Potential and Challenges of Two-Variable-Per-Inequality Sub-Polyhedral Compilation

Ramakrishna Upadrasta
INRIA and LRI, Paris-Sud 11 University
Ramakrishna.Upadrasta@inria.fr

Albert Cohen
INRIA
Albert.Cohen@inria.fr

Abstract

We explore the potential of sub-polyhedra as a means to fight some of the algorithmic complexity challenges in polyhedral compilation. The static analysis community has produced a rich variety of sub-polyhedral abstractions, trading precision for scalability for interprocedural analysis and symbolic model-checking. In this work, we evaluate the potential of the Two-Variable-Per-Inequality (TVPI) numerical abstract domain for polyhedral compilation. We demonstrate that strongly polynomial-time algorithms can be designed for the construction of complex affine transformations, with applications to affine scheduling and partitioning, with various objectives (parallelization, tiling, latency minimization). We identify some limitations, questions and research directions to be explored in depth in what we think is a new approach for the construction of scalable polyhedral compilation tools.

1. Introduction and Motivation

Static Control Parts (SCoPs) can be analyzed, modeled and transformed in the so-called Polyhedral model of compilation. Analyses and transformations of SCoPs involve a variety of geometrical and operation research algorithms on polyhedra. These algorithms are generally exponential in the loop nesting depth (dependence analysis, array expansion and contraction), and some of them can also be exponential in the number of statements in the SCoP (affine scheduling, partitioning, placement, code generation, for the most relevant optimization criteria) [DRV00].

Today and in the foreseeable future, the difficult problem remains to construct a "good" multidimensional affine transformation. The seminal work of Feautrier [Fea92] opened the avenue of constraint-based affine transformation methods, building on the affine form of the Farkas lemma. This approach has been refined, extended and applied in many directions. To cite only two recent achievements at the two extremes of the complexity spectrum: the tiling-centric Pluto algorithm of Bondhugula et al. [BHRS08] extending the Forward Communication Only (FCO) principle of Griebl [GFG02] for coarse-grain parallelization, and the complete, convex characterization and decoupled exploration heuristic of Pouchet et al. [PBB$^+$11]. Much progress has been made in the understanding of the theoretical and practical complexity of polyhedral compilation problems. Nevertheless, when considering multidimensional affine transformations, none of these appear to be strongly polynomial in the size of the program. The lowest complexity heuristics such as Pluto appear to be reducible to linear programming, which is only weakly polynomial in theory, and only at the cost of significant (yet practically effective) restrictions of the optimization space.

Scalability to large SCoPs is the main motivation of this work. This is best understood in light of the upcoming challenges of modeling large polyhedral process networks [Fea06], and compiling full functions as SCoPs, including irregular and data-dependent control and data flow [BPCB10]. Broadly, in this work, we classify algorithms that are linear or quasi linear in the number of dependences as *scalable algorithms*. On the other hand, high-complexity in the nesting depth of the program is typically not a source of scalability problems. Further, we prefer algorithms having low constant factors and being graph theory based, rather than being linear programming based. Such kinds are likely to be strongly polynomial time algorithms whose complexity does not depend on the encoding of the size of integers. The polyhedral community regularly uses linear programming, integer linear programming, Fourier-Motzkin elimination, vertex representation of polyhedra and integer sets. All these, we classify as asymptotically unscalable algorithms.

Some algorithms generate non-convex polyhedra as the result of solving parametric problems. Their complexity is generally not strongly polynomial, but the algorithms may still be considered scalable when the number of parametric contexts can be bounded w.r.t. the loop nesting depth.

Besides scalability, we are also interested in automatic parallelization and loop nest optimization in Just-In-Time (JIT) compilation scenarios. These scenarios are already considered for the evolution of back-end compilers for modern GPUs, although the existing tools are currently limited to local optimizations. In a JIT compilation context, linear time and space complexity in the number of dependences is clearly an upper bound. Better bounds would be widely appreciated.

2. Polyhedral Approximations in Static Analysis

The static analysis community has been considering the tradeoffs between different sub-polyhedra as a means of trading expressiveness for computational complexity in solving abstract interpretation problems. One main application of these approximate polyhedra, or *sub-polyhedra*, are static analysis and other run-time program verification problems like, value range analysis, bounds checking of indices of array variables, buffer overflows, validation of array accesses etc.

In formulating these applications, the static analysis community uses descriptions of programs called *abstract domains*. An abstract domain is presented as a lattice $\langle D, \sqsubseteq, \sqcup, \sqcap \rangle$, with $\sqsubseteq$ being an ordering predicate, and $\sqcap$ and $\sqcup$ being the associated meet and join operators respectively.

Many of these problems can be formulated as general polyhedral operations in the Poly abstract domain, in which D are convex sets that can be described by finite number of general linear inequalities. But the static analysis community has found Poly to be too expensive for the applications of interest, for several reasons.

Firstly, the need for join operators on polyhedra. The most precise join operation calculates the closure of the convex hull of the

two input polyhedra, that is, a set of inequalities that represent the smallest space that includes the two input polyhedra. One straightforward way to do the above is to use the generator representation of the two polyhedra and compute the convex hull. This method needs the generator representation of both of them, and a conversion from constraint representation to generator representation and back. Hence it can take exponential time as convex-hull of even two d-dimensional hypercubes requires just $2d$ inequalities for each hypercube, while needing 2^d vertices.

Secondly, it is the recursive nature of (flow) equations which are solved in an iterative way until a fix-point is reached. In solving these equations, the meet operator usually adds a few additional inequalities to an existing system and then the system is used for other operations like entailment checks (checking if $P_1 \subseteq P_2$ holds), or subjected again to join or even the projection operations. This leads to the need for a so called *closed system*, in which all information between any pairs of variables can be inferred by using the set of constraints that involve only those pairs of variables. Closure can also be considered as the process of calculating all possible projections. Needless to say, closure is very costly for Poly.

Finally, another reason is the scale of problems that are being dealt with. For the more demanding applications coming from runtime verification problems, the more powerful Poly are simply unaffordable. Miné [Min06] (page 4) says "*... polyhedron domain has a memory and time cost unbounded in theory and exponential in practice*". This is also substantiated by empirical studies by Laviron and Logozzo in [LL09] (page 2) : "*... Poly easily propagates the constraints. However, ... the price to pay for using Poly is too high: the analysis not scale beyond dozens of variables, ... while mostly one wants to solve hundreds of constraints and variables.*". These results are further substantiated by the much earlier study by Halbwachs et al. in [HMG06].

