Charles, P. J., Howe, J. M. & King, A. (2009). Integer polyhedra for program analysis. Lecture Notes in Computer Science, 5564, 85 - 99. doi: 10.1007/978-3-642-02158-9_9 <http://dx.doi.org/10.1007/978-3-642-02158-9_9>

Integer Polyhedra for Program Analysis

Philip J. Charles[1], Jacob M. Howe[2], and Andy King[1,3]

[1] University of Kent, Canterbury, CT2 7NF, UK
[2] City University London, EC1V 0HB, UK
[3] Portcullis Computer Security Limited, Pinner, HA5 2EX, UK

Abstract. Polyhedra are widely used in model checking and abstract interpretation. Polyhedral analysis is effective when the relationships between variables are linear, but suffers from imprecision when it is necessary to take into account the integrality of the represented space. Imprecision also arises when non-linear constraints occur. Moreover, in terms of tractability, even a space defined by linear constraints can become unmanageable owing to the excessive number of inequalities. Thus it is useful to identify those inequalities whose omission has least impact on the represented space. This paper shows how these issues can be addressed in a novel way by growing the integer hull of the space and approximating the number of integral points within a bounded polyhedron.

1 Introduction

The aim of this work is to take algorithms from computation geometry and linear programming and apply them to solve problems arising in program analysis using polyhedra. In abstract interpretation convex polyhedra have long been used to abstract the sets of values that variables may take [6]. This has proven to be attractive in program analysis because, as well as prescribing range constraints on variables, polyhedra can also describe interactions between variables.

Polyhedral analyses sometimes need to consider integrality [16], for instance, to derive invariants between integral objects such as loop counters and pointer offsets [20]. In such analyses variables are discrete, whereas polyhedra are defined over real or rational numbers. Further, polyhedra cannot express non-linear relationships; in this case, the non-linearity is either projected out or approximated in an ad hoc way. These drawbacks impede the accurate analysis of programs. In terms of tractability, polyhedra can be too expressive in some situations; an analysis can become overwhelmed by large systems of (non-redundant) inequalities. This paper presents a synthesis of solutions to the three problems introduced above: integrality, non-linearity and tractability.

The target of this work is abstract interpretation based analyses, such as those performed by [5]. In such an analysis a fixpoint in the meet semi-lattice of polyhedra over the variables of interest is calculated, where this fixpoint describes the values and relationships between program variables. The smaller the fixpoint (when the polyhedra is interpreted as a set of points), the more information it contains. In particular, if the polyhedra describes the values of integers (and no floating point variables), then tightening to the integer hull provides a systematic way of strengthening an analysis.

The starting point of the work is that a polyhedron can be grown to describe the integer solutions of a system of constraints. The process is incremental in nature. First an integer solution to the system of constraints is found. Then a second distinct integer solution is found whose distance is maximal from the first and the convex hull of this point and the previous space is taken. Iterating this mechanism, one of the inequalities that bounds the current space is chosen and a solution is found at maximal normal distance from the inequality. This process is repeated until all inequalities have been considered, giving the integer hull [10]. Computing the integer hull for arbitrary systems of even linear inequalities is NP-hard, limiting the size of problems likely to be solvable and motivating approximation techniques. Observe that the technique above gives a series of integer polyhedra approximating the solution from below, converging on the precise solution. It will also be seen that an approximation from above can be extracted from the algorithm. It is important to note that the input set of constraints is not necessarily linear, thus this approach addresses two of the three problems: integrality and non-linearity.

A potential drawback of the above technique is that the resulting integer polyhedron may involve an unmanageably large number of inequalities. This motivates a systematic technique for relaxing a polyhedron by reducing the number of inequalities. This is achieved by calculating a Monte Carlo approximation of the number of integer points that a constraint bars from a polyhedron. The least contributing constraints are relaxed. This approach provides a way of curbing the growth of inequalities and computing an integer approximation whose number of defining constraints does not exceed some bound, addressing the problem of tractability.

