, denote by $\mathrm{pred}(i) = \{j \in I | j \preceq i\}$ the set of elements that locate i, and by $\mathrm{succ}(i) = \{j \in I | j \succ i\}$ the set of elements that interfere with i.

The following is a corollary of Proposition 6.2:

Corollary 6.10 *If a path p as in Definition 6.9 parts i and ii exists, then there must exists a simple path with the same properties.*

The following proposition relates the locate and interfere relations to the algorithms presented earlier. The proof follows from Corollary 6.10.

Proposition 6.11 *Let $i \in I$, $j \in I$ be two elements.*

i. Suppose that j locates i. If $S_j \subset J_j$, then an application of Algorithm 6.6 would determine J_i.

ii. Suppose that i interferes with j. Consider an application of Algorithm 6.3 where $\{i, j\} \subset I'$. The interval J_j will not be determined as a result of such a run. Moreover, if $j \in I'$, the interval J_j is determined in this run if and only if for all values $i \in I'$, i does not interfere with j.

We characterize the values located in an iteration of Algorithm 6.8, in terms of pred and succ:

Corollary 6.12 *The following relations hold in an execution of steps iv–v: $I'' = \{k \in I' | \mathrm{succ}(k) \cap I' = \emptyset\}$ and $\hat{I} = \{k \in I \setminus I'' | \mathrm{pred}(k) \cap I'' \neq \emptyset\}$.*

Proposition 6.13 *The relation $\prec$ is a partial order on I.*

Proof: The transitivity of $\prec$ is immediate. We need to show that $\prec$ has no cycles. If there is a cycle, it follows from the transitivity that for some $j \in I$, j interferes with j. The path p is in this case a simple cycle that constitutes a closed certificate for ξ_j. This contradicts the assumption that ξ_j is not strongly infeasible. ∎

Note that each iteration of steps iii–v results in determining at least one interval, and therefore, the algorithm terminates within $O(n)$ iterations. In order to prove Theorem 6.1, however, we need to show that the expected number of iterations is only $O(\log^3 n)$.

Define $I^0 = \{i \in I | \mathrm{succ}(i) = \emptyset\}$.

Proposition 6.14

$$|I^0| + \sum_{i \in I \setminus I^0} \sum_{j \in \mathrm{succ}^{-1}(i)} 1/|\mathrm{succ}(j)| = |I|$$

Proof: The sum is well defined since in each inner sum, $\{i\} \cup \mathrm{succ}(i) \subset \mathrm{succ}(j)$, and hence $|\mathrm{succ}(j)| \geq 2$. Note that

$$\sum_{i \in I \setminus I^0} \sum_{j \in \mathrm{succ}^{-1}(i)} 1/|\mathrm{succ}(j)| = \sum_{i \in I \setminus I^0} |\mathrm{succ}(i)|/|\mathrm{succ}(i)| = |I| - |I^0| .$$

∎

Consider iterations of steps ii-v. For $k \geq 0$, denote by I_k, I^0_k, $\mathrm{succ}_k : I_k \rightarrow 2^{I_k}$, $\mathrm{pred}_k : I_k \rightarrow 2^{I_k}$, respectively, the contents of the sets I, I^0_k, and the corresponding succ and pred functions in the beginning of the kth iteration. For $k \geq 0$ and $i \in I_k$ we have $\mathrm{succ}_k(i) = \mathrm{succ}_0(i) \cap I_k$ and $\mathrm{pred}_k(i) = \mathrm{pred}_0(i)$ (since if $j \in \mathrm{pred}(i)$ is located, then so is i).