Hence, sub-polyhedral abstract domains started being used, by either restricting the shape of the polyhedron (the weakly relational domain approach) or by apriori limiting the number of linear-inequalities that can be solved (the bounded domains approach). The most obvious in the first category are (multidimensional) intervals, or "boxes". Beyond these are the Difference Bound Matrices (DBMs) [Bag97, Min01a, SKS00], Octagons, a.k.a. Unit Two Variables Per Inequality (Unit-TVPI or UTVPI) [Min01a, Min01b, Min06], Two Variables Per Inequality (TVPI) [SKH02], SubPolyhedra [LL09], Octahedra [CC04] etc. Other interesting ones include Logahedra and Pentagons. Some of these are summarized in Table 1.

Sub-polyhedra	Approximation (nature of constraint) $(a, b, c \in \mathbb{Q})$
Intervals	$a \leq x_i \leq b$
DBM	$x_i - x_j \leq c; x_i, x_j \geq 0$
Octagons (UTVPI)	$\pm x_i \pm x_j \leq c$
TVPI	$ax_i + bx_j \leq c$
Sub-Polyhedra (SubPoly)	LinEq $\otimes$ Interval
Pentagons	$a \leq x_i \leq b \wedge x_i < x_j$
Octahedra	$\pm x_i \pm \cdots \pm x_j \leq c$
Convex Polyhedra	$\sum a_i x_i \leq c$

Table 1. Various Sub Polyhedra

Of these, we discuss in detail the TVPI domain and its UTVPI specialization. The TVPI domain because of its excellent precision-complexity tradeoff, and UTVPI because of the maturity of the existing tools and applications.

2.1 Octagons (UTVPI)

Difference Bound Matrices (DBM) have constraints of the form $x_i - x_j \leq c; x_i, x_j \geq 0$. Their use as abstract domains is mainly by [Bag97, Min01a, SKS00].

Octagons have constraints of the form $\pm x_i \pm x_j \leq c$. They are mainly from [Min01a, Min06, Min01b, Min06, BHZ09], and are called so because in 2-dimensions, their geometric shape is octagonal. Octagons are similar to *Simple Sections* introduced by Balasundaram-Kennedy in [BK89], who introduced them in the context of loop-parallelization. They are also referred to as Unit Two Variables Per Inequality (UTVPI) because of the nature of their constraints. It is easy to see that Intervals $\subset$ DBM $\subset$ UTVPI.

Octagons, though not being closed under projection, have cubic-time closure and the optimum result of reduction of that problem to Floyd-Warshall is by Bagnara et. al in [BHZ09].

Octagons have the same asymptotic bounds as DBMs thanks to a data-structure innovation which uses an adjacency matrix representation. Most geometrical operations have quadratic-time (or worst case cubic-time) complexity.

DBMs and Octagons benefit from the algorithms suitable for TVPI (presented in the next section), but it is an open problem whether there exists a linear programming algorithm that has an asymptotically better running time than that of linear programming on non-closed TVPI systems. So is the cost of efficient Fourier-Motzkin on non-closed systems.

2.2 TVPI

TVPI have been pioneered as an abstract domain by Simon, King and Howe [SKH02, SKH10]. In TVPI polyhedra, each constraint is of the form: $ax_i + bx_j \leq c$. It is obvious to see that $UTVPI \subset TVPI$. TVPI are obviously closed under projection, and hence many algorithms on geometric operations that are developed for planar polyhedra (polygons) are directly applicable to general TVPI giving rise to simple algorithms with small complexity.

Linear Programming on TVPI has a longer history than their use in abstract domains. The linear programming community has been interested in them because of the strongly polynomial time algorithms for the linear programming problem on TVPI polyhedra.

The early work on using graph theory techniques for linear programming on TVPI systems was by Shostak [Sho81]. Aspvall and Shiloach [AS80] showed the polynomiality of the feasibility problem of TVPI-LP problems. This was followed by network flow based ("combinatorial") strongly polynomial time algorithms by Cohen and Megiddo [CM94]. The following result by Wayne [Way99] is the best result to date.

Lemma 2.1 [LP optimization on TVPI] Linear programming optimization on TVPI systems can be solved in $\mathcal{O}(m^3 n^2 \log B)$ worst case time, where m is the number of inequalities, n is the number of variables and B the upper bound on the size of the constants.

The above is a polynomial time combinatorial algorithm and uses a reduction of a specialized TVPI problem, namely one having monotone (DBM) inequalities, to a generalized min-cost-flow problem, namely one which allows negative circulations too. The former is a dual of the latter and hence they both have the same complexity bounds. Though the reduction uses DBM, result by Wayne can be generalized to the more general TVPI because of a previous result by Cohen and Megiddo.

It well known that for general polyhedra, the optimization and the feasibility problems have the same pseudo-polynomial time hardness. But it is interesting to note that they have different complexities on TVPI systems. The feasibility problem has lower complexity that the above result by Wayne on the optimization problem.

Hochbaum and Naor in [HN94] showed that feasibility (or satisfiability) of TVPI polyhedra can be determined in *strongly* polynomial time.

Lemma 2.2 [Feasibility on TVPI] Feasibility of TVPI stystems can be solved in $\mathcal{O}(mn^2 \log m)$ worst case time, where m is the number of inequalities, and n is the number of variables.

The above algorithm is surprisingly simple and uses a clever pruning technique to limit the redundant inequalities. It can be seen that the above result can as well be used to derive strongly polynomial time bounds for Fourier Motzkin elimination, and for projection of variables from TVPI systems.

Solving Integer-TVPI systems is NP-Complete and hence is no cheaper than solving general integer linear programs. The result is by [Lag85].

2.3 Complexities of general and sub polyhedra

Many times in this paper we will be referring to the *average* case complexity of Linear Programming on a *typical* input general convex polyhedron. Defining the problem and finding the complexity of the most commonly used simplex algorithm, which runs so well in practice, while taking exponential time on pathological inputs (as shown by Klee-Minty) has been a vexing issue for complexity theorists for decades. Researchers (like [ST04]) have even proposed new kinds of analysis "smoothened analysis" to prove the polynomiality of simplex algorithm. Thoroughout this paper, we will however be using the following result, which we believe is from [Sch86] and folklore.

Experimental complexity of LP and simplex algorithm Let Z be the complexity of Linear Programming using simplex algorithm, on a problem with n variables and m constraints. On a *typical* input, and with n and m being of the same order of magnitude ($m = \mathcal{O}(n)$), it can be assumed that $Z = \mathcal{O}(n^3)$ on *average*.

Comparison of complexities and scalabilities Many times in this paper, we would be comparing complexities of algorithms (mainly LP and FM) on Sub Polyhedra with that of the same algorithms on general polyhedra. Such a comparison becomes a difficult task, and one needs to be careful in generalizing such comparisons because of the following reasons.