This paper brings together a number of threads in program analysis and computational geometry and the contributions of the paper are summarised:

– The algorithm of [10] that grows the integer hull and allows anytime approximation from below and above is presented and elucidated.
– This algorithm can be adapted to calculate the integer hull of a projection of the input constraint system onto a subset of its variables. When running to completion this method can be used for over-approximating the integer solutions of a set of non-linear constraints, an approximation problem which thus far has not been satisfactorily addressed.
– A method to determine which constraints contribute little to the enclosed space, hence are candidate for relaxation, is presented. This is parameterised by the method used to determine this contribution and a Monte Carlo approximation technique is discussed in detail.
– The integer hull algorithm and one approach to relaxation have been implemented and the results of an empirical evaluation have been presented. The results are promising for the use of the algorithms in program analysis.

2 Growing Integer Hulls

This section details the calculation of the convex hull of the integer solutions – the so-called integer hull [18] – of a system of constraints C defined over totally ordered variables $x_1, \ldots, x_n$. The integer hull is approximated from below, growing it from a point by giving an inequality c to an oracle that will return a point p satisfying C, but not c; inequality c bounds the current hull, W. The convex hull of p and W is then calculated. This approach was first seen in [10] and is also remarked upon in [4]. Here, the algorithm is detailed with particular attention given to the maintenance of the inequalities describing the hull. Further attention is paid to the novel use of this algorithm in the context of program analysis, especially the way in which it can deal with non-linear constraints.

2.1 An integer hull algorithm

The following three procedures detail the integer hull algorithm implemented in Section 4. The main loop of the algorithm is contained in the second procedure, worklisthull. The first procedure, integerhull below, sets up the problem:

```
 1: procedure integerhull($C$)
 2: $\boldsymbol{p} := \mathsf{maximise}(C, x_1)$;
 3: if $\boldsymbol{p} = null$ then return $null$; end if
 4: $Ineqs := \{x_i \le p_i, -x_i \le -p_i \mid 1 \le i \le n\}$;
 5: $Ps := \{\boldsymbol{p}\}$;
 6: $Cons := \mathsf{sort}(Ineqs)$;
 7: $lastrank := 0$;
 8: $rank := \mathsf{rank}(Ineqs)$;
 9: while $Cons \neq []$ do
10:    $\boldsymbol{p'} := null$;
11:    while $Cons \neq [] \wedge \boldsymbol{p'} = null$ do
12:       $Cons \equiv f :: Rest,\ f \equiv \boldsymbol{c} \cdot \boldsymbol{x} \le d$;
13:       $\boldsymbol{p'} := \mathsf{maximise}(C \wedge \neg f, \boldsymbol{c} \cdot \boldsymbol{x})$;
14:       $Cons := Rest$;
15:    end while
16:    if $\boldsymbol{p'} = null$ then
17:       return $Ineqs$;
18:    else
19:       $lastrank := rank$;
20:       $Ineqs' := \mathsf{convexhull}(\boldsymbol{p'}, Ineqs)$;
21:       $rank := \mathsf{rank}(Ineqs')$;
22:       if $rank > lastrank$ then
23:          $Ineqs := Ineqs'$;
24:          $Cons := \mathsf{sort}(Ineqs)$;
25:          $Ps := Ps \cup \{\boldsymbol{p'}\}$;
26:       else
27:          return $\mathsf{worklisthull}(Ps, Ineqs, C)$;
28:       end if
29:    end if
30: end while
```

The purpose of this procedure is to calculate a first approximation (importantly, a simplex) of the integer hull that reaches the dimension of the final solution. Note that the integer hull might well be a hyperplane of lower dimension than n, the number of varibles. On line 2, a first integer point in the hull is calculated. This uses the auxiliary function maximise(C, c) that takes a system of constraints C and a cost function c and returns an integer solution to C that maximises c. If no such point exists it returns $null$. The choice of the first cost function is arbitrary. $Ineqs$ is a set of linear inequalities describing the current approximation and Ps is the set of points so far calculated. $Ineqs$ is then sorted by the number of points in Ps that lie on the boundary of an inequality. This ensures that the next discovered point will raise the dimension, if full dimensionality is not yet reached. The dimension of a set of inequalities is determined by function rank. The next point is determined on line 13 and (the topological closure of) the convex hull of this point with the previous hull is calculated on line 20 using an appropriate method. This process is repeated through lines 9 to 30 until either the integer hull is calculated or full dimensionality is reached.

Before passing on to worklisthull, it is worth noting that replacing lines 18-28 (and the rank variables and calculations) of integerhull with

```
$Ineqs :=$ convexhull$(\boldsymbol{p'}, Ineqs)$;
$Cons :=$ sort$(Ineqs)$;
```

will give a complete algorithm not using the following two procedures. This will be referred to later as integerhull′. This is essentially what is given in [10] and the additional procedures detail the use of simplicial faces to control the generation of new inequalities (which in integerhull′ result from the call to convexhull).