For $k \geq 0$, define

$$L_k = \{i \in I_k \setminus I^0_k | \sum_{j \in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \geq 1/4\} .$$

Proposition 6.15 *There exists a constant ρ_0 such that for every $k \geq 0$ and $i \in L_k \cup I^0_k$, $Prob\{i \notin I_{k+1}\} \geq \rho_0$.*

Proof: Consider $i \in I^0_k$. It follows from Corollary 6.12 that if $i \in I'$ in some iteration of steps iii– v, then i gets located. With probability at least 1/2, $i \in I'$ at the iteration where $\ell = 2^{\lfloor \log |I_k| \rfloor}$. Consider $i \in L_k$. For $0 \leq h \leq \lfloor \log |I_k| \rfloor$, denote

$$T_h = \{j \in \mathrm{succ}_k^{-1}(i) | 2^h \leq |I_k|/|\mathrm{succ}_k(j)| \leq 2^{h+1}\} .$$

It follows that

$$\sum_{0 \leq h \leq \lfloor \log |I_k| \rfloor} 2^h |T_h|/|I_k| \geq (1/2) \sum_{j \in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \geq 1/8 .$$

For $0 \leq h \leq \lfloor \log |I_k| \rfloor$, consider the iteration where $\ell = 2^h$. Consider an element $j \in T_h$. Note that $\{i\} \cup \mathrm{succ}_k(i) \subset \mathrm{succ}_k(j)$. Hence, $|\mathrm{succ}_k(j)| \geq 2$ and $\ell \leq |I_k|/|\mathrm{succ}_k(j)| \leq |I_k|/2$. Therefore, the probability that $\mathrm{succ}_k(j) \cap I' = \emptyset$ is at least

$$(1 - \ell/|I_k|)^{|\mathrm{succ}_k(j)|} \geq (1 - \ell/|I_k|)^{2|I_k|/\ell} \geq 1/16 .$$

Note that these probabilities are dependent for different elements of T_h. The probability that $j \in I'$ is at least $\ell/|I_k|$. These probabilities are independent for different elements of T_h. Using Corollary 6.12, it follows that the probability that i is not located in the iteration where $\ell = 2^h$ is at most

$$(1 - \ell/|I_k|)^{|T_h|} \leq ((1 - \ell/|I_k|)^{|I_k|/\ell})^{\ell |T_h|/|I_k|} \leq 4^{-2^h |T_h|/|I_k|} .$$

These probabilities are independent for different iterations. Hence, the probability that i is not located in any iteration is at most

$$\prod_{0 \leq h \leq \lfloor \log |I_k| \rfloor} 4^{-2^h |T_h|/|I_k|} = 4^{-\sum_{0 \leq h \leq \lfloor \log |I_k| \rfloor} 2^h |T_h|/|I_k|} \leq 4^{-1/8} < 0.85 .$$

Thus, i is located with probability at least 0.15. ∎

Proposition 6.16

$$\sum_{i\in L_k}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \geq 3(|I_k|-|I_k^0|)/4 \ .$$

Proof: The proof follows from Proposition 6.14 and from

$$\sum_{i\in I_k\setminus(I_k^0\cup L_k)}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \leq (|I_k|-|I_k^0|-|L_k|)/4 \leq (|I_k|-|I_k^0|)/4 \ .$$

∎

Proposition 6.17 *Denote $\rho_2 = 3\rho_0/8$ and $\rho_1 = \rho_0/2$. With probability at least ρ_1,*

$$|I_k^0 \setminus I_{k+1}| + \sum_{i\in I_k\setminus(I_k^0\cup I_{k+1})}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \geq \rho_2|I_k| \ .$$

Proof: Consider the random variable

$$R = |I_k^0 \setminus I_{k+1}| + \sum_{i\in L_k\setminus I_{k+1}}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \ .$$

Note that

$$\sum_{i\in I_k\setminus(I_k^0\cup I_{k+1})}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \geq \sum_{i\in L_k\setminus I_{k+1}}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \ .$$

Hence, it suffices to show that $Prob\{R \geq \rho_2|I_k|\} \geq \rho_1$. It follows from Proposition 6.15 that the expected value of R is

$$E(R) \geq \rho_0|I_k^0| + \rho_0 \sum_{i\in L_k}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \ .$$

The maximum value of R is at most

$$R_{\max} = |I_k^0| + \sum_{i\in L_k}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)| \ .$$

Note that

$$E(R) \leq Prob\{R < E(R)/2\}E(R)/2 + Prob\{R \geq E(R)/2\}R_{\max} \ .$$

Therefore,

$$Prob\{R \geq E(R)/2\} \geq E(R)/(2R_{\max} - E(R)) \geq E(R)/(2R_{\max}) \geq \rho_0/2 \ .$$