Firstly, though LP has an exponential time worst case complexity, and the corresponding value for FM is doubly exponential, these measures are largely built on artificial examples. The average case complexities on typical inputs is usually likely to be smaller. In particular, it is likely that LP runs on usual inputs in polynomial (cubic) time as noted above, and FM on a smaller than doubly exponential time.

Secondly, the complexity bounds of Sub Polyhedra (like in [HN94, CM94, Way99] for example) describe only worst case bounds with no mention of average case complexity. On the other hand, some of the complexity bounds on Sub Polyhedra given by static analysis community assume closure. Closure is generally not needed for sub-polyhedral compilation as the set of inequalities do not get altered.

Thirdly, the average case of these algorithms on Sub Polyhedra on typical inputs is bound to be much better than what is guaranteed by the worst case bounds. For example, though the LP algorithm of Wayne has seemingly quintic complexity, it may as well be that it runs on optimum cubic time complexity, similar to Floyd-Warshall algorithm.

Finally, the average case complexity is only one of the many measures to quantify the hard problem of scalability.

With the above note, in Table 2 summarizes the relevant complexities of the interesting Sub-Polyhedra along with the same complexities on general polyhedra, each defined in n variables, m inequalities and d dimensions.

The $\mathcal{O}_c$-notation is to denote the worstcase measure when the particular sub-polyhedron is apriori in closed form. Rest of the complexities in the table refer to worst case complexities, except for the Fourier-Motzkin on Convex Polyhedra, where we have used lower bounds. NPC and WNPC stand for NP-Complete and Weakly NP-Complete complexities respectively.

2.4 Algorithms to approximate into sub-polyhedra

A general polyhedron has to be converted into a Sub-Polyhedra for it use the algorithms or libraries that specialize in it. An approximation can be an over-approximation or under-approximation. Whichever be the kind of approximation, broadly, here are some issues that the approximation algorithm has to satisfy:

- **Legality:** The approximated polyhedron should not yield illegal optimizations.
- **Cost:** Small scalable running time: The conversion process should be asymptotically small (polynomial-time) as well as appreciably simpler.
- **Closest:** The approximated polyhedron should be as close to the original polyhedron as possible. In case of rational approximation, the volume of the approximated sub-polyhedron should be as close to the original one as possible. In case of integer approximation, it should leave out as few number of integer points as possible.
- **Expressiveness:** The approximation should give non-trivial approximations for sizable number of interesting inputs.

There are some more details that the algorithm that needs to satisfy. We will point them in Section 4 and Section 5.

From the sub-polyhedral literature, we are not aware of any systematic study of under-approximation algorithms. We however are aware of two algorithms on over-approximation. Here is a summary of them.

2.4.1 Interval over-approximation

Interval over-approximation of an arbitrary polyhedron is a trivial approximation and has been assumed and solved many times in polyhedral literature.

For a system with n variables and m constraints, one can run $2n$ linear programs, twice per each variable x_i, with the objective function being $\max(x_i)$ and $\min(x_i)$, each returning the upper-bound and lower-bound respectively of variable x_i.

The above method is simple, and has cost $2nZ$, which on average case takes $\mathcal{O}(n^4)$ time, and could be quite affordable if n is small. Further, the method can use an existing simplex implementation.

2.4.2 UTVPI over-approximation by Miné

Miné (in [Min06], section 4.3) has proposed an algorithm to over-approximate a general polyhedron into a UTVPI-over-approximation. The algorithm however, uses the generator representation and has the cost $\mathcal{O}(d^2(|R| + |V|))$, where d is the dimension and $|R|$ and $|V|$ are the number of rays and vertices respectively. It returns the tightest over-approximation-UTVPI of the given Polyhedron.

Briefly, the method uses a greedy approach of incrementally building an Octagon, exploiting another already developed algorithm to do convex union of two octagons and octagonalize the result. The over-approximation algorithm begins with an arbitrary vertex of the input polyhedron, which is trivially an Octagon and iteratively adds vertices to it, each time obtaining an incremental octagonal convex-hull for that particular set of vertices which have been over-approximated. A similar procedure is repeated for each ray of the polyhedron as well. The result of this algorithm is the tightest over-approximation for the input polyhedron.

Problem	Convex Polyhedra	TVPI	UTVPI/DBM
GEN ($\mathcal{V} \leftrightarrow \mathcal{H}$)	$\mathrm{EXP}(n)$, $\mathrm{EXP}(d)$	$\mathrm{EXP}(n)$, $\mathrm{EXP}(d)$	$\mathrm{EXP}(n)$, $\mathrm{EXP}(d)$
LP-OPT	WNPC	$\mathcal{O}(m^3n^2 \log B)$	$\mathcal{O}(m^3n^2 \log B)$
LP-OPT (2 variables in objective function)	WNPC	$\mathcal{O}_c(\log m)$	$\mathcal{O}_c(1)$
LP-FEAS	WNPC	$\mathcal{O}(mn^2 \log m)$	$\mathcal{O}(mn^2 \log m)$
FM	$\mathrm{EXP}(n)$, $\mathrm{EXP}(d)$	$\mathcal{O}(mn^2 \log m)$	$\mathcal{O}(mn^2 \log m)$
CLOSURE	–	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
ILP	NPC	NPC	NPC?

Table 2. General and Sub Polyhedra: Comparison of (Worstcase) Complexities

Needless to say, the method given uses the generator representation of the polyhedron which could be costly. We are aware of no efficient algorithm to directly compute the UTVPI over-approximation using only the constraint representation.

2.4.3 TVPI over-approximation by Simon-King-Howe

Simon, King and Howe (in [SKH10], Section 3.2.6) propose an algorithm that approximates a constraint $a_1x_1 + \cdots + a_nx_n \leq c$ of a Polyhedron P using the set of constraints $a_jx_j + a_kx_k \leq c - c_{j,k}$ for every $1 \leq j < k \leq n$. They propose that each $c_{j,k}$ can be calculated as the result of a linear programming problem: $c_{j,k} = minExp(\sum_{i\in[1,n]\setminus\{j,k\}} a_ix_i, \mathrm{OA}(P_i))$ and $\mathrm{OA}(P_i)$ is the incremental over-approximation polyhedron obtained at the i'th step of computation. The above step[1] needs to be iterated for each non-TVPI constraint of P.

For converting a general polyhedron with n variables and m constraints into a TVPI over-approximation, it can be seen that the above algorithm has cost n^2m^2Z, which in average case takes $\mathcal{O}(n^7)$ time. This could be quite expensive, unless n is very small.