Procedure worklisthull is passed a set of points, a set of inequalities describing their integer hull and the input constraints. This procedure works on a simplicial input and the final point calculated by the integerhull is not included in the set of points; worklisthull provides the main loop and is given below:

```
 1: procedure worklisthull$(Ps, Ineqs, C)$
 2: $Hull =$ dimred$(Ps, Ineqs)$;
 3: $Worklist =$ faces$(Ps, Ineqs)$;
 4: while $Worklist \neq \phi$ do
 5:     $f(Vs, ineq) \in Worklist$;
 6:     $\boldsymbol{p} :=$ maximise$(C \wedge \neg ineq, \boldsymbol{c} \cdot \boldsymbol{x})$ where $ineq \equiv (\boldsymbol{c} \cdot \boldsymbol{x} \leq d)$;
 7:     if $\boldsymbol{p} = null$ then
 8:         $Hull := Hull \cup \{ineq\}$;
 9:         $Worklist := Worklist \setminus \{f(Vs, ineq)\}$
10:     else
11:         $Ps := Ps \cup \{\boldsymbol{p}\}$;
12:         $Worklist :=$ hull$(\boldsymbol{p}, Worklist, Ps)$;
13:     end if
14: end while
15: return $Hull$;
```

In worklisthull a worklist of consists of faces, where a face $f(Vs, ineq)$ is a set of integer points Vs, with $|Vs|$ equal to the dimension of the integer hull, and inequality $ineq$ with each point in Vs lying on the boundary of $ineq$. Each face is a simplex of dimension $|Vs| - 1$. $Hull$ represents the inequalities in the integer hull. It is initialised with any dimension reducing inequalities, determined by auxiliary dimred – every point in Ps will be on the boundary of such an inequality. The auxiliary faces sets up the initial worklist. Whilst there are faces in the worklist a face $f(Vs, ineq)$ is selected and the oracle is asked for a point $\boldsymbol{p}$ satisfying C, but not $ineq$, line 6. If there is no such point, then $ineq$ is added to $Hull$, line 7. If there is, line 10, the procedure hull determines a new worklist, replacing any face not satisfied by $\boldsymbol{p}$ with a set of new faces whose determining points will include the new point. Note that the call to hull will remove the current face from the worklist.

Procedure hull, below, takes the place of a convex hull calculation in worklisthull:

```
 1: procedure hull$(\boldsymbol{p}, Worklist, Ps)$
```

```
2: $NewWorklist = \phi$;
3: for all $f(Vs, ineq) \in Worklist$ do
4:   if $\boldsymbol{p} \models ineq$ then
5:     $NewWorklist := NewWorklist \cup \{f(Vs, ineq)\}$;
6:   else
7:     for all $\boldsymbol{v} \in Vs$ do
8:       $Vs' := (Vs \setminus \{\boldsymbol{v}\}) \cup \{\boldsymbol{p}\}$;
9:       $ineq' := \mathsf{ineq}(Vs', \boldsymbol{v})$;
10:      if $\forall \boldsymbol{q} \in Ps.\boldsymbol{q} \models ineq'$ then
11:        $NewWorklist := NewWorklist \cup \{f(Vs', ineq')\}$;
12:      end if
13:    end for
14:  end if
15: end for
16: return $NewWorklist$;
```

The procedure will retain any face satisfied by the new point $\boldsymbol{p}$, line 4 ($\models$ denotes the satisfaction relation). An unsatisfied face forms the base of a simplicial cone whose pinacle is $\boldsymbol{p}$. The faces of this cone are the simplicies obtained by replacing one of the points defining the base by $\boldsymbol{p}$, line 8. The inequality for this new face can be calculated (see below), from these points, plus the discarded point, line 9. Finally, the worklist need only retain faces that are currently satisfied by all discovered points, others are discarded, line 10.