It follows that with probability at least $\rho_0/2$,

$$R \geq (\rho_0|I_k^0| + \rho_0 \sum_{i\in L_k}\sum_{j\in \mathrm{succ}_k^{-1}(i)} 1/|\mathrm{succ}_k(j)|)/2 \geq (\rho_0|I_k^0| + 3\rho_o(|I_k|-|I_k^0|)/4)/2 \geq 3\rho_0|I_k|/8 \ .$$

The second to last inequality follows from Proposition 6.16. ∎

Proposition 6.18 *Consider a sequence of* $h' \geq 4c_0/\rho_1\lceil\log_{c_1}|I_k|\rceil$ *iterations of steps ii-v, starting at the kth iteration, where* $c_0 \geq 1$ *and* $c_1 > 1$ *are constants that are specified below. With probability at least* $1/2$, $|I_{k+h'}| < (1-\rho_2/2)|I_k|$.

Proof: Let H be the set of all $1 \leq h \leq h'$ such that

$$|I^0_{k+h-1} \setminus I_{k+h}| + \sum_{i\in I_{k+h-1}\setminus(I^0_{k+h-1}\cup I_{k+h})}\ \sum_{j\in \mathrm{succ}^{-1}_{k+h-1}(i)} 1/|\mathrm{succ}_{k+h-1}(j)| \geq \rho_2|I_{k+h-1}| \ .$$

It follows from Proposition 6.17 that for $1 \leq h \leq h'$, $h \in H$ with probability at least ρ_1. Hence, using bounds on the Binomial distribution, with probability $1-\exp(O(-h'\rho_1)) \geq 1/2$, $|H| \geq c_0\lceil\log_{c_1}|I_k|\rceil$. We assume that $|H| \geq c_0\lceil\log_{c_1}|I_k|\rceil$ and $|I_{k+h'}| \geq (1-\rho_2/2)|I_k|$ and obtain a contradicition. Consider $h \in H$. It follows, using Proposition 6.14, that

$$|I^0_{k+h-1} \cap I_{k+h}| + \sum_{i\in I_{k+h}\setminus I^0_{k+h-1}}\ \sum_{j\in \mathrm{succ}^{-1}_{k+h-1}(i)} 1/|\mathrm{succ}_{k+h-1}(j)| =$$

$$= |I^0_{k+h-1} \cap I_{k+h}| + \sum_{i\in I_{k+h}\setminus I^0_{k+h-1}} |\mathrm{succ}_{k+h}(i)|/|\mathrm{succ}_{k+h-1}(i)| < (1-\rho_2)|I_{k+h-1}|$$

We obtain that

$$\sum_{i\in I_{k+h}\setminus I^0_{k+h-1}} |\mathrm{succ}_{k+h}(i)|/|\mathrm{succ}_{k+h-1}(i)| < (1-\rho_2)|I_{k+h-1}| - |I^0_{k+h-1} \cap I_{k+h}|$$

$$\leq (1-\rho_2)|I_{k+h}|/(1-\rho_2/2) - |I^0_{k+h-1} \cap I_{k+h}| \ .$$

Denote $\rho_3 = (1-\rho_2)/(1-\rho_2/2)$ and $\rho_4 = (1+\rho_3)/2$. Note that $\rho_3 < 1$ and $\rho_3 < \rho_4 < 1$. The number of elements in the summation that are not smaller than ρ_4 is less than

$$(\rho_3|I_{k+h}| - |I^0_{k+h-1} \cap I_{k+h}|)/\rho_4 \ .$$

The number of elements in the summation is $|I_{k+h} \setminus I^0_{k+h-1}| = |I_{k+h}| - |I^0_{k+h-1} \cap I_{k+h}|$. Hence, more than

$$(1-\rho_3/\rho_4)|I_{k+h}| + (1/\rho_4 - 1)|I^0_{k+h-1} \cap I_{k+h}| \geq (1-\rho_3/\rho_4)|I_{k+h}|$$