Further, the above method is hampered by unbounded variables in the original set of constraints. For example the polyhedron $\{x, y, z | x + y + z \leq 1\}$ cannot be over-approximated by the above method as $c_{j,k}$ variables are returned as unbounded variables each time in the calls to $minExp$!

Other than the above result by Simon, King and Howe, as far as we know, this process of efficiently over-approximation of a general linear program using TVPI constraints is an open problem. In their paper on Logahedra [HK09] (Section 4.4), Howe and King propose an algorithm to approximate a finite set of Poly constraints into Logahedra constraints. It remains to be seen if this method can be extended to under/over-approximate a Poly system from scratch.

2.4.4 Under-approximation through over-approximation

To obtain an sub-polyhedral under-approximation of a rational polyhedron $\mathcal{P}$, one can transform the polyhedron into the dual-space obtaining $\mathcal{P}^*$, and then obtain the over-approximation of the dual-polyhedron $\mathrm{OA}(\mathcal{P}^*)$ and convert the over-approximated dual polyhedron into primal space ($\mathrm{OA}(\mathcal{P}^*) \rightarrow \mathrm{UA}(\mathcal{P})$) to get an under-approximation $\mathrm{UA}(\mathcal{P})$. In effect, we will be doing $\mathcal{P} \rightarrow \mathcal{P}^* \rightsquigarrow \mathrm{OA}(\mathcal{P}^*) \rightarrow \mathrm{UA}(\mathcal{P})$. A similar procedure can be used for the reverse process as well. The proof of these methods[2] is trivial using a simple convexity argument on the polyhedra involved.

The above procedures make the under-approximation and over-approximation as asymptotically comparable methods, but for the transformation into dual-space. In case when the constraints/vertices of the input polyhedron are too many, then a cheaper direct method could be used.

We are also not aware of any work on integer TVPI (or UTVPI) approximations, other than the obvious ones of directly using the above suggested rational approximations. Obviously, using such methods could yield a very weak approximations. Further literature study is needed, though we suspect that integer approximations may be NP-Complete problems.

[1] One way of looking at the above step is that it fills in $c_{j,k}$ entries into a lower-diagonal matrix of size $n \times n$.

[2] These methods work only when $0 \in \mathcal{P}$. Otherwise, the polyhedra should be shifted so as to satisfy this condition. We believe that it can be done with low complexity, or it can be ensured that the input itself can be modified accordingly.

3. Sub-polyhedral compilation approaches

Whichever be the flavor of sub-polyhedra, there are some choices on how one may use that flavor depending on the precise decision or optimization problem.

Dependence approximation One may assume that the dependence polyhedra satisfy some approximation criterion. The input to the optimization or parallelization problem is chosen to belong to a pre-determined flavor of sub-polyhedra. The dependences which do not satisfy those criterion must be *over-approximated* by an appropriately designed heuristic. Some degrees of freedom may be lost, but only correct transformations will be modeled. This necessary approximation is in the same spirit as classical abstractions and algorithms presented in [YAI94, DRV00]. We will discuss dependence approximation with TVPI sub-polyhedra in Section 4.

Constraint approximation Another choice is to keep dependence relations as general polyhedra, but approximate the constraints on the affine transformation itself. After the transformation to the dual space through the Farkas lemma, the constraint polyhedron — the (μ, λ)-polyhedron of Farkas multipliers — must be *under-approximated*. Again, some legal affine transformations may be lost. We will discuss constraint approximation with TVPI sub-polyhedra in Section 5.

Additional approximations There are at least two more ways in which sub-polyhedra could be used in polyhedral compilation. These are *domain approximation* and *objective function approximation*. In most cases, iteration domains (loop bounds) form multi-dimensional intervals, and most non-interval cases belong to the UTVPI flavor (triangular loops). Approximations are possible, but require specific handling in the code generator to generate precise iteration domain traversals; we believe the problem can be reduced to code generation for irregular, data-dependent control flow [BPCB10]. Regarding the approximation of objective functions, the strongly polynomial time bounds for TVPI discussed in Section 2 only hold for a specific class of optimization problems (to which the lexicographic minimum can be reduced). When the objective function is arbitrary, linear programming seems to be required. This is discussed in detail in the literature [HN94, CM94, Way99]. We believe solutions can be found to enable both kinds of approximations, but leave these as open problems.

4. Dependence Approximation Using TVPI

The goal of this section is to progress towards the design of a Feautrier-like scheduling algorithm building on the previously discussed sub-polyhedral techniques. It is an open ended section at this point.

4.1 Existing dependence approximations

Over approximations of dependence polyhedra is a fairly well known technique. This kind of dependence approximation leads to loss of precision in the input itself. Different dependence abstractions have been proposed for dealing with loop nest optimization and parallelization. They have been summarized in [YAI94] as well as in [DRV00]. Our view of them is summarized in the following hierarchy, where DL stands for dependence levels, DDV for dependence direction vectors, DC for dependence cones, D for dependence difference polyhedron, DP for dependence polyhedron and DI for dependence $\mathbb{Z}$-polyhedron:

$$\underbrace{\text{DL} \subset \text{DDV} \subset \text{DC}}_{\text{sub-polyhedral}} \subset \underbrace{\text{D} \subset \text{DP}}_{\text{polyhedral}} \subset \underbrace{\text{DI} \subset \cdots}_{\text{beyond polyhedal}}$$

Interestingly, the above hierarchy of abstractions is largely different from the one introduced by the static analysis community and described in Section 2:

$$\text{Intervals} \subset \text{DBM} \subset \text{Octagons} \subset \text{TVPI} \subset \text{Poly}$$

TVPI polyhedra are not directly comparable with other abstractions such as DC, but it is larger than DDV and smaller than D.

All the above varieties of abstractions are different approximations of DP (and of DI, and of the exact, potentially non-affine, extensional relation). Considering dependence relations, an approximation is always an over-approximation; it will only yield correct transformations although it may induce a loss of potentially interesting schedules.

The overall motivation for the above sub-polyhedral abstractions, as well as the original papers that proposed them, has been to solve a particular parallelization algorithm or transformation. For example, DL has been proposed for data-parallelism detection and loop reversal, and DDV for all unimodular transformations.

But the goal of the comparison by Yang et al. [YAI94] has not been to find whether the particular dependence abstraction *scales* well, but whether the particular abstraction *fits* well its application purpose.

We already know that TVPI is associated with excellent strongly polynomial algorithms. But we still need to study how to approximate a polyhedral dependence relation into a TVPI, and then to study how TVPI dependence relations *fit* their purpose in affine transformation methods such as Feautrier's scheduling algorithm.

