Mechanically Proving Approximation Bounds on Lattice-Linear Parallel Algorithms

Abdullah Rasheed

1 Introduction

Lattice-Linear Predicates (LLPs) describe a class of predicates defined over lattices of vectors for which the problem of finding a minimal satisfying vector has a simple parallel/distributed algorithm. LLPs have been shown to be effective at parallelizing a variety of well-known optimization problems improving upon existing solutions. Garg's algorithm for detecting LLPs runs in time proportional to the height of the lattice, offering a large cut-down on the solution space [3]. For lattices with very large height, however, it is simply infeasible to parallelize any solution. Many such problems are NP-hard, and thus if P $\neq$ NP, then there does not exist any algorithm for them in NC.

To combat the hardness of such problems, a common remedy is to resort to approximations. While LLPs describe optimization problems, it is hard to encode approximation algorithms as LLPs because of their optimality. In this project, we discover a method for exploiting the lattice via abstraction to implicitly achieve *non-compounding* approximations with LLPs. Furthermore, we represent this method in ACL2 to mechanically construct robust proofs of correctness.

2 Background

A lattice is a partially-ordered set where each pair of elements has a well-defined least upper bound (join) and greatest lower bound (meet). The lattice is said to be distributive if join distributes over meet (equivalent to meet distributes over join). We will assume distributivity in our lattices.

A Galois connection between two lattices A and B is a pair of functions $\alpha : A \to B$ and $\gamma : B \to A$ such that $\gamma(\alpha(a)) \geq a$ and $\alpha(\gamma(b)) \leq b$. A Galois connection is called a Galois insertion (GI) if $\alpha(\gamma(b)) = b$. That is, B is exactly representable in A. We will assume only GIs throughout this project since is well-known that a Galois connection can be reduced to a GI. It is also known that α uniquely determines γ, and vice versa. Finally, it is known that for order-preserving $f : A \to A$, $\alpha \circ f \circ \gamma \leq g$ pointwise for all $g : B \to B$ such that $\alpha \circ f \leq g \circ \alpha$. These well-known properties allow us to simplify the project, and we do not verify them in ACL2.

d is a quasi-metric on A if for all $x, y, z \in A$, $d(x, y) = 0$ iff $x = y$, $d(x, y) > 0$ iff $x > y$, and $d(x, y) + d(y, z) \geq d(x, z)$. For a practical implementation, a specific measure and metric is important, but that is not within the scope of this project.

Let L be a distributive lattice of n-dimensional vectors and B be a predicate over L. Then, we define

$$forbidden(G, i, B) \coloneqq \forall H \in L : G \leq H : G[i] = H[i] \implies \neg B(H)$$

. This states that component i is forbidden in vector G in L if, no matter how G is progressed, fixing i keeps the predicate false. We then say that B is lattice-linear if for any G not satisfying B, G has a forbidden component. For the purposes of minimality, we also need to know how much to progress a forbidden component so as to make it no longer forbidden while not jumping over the optimal solution. Let $\delta(G, i, B)$ be the greatest value such that for all $H \in L$ such that $H \geq G$, $H[i] < \delta(G, i, B) \implies \neg B(H)$ whenever $forbidden(G, i, B)$. That is, it is the least value we need to bring component i up to in order to make it no longer forbidden. Advancing a vector G with a forbidden component i is done by setting the ith component to $\delta(G, i, B)$.

Finding the least vector satisfying B is solvable in parallel time proportional to the height of the lattice [3]. In general, this problem finds a simple parallel (and distributed) algorithm that guarantees optimality. We later prove its correctness mechanically along with newer results.

3 A Method for Approximating LLP Problems

Suppose we have an LLP B over a lattice of n-dimensional vectors L with a quasi-metric d defined over it. Let L' be a lattice of m-dimensional vectors with (α, γ) a GI between L and L'. We say that $f : L \to L$, $f' : L' \to L'$ are ϵ-complete (e.c.) w.r.t the GI if $d(f(G), (\gamma \circ f' \circ \alpha)(G)) \leq \epsilon$ for all G in L. This notion of *partial completeness* has been studied in abstract interpretation and utilized in static analysis [1, 4, 2].