elements are smaller than ρ_4. Using $|I_{k+h}| \geq (1-\rho_2/2)|I_k|$, it follows that after each iteration in H, for more than $(1-\rho_3/\rho_4)(1-\rho_2/2)|I_k|$ elements, the number of successors (size of succ_{k+h}) reduces by a factor of at least ρ_4. Let

$$c_0 = 1/((1-\rho_3/\rho_4)(1-\rho_2/2)) + 1$$

and $c_1 = (1-\rho_4)^{-1}$. The size of elements in the range of succ_k is bounded by I_k. Hence,

$$|H| < c_0\lceil\log_{c_1}|I_k|\rceil \ ,$$

which contradicts our assumption. ∎

Corollary 6.19 *The expected number of iterations of steps ii-v is $O(\log^2 n)$.*

Proof of Theorem 6.1: Algorithm 6.8 terminates with a correct solution. Each execution of steps ii-v amounts to $O(\log n)$ calls to Algorithms 6.3 and 6.6. Hence, it follows from Corollary 6.19 that Algorithm 6.8 performs an expected number of $O(\log^3 n)$ applications of Algorithms 6.3 and 6.6. Recall that Algorithms 6.3 and 6.6 run in $O(mn)$ time sequentially, and on a CRCW PRAM, in $O(n)$ time using $O(m)$ processors. It follows that the expected running time of Algorithm 6.8 is as stated in the theorem. ∎

7. Monotone systems

Consider the following two operations on vectors in R^n:

$$\begin{aligned} \boldsymbol{x} \vee \boldsymbol{y} &= (\max\{x_1, y_1\}, \ldots, \max\{x_n, y_n\})^T \quad &\text{(the } \textit{join} \text{ of } \boldsymbol{x} \text{ and } \boldsymbol{y}) \\ \boldsymbol{x} \wedge \boldsymbol{y} &= (\min\{x_1, y_1\}, \ldots, \min\{x_n, y_n\})^T \quad &\text{(the } \textit{meet} \text{ of } \boldsymbol{x} \text{ and } \boldsymbol{y}) \;. \end{aligned}$$

Consider a TVPI system as in Definition 1.1, where the matrix $\boldsymbol{A}^T$ (resp., $-\boldsymbol{A}^T$) is a *pre-Leontief* matrix, that is, each row of $\boldsymbol{A}$ contains at most one positive (resp., negative) entry. The set of solutions of such a system constitutes a semilattice relative to the $\vee$ (resp., $\wedge$) operation (see Cottle and Veinott [10]). It follows that if all the variables are bounded from above (resp., below), there exists a solution where all the variables are maximized (resp., minimized) simultaneously, hence, the vector $\boldsymbol{x}^{\max}$ (resp., $\boldsymbol{x}^{\min}$) is feasible. Also, the associated graph of such a system (see Definition 2.1) does not contain edges from $\underline{V}$ to $\overline{V}$ (resp., from $\overline{V}$ to $\underline{V}$).

A TVPI system where the two non-zero coefficients in each inequality have opposite signs (equivalently, both $\boldsymbol{A}^T$ and $-\boldsymbol{A}^T$ are pre-Leontief) is called *monotone*. The set of solutions of monotone systems constitutes a lattice relative to the $\vee$ and $\wedge$ operations. Note that the converse is also true: if a polyhedral set constitutes a lattice relative to the $\vee$ and $\wedge$ operations, then it can be represented as the set of solutions of some monotone system. This converse is a direct consequence of the following result due to Veinott [18]: a polyhedral set is a semilattice relative to the $\vee$ (resp., $\wedge$) operation if and only if it constitutes the set of solutions of some linear system $\boldsymbol{Ax} \le \boldsymbol{b}$ (resp., $\boldsymbol{Ax} \ge \boldsymbol{b}$), where the matrix $\boldsymbol{A}^T$ (resp., $-\boldsymbol{A}^T$) is pre-Leontief. Consider, for example, the non-monotone system in Figure 6. The points $(5,1)$ and $(3,4)$ are feasible, but the point $(5,1) \wedge (3,4) = (3,1)$ is not. Monotone systems are related to generalized network flow problems: the linear programming dual of the uncapacitated generalized transshipment problem is monotone. In [4, 7] we present algorithms for generalized network flow problem that exploit this relationship.