4.2 TVPI approximation of dependence polyhedra

As pointed out in early sections, TVPI has been discovered by theoreticians to have a scalable linear programming solution and has been applied by static analysis people to their problems. We point out the difficulties that rise when trying to use the state of the art techniques to polyhedral compilation.

Is there a need for parametric TVPI? As a means of motivation for TVPI approximation of dependences, we point out that none of the existing sub-polyhedral abstractions (DL, DDV and DC) deal with parameters. The primary assumption of non-affine transformation algorithms — including the polyhedral Darte-Vivien algorithm — on such approximations is that parameters are unbounded at compile time.

Feautrier's polyhedral approximation of dependences (as well as the more powerful DI abstraction) are really different from the ones lower in the hierarchy because they can handle the parameters symbolically. A parameter can be considered a constant that is not known at compile time, which is not equivalent to considering it as an additional index-variable in several key problems, including linear programming and piece-wise feasibility with respect to the parametric context.

But the surprising fact is that Farkas-based affine transformations do not need to solve a parametric linear programming formulation at all!

Hence it is clear that though solving parametric linear programs cannot be avoided for solving the dependence analysis problem, it can however be avoided for affine transformation algorithms using the Farkas lemma or Vertex method. Such algorithms can use existing sub-polyhedra without the need for a paramteric extension.

Yet more complex affine transformations such as index-set splitting or iteration-space slicing may need parameterized algorithms at their core.

The cost of parametric sub-polyhedra For the problems that do benefit from parameterization, it would be tempting to take sub-polyhedra (like UTVPI or TVPI) and add parameterization to it and pay a small additional cost, while still retaining the strong-polynomiality bounds. The main turning point in polyhedral compilation is the work by Feautrier on PIP [Fea88]. Using PIP, one can represent constraints of the form $aI + bN \leq c$, where I is the iterator vector, and N is the parameter vector. But, it turns out that parameterization itself is an exponential problem, since it has to decide on the sign of a parametric expressions. Therefore, even if the non-parametric part of the sub-polyhedra could be solved in linear time, like in intervals, the parametric part may still take exponential time.

Looking for a parametric TVPI abstract domain In the case of parameterization of TVPI, one can allow with the above assumptions only three kinds of scenarios: II, IP and PP, when the constraints are of the form $ai + bj \leq c$, $ai + bn \leq c$ and $an + bm \leq c$ respectively.

A general extension remains to be designed that preserves the scalability of TVPI algorithms. We identified two early directions:

- symbolic computation with parameters, through the existing Fourier-Motzkin method for TVPI or min-cost-flow algorithms, with on-demand elimination of the parameter, approximating it with its bounds in the parametric context;
- in some JIT compilation scenarios with partial evaluation opportunities, instantiate some parameters and resort to non-parametric TVPI algorithms; since JIT compilation is one major motivation for this work, this path is worth exploring further.

Approximation into parametric TVPI Here we are talking about the algorithm that converts a general dependence polyhedron into a sub-polyhedral over-approximation. One step to meaningfully resolve the parameters in using polyhedra is to make them unbounded, like even Darte-Vivien do.

But parameters can also be eliminated by assuming some context. Or, Even otherwise, their number can be reduced by a simple Fourier-Motzkin elimination. For example, if a dependence polyhedron has constraint $n_1 > n_2$, then all occurrences of n_2 can be replaced by n_1. Needless to say, one loses some precision, but the dependence polyhedron remains parametric, being more powerful than Darte-Vivien's dependence cone approximation.

4.3 Towards a Feautrier-TVPI algorithm?

In the previous section, we assumed that a dependence analysis will result in a reduced dependence graph annotated with TVPI polyhedra. A first strategy in scheduling this approximate dependence graph is to use an unmodified, general scheduling method

like Feautrier's. The question remains as to what the complexity measures the strategy discussed in the previous section, will bring.

In Feautrier's algorithm, there are two applications of the Farkas lemma. The first one is expressed in terms of variables (μ, I, N). The second one in terms of variables (λ, I, N). The above two polyhedra are "equated" in a matching step, with the equality resulting from convexity properties (as proven by Feautrier).

The result of the above equations is a large system, $\mathcal{P} = \cap_{e \in E} \mathcal{P}_e = (\mu, \lambda)$.

The dependence polyhedron $\mathcal{D}_e$ satisfying some property, like it being a TVPI polyhedron, does not necessarily mean that $\mathcal{P}_e$ has to satisfy the same requirement. Neither are they directly related to each other by duality. But, we think that some measures of the good bounds on TVPI should reflect on $\mathcal{P}_e$ as well. Regarding the problem of the existence of strongly polynomial time bounds on solving $\mathcal{P}$, we think that understanding the duality of TVPI polyhedra is a key.

This path of investigation is currently open. Empirical evaluation will also be needed.

5. Constraint Approximation Using TVPI

In Feautrier's algorithm [Fea92], the dependence constraints of a particular dependence edge are pseudo-linear (or quasi-affine) constraints involving (μ, I, N). These are converted into a $\mathcal{P}_e = (\mu, \lambda)$ system by application of affine form of Farkas-lemma, where the newly created λ-variables (along with μ-variables) are called the *Farkas multipliers*. The set of constraints one obtains is a polyhedron $\mathcal{P} = \cap_{e \in E} \mathcal{P}_e$. The polyhedron $\mathcal{P}$ is amenable to Linear Programming or Fourier-Motzkin elimination and any rational point that satisfies $\mathcal{P}$ is considered a valid schedule.

Note that $\mathcal{P}$ is *not* a parametric polyhedron: classical non-parametric algorithms can be used. This is an encouraging observation for the effectiveness of TVPI sub-polyhedral approaches.

Feautrier suggests preprocessing $\mathcal{P}$ with an initial Gaussian elimination step to remove some Farkas multipliers. This is possible because the Farkas multipliers occur in equations and the elimination can be done on a per-dependence basis. But, this step will *not* remove all the Farkas multipliers as $\mathcal{P}_e$ is too under-constrained at least with respect to the λ-variables. This is complicated by the fact that the systems $\mathcal{P}_e$ and $\mathcal{P}_{e'}$ on different edges e and e' are not independent systems with respect to the μ-variables.

Our assumption in this section is that $\mathcal{P}$ has too many variables and constraints, and hence solving it will be asymptotically unscalable. As no strongly polynomial algorithm exists for linear programming, solving $\mathcal{P}$ means we will not meet our scalability criterion.