The basic idea is as follows: with L' being a lattice with smaller height than L, progress through the smaller lattice by advancing its forbidden components. Every time a component is advanced, we keep track of the concretized vector in L ($\gamma(G)$). Subsequently, the next time a component is advanced, we check whether we have stepped over a feasible solution in L by checking $B(f(G))$ where G is the last position in L stored. It is not immediately obvious that if $f(G)$ is feasible, then it must be optimal (proven in ACL2, described later), so

$$d(f(G), (\gamma \circ f' \circ \alpha)(G)) = d(OPT, (\gamma \circ f' \circ \alpha)(G)) \leq \epsilon$$

where OPT is the optimal solution in L. This gives $(\gamma \circ f' \circ \alpha)(G)$ as a feasible ϵ-approximation. We can abuse the GI property to make this more efficient (also proven in ACL2) since repeated application of $\gamma \circ f' \circ \alpha$ gives

$$\gamma \circ f' \circ \alpha \circ \gamma \circ f' \circ \alpha = \gamma \circ f' \circ \alpha$$

Many applications of this will cancel out many instances of $\alpha \circ \gamma$ by the GI property, thereby saving many unnecessary computations.

Counter-intuitively, the ϵ distance does *not* accumulate because we "fix" a comparison point in L at each step. This is the crux and novelty of the project, proven in ACL2.

4 ACL2 Representation and Results

In this section, we will use "components" and "states" interchangeably.

We represent LLPs abstractly so as to apply these proofs to any concrete or realized instance by utilizing the "encapsulate" primitive from ACL2. This approach provides the definitional properties of LLPs without exposing the definition of the abstract functions and predicates. The encapsulate includes properties for the necessary theory and realizable functions and constants.

In LLPs, lattices are typically over vectors of reals ordered component-wise, however for the sake of simplicity in this project, we consider vectors of naturals ordered similarly. This comes in the form of the definitions `vecp` and `lattice-leq`, both of which have a small number of associated theorems.

From these definitions and the basic properties provided by encapsulate, we prove a number of important lemmas. The most important theorems proven are:

1. `bounded-by-opt`: This proves that advancing a vector below OPT keeps it below OPT.

2. `gi-makes-stepping-efficient`: This proves that GI's enable an optimization on the approximation algorithm.

3. `constant-approx-dist`: This proves that the algorithm gives a constant approximation factor (namely, ϵ) given that it only takes a finite number of steps to reach OPT in the standard LLP algorithm.

4. `lattice-leq-component` & `advance-is-higher-lattice`: These prove that advancing a vector with the aforementioned advance rule moves it strictly up the lattice. This directly implies (with `bounded-by-opt`) that it only takes a finite number of steps to reach OPT in the standard LLP algorithm (thus supporting `constant-approx-dist`).

The ACL2 code can be found here.

5 Conclusion

In this project, presented a novel technique for representing approximation algorithms with LLPs, modeled this technique in ACL2, and utilized the ACL2 theorem prover to prove an approximation bound on this technique. Using an autmated theorem prover like ACL2 not only gave strong assurance of the technique's validity, but also offered valuable insights into automated deduction and the technique's underlying recursive structure. This project can be further extended to apply these ACL2 theorems to verify a concrete application of this technique.

References

[1] Marco Campion, Mila Dalla Preda, and Roberto Giacobazzi. Partial (in)completeness in abstract interpretation: limiting the imprecision in program analysis. *Proc. ACM Program. Lang.*, 6(POPL), January 2022.

[2] Marco Campion, Caterina Urban, Mila Dalla Preda, and Roberto Giacobazzi. A formal framework to measure the incompleteness of abstract interpretations. In *Static Analysis: 30th International Symposium, SAS 2023, Cascais, Portugal, October 22–24, 2023, Proceedings*, page 114–138, Berlin, Heidelberg, 2023. Springer-Verlag.

[3] Vijay K. Garg. Predicate detection to solve combinatorial optimization problems. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, page 235–245, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Dennis Liew, Tiago Cogumbreiro, and Julien Lange. Sound and partially-complete static analysis of data-races in gpu programs. *Proc. ACM Program. Lang.*, 8(OOPSLA2), October 2024.