The algorithms presented here for solving the feasibility of TVPI systems can be adapted to find the vectors $\boldsymbol{x}^{\min}$, $\boldsymbol{x}^{\max}$ for monotone systems. Note that in the associated graph of a monotone system there are no edges passing between the two sets of vertices $\overline{V}$ and $\underline{V}$. It follows that all certificates are either closed certificates or simple paths.

We consider an application of Algorithm 3.3 to a monotone system, with the goal of finding the point $\boldsymbol{x}^{\max}$. It suffices to rely on a simpler notion of locating values: *locating* the

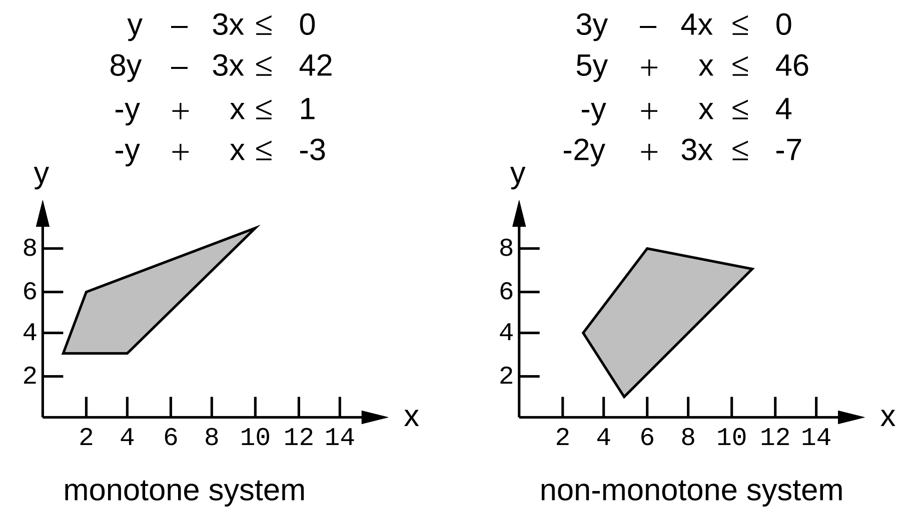

Figure 6: Examples of a monotone system and a non-monotone system

value ξ means locating it with respect to $x_i^{\max}$ (rather than determining its position relative to $x_i^{\min}$ as well). Problem 3.6 amounts to the following:

Problem 7.1 [Locate a pool of values] Given are values ξ_i $(i \in I)$ for the corresponding variable x_i $(i \in I)$. Locate these values, i.e., determine if $\xi_i > x_i^{\max}$ $(i \in I)$. Choose intervals J_i $(i \in I)$ as follows. $J_i = [\xi_i, \infty]$, if $\xi_i \leq x_i^{\max}$ and $J_i = [-\infty, \xi_i]$, if $\xi_i \geq x_i^{\max}$ $(i \in I)$.

It is easy to see that the intervals J_i are such that $\{\boldsymbol{x} | \bigwedge_{i \in I} \ x_i \in J_i\}$ contains the simultaneous maximum.

Consider step vii of Algorithm 3.3. The interval S_i' are feasible for x_i $(1 \leq i \leq n)$. Hence, for $1 \leq i \leq n$, $b_i' = x_i^{\max}$ (x_i is unbounded if and only if $b_i' = \infty$). If all variables are bounded, $\boldsymbol{b}' = \boldsymbol{x}^{\max}$. Otherwise, the algorithm supplies dependencies that allow us to compute feasible solutions when we determine some variables and maximize others.