In this section, we propose two methods to obviate this difficulty using Sub Polyhedra. Firstly, we give a rough measure of algorithmic complexity. Next, we discuss two previous techniques which do not use Farkas lemma. Next, we briefly discuss another two previous techniques which have similar motivation as ours. Next, we give two approaches which use sub-polyhedra to alleviate the scalability problem. Next, we give examples and in the final sub-section, pose a lot of open questions.

5.1 Scalability issues in Farkas-based methods

Consider a dependence multigraph (V, E). Let $n = |V|$ be the number of statements in the program and $m = |E|$ be the number of edges. We can assume that all statements have the same dimension and that the maximum dimension of any statement in the program is d — counting both iteration and parameter dimensions. Let n_k and m_k be the respective number of variables and constraints of the domain polyhedron for statement k. The polyhedron of a dependence $S_i \rightarrow S_j$ between two statements i and j has dimension $2d$.[3] Let n_e and m_e be the respective number of variables and constraints of the polyhedron of dependence edge e.

Applying Feautrier's one-dimensional scheduling, we have one λ variable per dependence and per dependence constraint. So, total number of λ variables is $2d\Sigma_{e \in E} m_e = 2dm\hat{m_E}$, where $\hat{m_E}$ is the average number of constraints per dependence polyhedron. Also, we have one μ variable per statement and per domain constraint. So the total number of μ variables is $d\Sigma_{k \in V} m_k = dn\hat{m_V}$, where $\hat{m_V}$ is the average number of constraints per domain polyhedron. Besides, the total number of constraints in the complete system is dm.

Assuming an average cubic complexity of linear programming (the best algorithm for state-of-the-art Farkas-based methods), this brings the total complexity to the order of $d^3m^3(\hat{m_V} + \hat{m_E})^3$. The m^3 factor may get closer to n^6 on programs with many dependences, or worse in presence of piecewise-affine dependence relations (e.g., in some cases after array data-flow analysis). This is clearly not scalable.

Analyzing the Pluto algorithm leads to a similar computation and the same non-scalability result. Practical evidence shows that it scales better on all benchmarks considered so far. But no benchmark with thousands of loops and statements have been processed either, which will no doubt happen with three-address code representations [TCE+10] and extensions to irregular, data-dependent control flow [BPCB10]. Considering Feautrier multi-dimensional scheduling only makes things worse, as it involves solving a mixed-integer linear programming problem.

5.2 Two non Farkas-based methods

To address this scalability challenge, one may first consider alternatives to Farkas-based methods. Two of them come to mind within the polyhedral model.

The Vertex method It was the most general "pre-Feautrier" affine scheduling algorithm [RPF86, QD89]. In this method, non-linear constraints on schedule coefficients (as variables) are avoided by converting dependence constraints from the hyperplane to the dual vertex/ray representation followed by an instantiation of the iteration variables. This process leads to a constraint system with too many variables and constraints: practical dependence polyhedra are likely to have a very small hyperplane representation while having an exponential number of vertices. The method has more scalability limitations than Farkas-based scheduling, as shown by Feautrier [Fea92] and confirmed experimentally by Balev [BQRR98].

Darte-Vivien's method The Darte-Vivien algorithm [DV97] considers dependence cone approximations of distance vectors and abstracts away all parameters. The algorithm uses an uniformization of affine dependences, building on the technique introduced by Karp-Miller-Winograd. Although optimality of the number of parallel loops has been proven, much expressiveness is lost in terms of multi-dimensional affine transformation beyond the innermost loops [DV97].

As far as we know, an empirical study of the Darte-Vivien algorithm vis-a-vis Feautrier's method is missing. The loss of expressiveness may not be worth the complexity and scalability gain. Further study is needed.

5.3 Two Farkas-based approaches

Within the framework suggested by Feautrier [Fea92], two approaches have recently been suggested which have very similar goals as ours.

Feautrier's scalable modular scheduling Feautrier's approach [Fea06] starts with Gaussian elimination, and combines a Minkowski

[3] This is an upper bound: $2d$ minus the number of parameters to be precise.

decomposition of $\mathcal{P}_e$ with a modular parametric linear programming algorithm.

We consider this approach as complementary to ours. It does not address scalability without making some assumptions on a modular decomposition of the problem, but Feautrier's approach may have a positive influence on the precision of the sub-polyhedral abstractions as well as on practical scalability.

Pouchet's FM library and heuristics The other notable work in avoiding the cost of solving the systems is implemented in Pouchet's redundancy aware Fourier-Motzkin (FM) library. Though a detailed comparison is needed, we note that Pouchet's method does not asymptotically reduce the complexity in a predictable way. Nevertheless, it is clear that $\mathcal{P}$ contains many valid yet useless transformations, and many redundant ones in terms of generated code or behavior on the target platform. This is why we consider our work to be complementary to the FM approach and the heuristics therein.

5.4 Two new approaches using sub-polyhedra

In this subsection, we suggest two ways of using sub-polyhedra to alleviate the scalability problem. Firstly, we suggest a direct method using under-approximation of sub-polyhedra. Later, we sketch a method that uses existing over-approximation methods through a simple duality argument.

A direct approximation method If each $\mathcal{P}_e$ can be *under-approximated* using a TVPI, then the resulting system can use the Fourier-Motzkin technique developed by Hochbaum-Naor in [HN94]. We suggest the following:

1. Approximate each $\mathcal{P}_e$ into a TVPI-polyhedron UA$(\mathcal{P}_e)$, using the method suggested in Section 2.4.
2. Build UA$(\mathcal{P}) = \cap_{e\in E}UA(\mathcal{P}_e)$ as suggested by Feautrier to build the complete constraint system.
3. Solve the system UA$(\mathcal{P})$ using Hochbaum-Naor's Fourier-Motzkin technique [HN94].

In Step 1 of the above method, as the individual constraint polyhedra ($\mathcal{P}_e$'s) could be quite small, we could perhaps even use the polynomial time approximation method suggested in Section 2.4.3 by Simon-King-Howe (albeit with $\mathcal{O}(n^7)$ worst case complexity) or King-Howe. As these algorithms give an over-approximation, one has to use the duality transformation to obtain an under-approximation. Step 2 of the above method is exactly as suggested by Feautrier and involves collating the under-approximated sub-polyhedral systems. Step 3 of the above method uses the Fourier-Motzkin pruning technique of Hochbaum-Naor to determine the feasibility of a linear system, and thus benefits from the strongly polynomial time bounds (with $\mathcal{O}(mn^2 \log m)$ worst case complexity) therein.

We claim that a valid schedule can be determined using the above method in strongly polynomial time. It follows directly from the polynomial-time claims of under-approximation as well as the bounds of Hochbaum-Naor's algorithm.

Indirect approximation using duality Alternatively, the duality theorem be used on each $\mathcal{P}_e$ and each dual polyhedron $\mathcal{P}_e^*$ be *over-approximated*[4] by a min-cost-flow polyhedron (or even a generalized min-cost-flow polyhedron, as defined by Wayne [Way99]). The resultant over-approximation's dual polyhedron will be a TVPI polyhedron. We are not aware of any literature on the subject, but we think that it may be implicit in the work of Cohen-Megiddo in [CM94]. More study is needed to assess the effectiveness of this approach.

[4] This over-approximation is *not* the same as the dependence over-approximation discussed in Section 4. Here, it is the dual of the constraint system that is being over-approximated, while there, it is the dependence polyhedron that is being over-approximated.

5.5 Example

We illustrate the direct under-approximation approach on one Feautrier's one-dimensional scheduling Example 1 [Fea92] (or Example 20, page 220–222 in Darte et al. [DRV00]),

$$\begin{array}{ccccccccccc}
\mu_{2,1} & - & \mu_{2,2} & - & \mu_{1,1} & + & \mu_{1,2} & = & \lambda_{1,1} & - & \lambda_{1,2} \\
 & & & & \mu_{2,3} & - & \mu_{2,4} & = & \lambda_{1,3} & - & \lambda_{1,4} \\
 & & \mu_{2,2} & + & \mu_{2,4} & - & \mu_{1,2} & = & \lambda_{1,2} & & \\
 & & \mu_{2,0} & - & \mu_{1,0} & - & 1 & = & \lambda_{1,0} & &
\end{array}$$

After simplification by Gaussian elimination, followed by removal of λ-variables, the above becomes

$$\begin{array}{lccccccc}
C_1: & & & \mu_{2,0} & \geq & \mu_{1,0} & + & 1 \\
C_2: & \mu_{2,1} & + & \mu_{2,4} & \geq & \mu_{1,1} & & \\
C_3: & \mu_{2,2} & + & \mu_{2,4} & \geq & \mu_{1,2} & & \\
C_4: & & & \mu_{2,3} & \geq & \mu_{2,4} & + & 1
\end{array}$$

Let us denote the sub-systems as $\mathcal{P}_1 = \{C_1, C_2, C_3\}$ from dependence $\mathcal{D}_1$ and $\mathcal{P}_2 = \{C_4\}$ from dependence $\mathcal{D}_2$. It can be seen that the sub-systems $\mathcal{P}_1$ and $\mathcal{P}_2$ are not independent, sharing variable $\mu_{2,4}$ with each other. Feautrier suggests two schedules to the above system.

Schedule 1: Arbitrarily choose $\mu_{1,0} = \mu_{1,1} = \mu_{1,2} = \mu_{2,1} = \mu_{2,2} = \mu_{2,4} = 0$ and $\mu_{2,0} = \mu_{2,3} = 1$ which leads to the schedules $\{\theta(S_1, i, N) = 0; \theta(S_2, i, j, N) = j + 1\}$.

Schedule 2: This time, one can choose $\mu_{1,0} = \mu_{1,2} = \mu_{2,2} = \mu_{2,4} = 0$ and which leads to $\mu_{2,1} = \mu_{1,1} = 1$. The schedules are $\{\theta(S_1, i, N) = i; \theta(S_2, i, j, N) = i + j + 1\}$.

Let us now study three sub-polyhedral abstract domains and the impact on the affine schedules, in an increasing order of expressiveness. All three abstract domains are amenable to strongly polynomial algorithms to compute such schedules.

Interval under-approximation Neither of the sub-systems $\mathcal{P}_1$ and $\mathcal{P}_2$ are Intervals. One can however under-approximate each of them so that the resultant sub-systems are interval sub-polyhedra. Here is one possibility among many others.

For $\mathcal{P}_2$, choose the under-approximation Interval$(\mathcal{P}_2) = \{\mu_{2,3} \geq 1, \mu_{2,4} = 0\}$ which is an interval. This leads to $\mathcal{P}_1$ having the system $\{\mu_{2,0} \geq \mu_{1,0}+1, \mu_{2,1} \geq \mu_{1,1}, \mu_{2,2} \geq \mu_{1,2}\}$, which is still not an interval. One can then choose $\{\mu_{1,0} = 0, \mu_{1,1} = 0, \mu_{1,2} = 0\}$ leading to Interval$(\mathcal{P}_1) = \{\mu_{2,0} \geq 1, \mu_{2,1} \geq 0, \mu_{2,2} \geq 0\}$. The overall system is $\{\mu_{2,3} \geq 1, \mu_{2,4} = 0, \mu_{2,0} \geq 1, \mu_{2,1} \geq 0, \mu_{2,2} \geq 0\}$. One solution of the system is $\{\mu_{2,3} = 1, \mu_{2,4} = 0, \mu_{2,0} = 1, \mu_{2,1} = 0, \mu_{2,2} = 0\}$, which results in schedules $\{\theta(S_1, i, N) = 0; \theta(S_2, i, j, N) = j + 1\}$, which is same as Schedule 1 as illustrated by Feautrier.

UTVPI under-approximation It can be observed that $\mathcal{P}_2$ is already a UTVPI. On the other hand, $\mathcal{P}_1$ is "mostly" a UTVPI with constraint C_1 already being a UTVPI needing no approximation, while C_2 and C_3 have to be under-approximated into UTVPI. To obtain the UTVPI under-approximation intuition, one can observe that $\mu_{2,4}$ is a shared variable between the non-UTVPI constraints C_2, C_3 and UTVPI constraint C_4. Since there is no other shared variable, we are free to choose any non-negative values to them, without effecting the others. The variable $\mu_{2,4}$ along with each of the pairs of variables $\{\mu_{2,1}, \mu_{1,1}\}$ and $\{\mu_{2,2}, \mu_{1,2}\}$, creates halfspaces passing though origin dividing their corresponding $3d$-space accordingly. One can simply set the value $\mu_{2,4} = 1$ leading to UA$(C_2) = \{\mu_{2,1} + 1 \geq \mu_{1,1}\}$ and UA$(C_3) = \{\mu_{2,2} + 1 \geq \mu_{1,2}\}$. Solving this system and choosing the other variables as $\mu_{1,0} =$

$\mu_{1,1} = \mu_{1,2} = 1$ and $\mu_{2,0} = \mu_{2,1} = \mu_{2,2} = 0$ and leads to the schedules $\{\theta(S_1, i, N) = 1 + N; \theta(S_2, i, j, N) = j + N\}$.