Remark 7.2 An algorithm that computes $\boldsymbol{x}^{\max}$ (resp., $\boldsymbol{x}^{\min}$) for feasible monotone systems can be applied to compute $\boldsymbol{x}^{\max}$ (resp., $\boldsymbol{x}^{\min}$) for TVPI systems where the matrix $\boldsymbol{A}^T$ (resp., $-\boldsymbol{A}^T$) is pre-Leontief. Consider a system $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ where at most one of the two nonzero coefficients in each row is positive. Obtain a monotone system $\boldsymbol{A}'\boldsymbol{x} \leq \boldsymbol{b}'$ by omitting all rows of $\boldsymbol{A}$ where there are two negative entries. Compute the vector $\boldsymbol{y}$ that maximizes all variables in the resulting monotone system. If the original system $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ is feasible, $\boldsymbol{y}$ is the vector that maximizes all variables.

8. Concluding remarks

In this paper we considered systems of linear inequalities, where each inequality has at most two nonzero coefficients (TVPI systems). We presented algorithms that solve the feasibility problem, that is, either find a point that satisfies all the inequalities or conclude that no such point exists. We gave a $\tilde{O}(mn^2)$ deterministic algorithm and a $\tilde{O}(n^3 + mn)$ expected time randomized algorithm. The complexities of the respective parallel implementations are: $\tilde{O}(n)$ time using $O(mn)$ processors, and $\tilde{O}(n)$ expected time using $O(n^2 + m)$ processors. Although the analysis of these algorithms seems quite lengthy, the algorithms themselves are simple. The underlying computation amounts to the basic Bellman-Ford and Floyd-Warshall [8] shortest path algorithms where only simple data structures are used. Our algorithms are strongly polynomial, that is, the number of operations does not depend on the size of input numbers. We comment that the size (number of bits) of the numbers generated by the algorithms is $O(nC)$, where C is the size of the largest coefficient in the input.

We give some comments and suggestions for further research:

The time complexity of the randomized algorithm involves many logarithmic factors ($\log^5 n$). We suspect that a more careful analysis may eliminate some of these factors. The constant factors in the algorithms of Section 5 are quite large, but probably can be reduced significantly. We believe that some appropriate version of the algorithms presented here could work well in practice. Another obvious question is whether the $\tilde{O}(n^3+mn)$ bound can be achieved deterministically.

We discuss the possibility of improving over a $\tilde{O}(n^3 + mn)$ bound. The $\tilde{O}(n^3)$ factor results from the all-pairs shortest paths Floyd-Warshall computation and is inherent from our basic framework. A different approach, however, may yield a $\tilde{O}(mn)$ algorithm. We note that more work efficient parallel all-pairs shortest paths algorithms that are applicable for semirings, can be combined with our techniques (instead of the Floyd-Warshall algorithm) to obtain better bounds. In particular, a polylogarithmic time $f(m,n) = o(n^3)$ work all-pairs shortest-paths algorithm yields a $\tilde{O}(f(m,n)+mn)$ algorithm for solving TVPI systems. We comment that better bounds are obtained for TVPI systems where the associated graph has some special structure for which more work-efficient shortest paths algorithms exist. When the associated graph has a separator decomposition using $O(n^{\mu})$ size separators (e.g., planar graphs, k-dimensional grid graphs bounded on most dimensions, or bounded tree-width graphs), the corresponding TVPI systems can be solved in $\tilde{O}(n^{3\mu} + mn)$ time, using a shortest paths algorithm by the first author [5]. Consider, as an example, a TVPI system such that the variables correspond to vertices of a 3-dimensional grid, and all inequalities are "local" (that is, for each inequality, the distance on the grid between the two vertices corresponding to the variables in the inequality is bounded by some constant). The associated graph has a separator decomposition using $O(n^{2/3})$ size separators, and hence, these TVPI systems can be solved in $\tilde{O}(mn)$ expected time.

The feasibility problem of monotone systems includes as a very special case the problem of detecting existence of negative-weight directed cycles in a graph with n nodes, m edges, and real weights associated with the edges (using a well known reduction). The best known bound for detecting negative weight cycles is $O(mn)$, and hence, we believe it is unlikely that a $\tilde{o}(mn)$ algorithm exists for solving TVPI systems. We sketch the reduction. Consider a weighted graph $G = (V, E, w)$, where $w : E \to R$. The corresponding monotone system is as follows. For each node $v \in V$ assign a variable x_v. For each edge $e = (u, v) \in E$ assign the inequality $x_v - x_u \leq w(e)$. It follows from Proposition 2.10 that G contains a negative weight cycle if and only if the system is infeasible.