TVPI under-approximation Since $\mathcal{P}_2$ is already a UTVPI it is a TVPI as well, while $\mathcal{P}_1$ is needs to be under-approximated into one. The above UTVPI under-approximations are valid as TVPI under-approximations well.

6. Future Work

We have raised many questions and provided very few answers to the scalability challenge. In the near future, we would like to collect and/or synthesize examples that demonstrate empirically the unscalability of (integer) linear programming and Fourier-Motzkin techniques. We would also like to validate one or more strongly polynomial approaches based on TVPI or UTVPI on these examples. We will use existing frameworks like GRAPHITE and PoCC for this purpose.

Acknowledgments We are grateful to Paul Feautrier for the tremendously helpful suggestions and feedback on our work. We are also thankful for Axel Simon, as well as the reviewers of IMPACT-2011 for a careful reading of the paper and the valuable feedback offered.

References

[AS80] Bengt Aspvall and Yossi Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM J. Comput.*, 9(4):827–845, 1980.

[Bag97] R. Bagnara. *Data-Flow Analysis for Constraint Logic-Based Languages*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy, March 1997. Printed as Report TD-1/97.

[BHRS08] Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In Rajiv Gupta and Saman P. Amarasinghe, editors, *PLDI*, pages 101–113. ACM, 2008.

[BHZ09] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. Weakly-relational shapes for numeric abstractions: improved algorithms and proofs of correctness. *Formal Methods in System Design*, 35(3):279–323, 2009.

[BK89] Vasanth Balasundaram and Ken Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. In *PLDI*, pages 41–53, 1989.

[BPCB10] Mohamed-Walid Benabderrahmane, Louis-Noël Pouchet, Albert Cohen, and Cédric Bastoul. The polyhedral model is more widely applicable than you think. In *Proceedings of the International Conference on Compiler Construction (ETAPS CC'10)*, number 6011 in LNCS, Paphos, Cyprus, March 2010. Springer-Verlag.

[BQRR98] Stephan Balev, Patrice Quinton, Sanjay V. Rajopadhye, and Tanguy Risset. Linear programming models for scheduling systems of affine recurrence equations - a comparative study. In *SPAA*, pages 250–258, 1998.

[CC04] Robert Clarisó and Jordi Cortadella. The octahedron abstract domain. In Roberto Giacobazzi, editor, *SAS*, volume 3148 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2004.

[CM94] Edith Cohen and Nimrod Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.*, 23:1313–1350, December 1994.

[DRV00] Alain Darte, Yves Robert, and Frédéric Vivien. *Scheduling and Automatic Parallelization*. Birkhaüser, 2000.

[DV97] Alain Darte and Frédéric Vivien. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. *Int. J. Parallel Program.*, 25:447–496, December 1997.

[Fea88] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22(3):243–268, 1988.

[Fea92] Paul Feautrier. Some efficient solutions to the affine scheduling problem: I. one-dimensional time. *Int. J. Parallel Program.*, 21:313–348, October 1992.

[Fea06] Paul Feautrier. Scalable and structured scheduling. *International Journal of Parallel Programming*, 34(5):459–487, 2006.

[GFG02] Martin Griebl, Paul Feautrier, and Armin Größlinger. Forward communication only placements and their use for parallel program construction. In William Pugh and Chau-Wen Tseng, editors, *LCPC*, volume 2481 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2002.

[HK09] Jacob M. Howe and Andy King. Logahedra: A new weakly relational domain. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis*, ATVA '09, pages 306–320, Berlin, Heidelberg, 2009. Springer-Verlag.

[HMG06] N. Halbwachs, D. Merchat, and L. Gonnord. Some ways to reduce the space dimension in polyhedra computations. *Form. Methods Syst. Des.*, 29:79–95, July 2006.

[HN94] Dorit S. Hochbaum and Joseph Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, 1994.

[Lag85] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM J. Comput.*, 14(1):196–209, 1985.

[LL09] Vincent Laviron and Francesco Logozzo. Subpolyhedra: A (more) scalable approach to infer linear inequalities. In Neil D. Jones and Markus Müller-Olm, editors, *VMCAI*, volume 5403 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2009.

[Min01a] Antoine Miné. A new numerical abstract domain based on difference-bound matrices. In Olivier Danvy and Andrzej Filinski, editors, *PADO*, volume 2053 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2001.

[Min01b] Antoine Miné. The octagon abstract domain. In *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, WCRE '01, pages 310–, Washington, DC, USA, 2001. IEEE Computer Society.

[Min06] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[PBB+11] Louis-Noël Pouchet, Uday Bondhugula, Cédric Bastoul, Albert Cohen, J. Ramanujam, P. Sadayappan, and Nicolas Vasilache. Loop transformations: convexity, pruning and optimization. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 549–562. ACM, 2011.

[QD89] Patrice Quinton and Vincent Van Dongen. The mapping of linear recurrence equations on regular arrays. *VLSI Signal Processing*, 1(2):95–113, 1989.

[RPF86] Sanjay V. Rajopadhye, S. Purushothaman, and Richard Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In Kesav V. Nori, editor, *FSTTCS*, volume 241 of *Lecture Notes in Computer Science*, pages 488–503. Springer, 1986.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[Sho81] Robert Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28:769–779, October 1981.

[SKH02] Axel Simon, Andy King, and Jacob M. Howe. Two variables per linear inequality as an abstract domain. In Michael Leuschel, editor, *LOPSTR*, volume 2664 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2002.

[SKH10] A. Simon, A. King, and J. M. Howe. The Two Variable Per Inequality Abstract Domain. *Higher Order and Symbolic Computation*, 2010.

[SKS00] Ran Shaham, Elliot K. Kolodner, and Shmuel Sagiv. Automatic removal of array memory leaks in java. In David A. Watt, editor, *CC*, volume 1781 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2000.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51:385–463, May 2004.

[TCE+10] K. Trifunovic, A. Cohen, D. Edelsohn, F. Li, T. Grosser, H. Jagasia, R. Ladelsky, S. Pop, J. Sjödin, and R. Upadrasta. Graphite two years after: First lessons learned from real-world polyhedral compilation. In *GCC Research Opportunities Workshop (GROW'10)*, Pisa, Italy, January 2010.

[Way99] Kevin D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 11–18, New York, NY, USA, 1999. ACM.

[YAI94] Yi-Qing Yang, Corinne Ancourt, and François Irigoin. Minimal data dependence abstractions for loop transformations. In Keshav Pingali, Utpal Banerjee, David Gelernter, Alexandru Nicolau, and David A. Padua, editors, *LCPC*, volume 892 of *Lecture Notes in Computer Science*, pages 201–216. Springer, 1994.