Our algorithms solve the feasibility problem of TVPI systems. They can be adapted, however, to (i) find a solution that maximizes a specific variable (ii) and find the lexicographic maximum. Questions that remain open regard finding an optimal solution relative to an arbitrary linear objective function. It is not known whether a strongly polynomial time algorithm exists for the problem, or whether we can achieve a time bound that is better than specializations of existing general LP algorithms. A partial result is due to Cosares [9] who showed that when the objective function has a fixed number of nonzero entries, the problem can be solved in strongly polynomial time bounds. The degree of the polynomial, however, grows linearly with the number of nonzero entries.

We discuss the parallel complexity of solving TVPI systems. Lueker, Megiddo, and Ramachandran [14] showed that the problem of finding an optimal solution relative to a general objective function is $\mathcal{P}$-complete. It is not known, however, whether the feasibility problem is $\mathcal{P}$-complete. The algorithms presented in this paper have a parallel running time of $\tilde{O}(n)$. This is the best known parallel time bound achievable by a polynomial number of processors. Lueker, Megiddo, and Ramachandran [14] gave an algorithm for the problem that runs in polylogarithmic time, but uses $n^{O(\log n)}$ processors.

Acknowledgements. The authors would like to thank the anonymous referee for very careful reading and many good suggestions for improving the presentation.

References.

[1] I. Adler and S. Cosares. A strongly polynomial algorithm for a special class of linear programs. *Oper. Res.*, 39:955–960, 1991.

[2] B. Aspvall. *Efficient algorithms for certain satisfiability and linear programming problems.* PhD thesis, Department of Computer Science, Stanford University, Stanford, Ca., 1980.

[3] B. Aspvall and Y. Shiloach. A polynomial time algorithm for solving systems of linear inequlities with two variables per inequality. *SIAM J. Comput.*, 9:827–845, 1980.

[4] E. Cohen. *Combinatorial Algorithms for Optimization Problems.* PhD thesis, Department of Computer Science, Stanford University, Stanford, Ca., 1991.

[5] E. Cohen. Efficient parallel shortest-paths in digraphs with a separator decomposition. In *Proc. 5th annual ACM Symposium on Parallel Algorithms and Architectures*, pages 57–67. ACM, 1993.

[6] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 145–155. ACM, 1991.

[7] E. Cohen and N. Megiddo. New algorithms for generalized network flows. In D. Dolev, Z. Galil, and M. Rodeh, editors, *Proc. of the 1st Israeli Symposium on the Theory of Computing and Systems*, pages 103–114. Springer-Verlag, Lecture Notes in Computer Science Vol. 601, 1992.

[8] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. McGraw-Hill Book Co., New York, 1990.

[9] S. Cosares. *On the complexity of some primal-dual linear programming pairs*. PhD thesis, Department of IEOR, University of California, Berkeley, Ca., 1988.

[10] R. W. Cottle and A. F. Veinott Jr. Polyhedral sets having a least element. *Math. Prog.*, 3:238–249, 1972.

[11] W. Feller. *An introduction to probability theory and its applications*. John Wiley & Sons, New York, 1950.

[12] D. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. Manuscript, 1991.

[13] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwenm, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 17, pages 869–941. MIT Press/Elsevier, Cambridge, MA, 1991.

[14] G. S. Lueker, N. Megiddo, and V. Ramachandran. Linear programming with two variables per inequality in poly log time. *SIAM J. Comput.*, 19(6):1000–1010, 1990.

[15] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12:347–353, 1983.

[16] C. G. Nelson. An $n^{O(\log n)}$ algorithm for the two-variable-per-constraint linear programming satisfiability problem. Technical Report AIM-319, Stanford University, 1978.

[17] R. Shostak. Deciding linear inequalities by computing loop residues. *J. Assoc. Comput. Mach.*, 28:769–779, 1981.

[18] A. F. Veinott Jr. Representation of general and polyhedral subsemilattices and sublattices of product spaces. *Linear Algebra and its Applications*, 114/115:681–704, 1989.

Contents.