The plane through a set of d independent points, $\boldsymbol{p_1}, ..., \boldsymbol{p_d}$, can be calculated in constant time for fixed d by solving the parametric description of the plane using Gaussian elimination. That is, $plane = \boldsymbol{p_1} + \sum_{i=2}^{d} \lambda_i.vec(\boldsymbol{p_1}, \boldsymbol{p_i})$ where $vec(\boldsymbol{p_i}, \boldsymbol{p_j})$ is the vector from point $\boldsymbol{p_i}$ to point $\boldsymbol{p_j}$. Set up a matrix where the first $d-1$ columns are given by $vec(\boldsymbol{p_1}, \boldsymbol{p_{i+1}})$, the next d columns are the unit vectors for each dimension and the final column is the entries of $\boldsymbol{p_1}$. Use Gaussian elimination to set the first $d-1$ entries of the final row to 0 and read off the equation of the plane from the entries in the remaining columns of this row. The discarded point can then be used to determine the desired inequality.

Anytime Approximations During execution of worklisthull at any point the accumulated inequalities of $Hull$ and $Worklist$ determine an integer polyhedron that is an underapproximation of the integer hull, allowing anytime approximation from below. Further, note that at any point $Hull$ is a potentially unbounded polyhedron (but not necessarily an integer polyhedron) that is an over-approximation of the integer hull, allowing anytime approximation from above. Algorithms with anytime approximation are paricularly attractive for program analysis when attempting to bound the time the analysis takes. Both under and over-approximations are useful, depending the whether analysis is for properties that definitely hold, or potentially hold.

Example 1. Consider the following linear constraints, $C = \{-11x+y \leq -8, 2x+8y \leq 71, 8x+4y \leq 67, 19x+2y \leq 116, -4x-11y \leq -35\}$. This is represented by the dotted lines in Fig. 1. The initial call to maximise gives p_1, subsequent calls from integerhull give the points p_2 and p_3 and the simplicial under-approximation of the integer hull given by the continuous lines in Fig. 1 a).

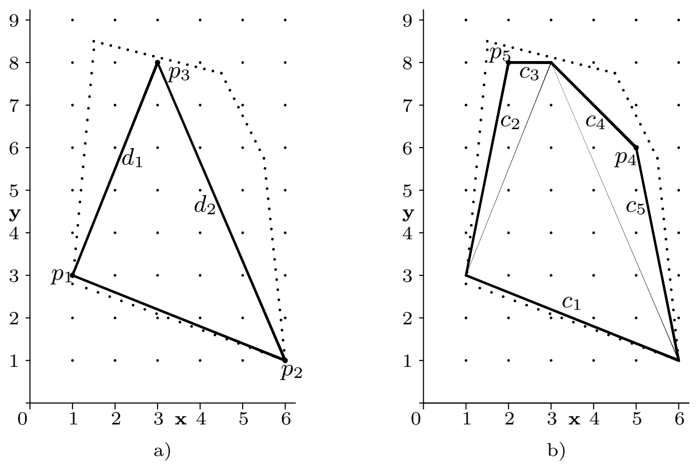

Fig. 1. Integer hull of a set of linear constraints

In worklisthull, the face with inequality $c_1 \equiv -2x - 5y \leq -17$ will be first selected. The call maximise(C, c_1) will return *null* and c_1 will be added to *Hull*. Next consider $face(\{(6,1),(3,8)\}, d_2)$, giving the call to maximise(C, d_2) that will return $(5,6)$. In hull, the face with inequality d_1 is satisfied by the new point and will remain in the worklist. The new faces are $face(\{(5,6),(3,8)\}, c_4)$ and $face(\{(6,1),(5,6)\}, c_5)$. To determine c_4 consider the matrix on the left, which with one elimination step gives that on the right:

$$\begin{bmatrix} -2 & 1 & 0 & 5 \\ 2 & 0 & 1 & 6 \end{bmatrix} \qquad \begin{bmatrix} -2 & 1 & 0 & 5 \\ 0 & 1 & 1 & 11 \end{bmatrix}$$

This allows the result to be read off: $c_4 \equiv x + y \leq 11$ (the point (6,1) has been used to determine the inequality). Similarly, $c_5 \equiv 5x + y \leq 31$. Further iterations give $c_2 \equiv -5x + y \leq -2$ and $c_3 \equiv y \leq 8$. Since there are no further external points satisfying C these will be added to *Hull* which will finally be returned.

Note that a tightening has been achieved. The input C projected onto x gives range [0.984,6] whereas the integer hull gives [1,6], and for y [1,8.5] becomes [1,8].

2.2 Working in a projected space and non-linear constraints

This section builds upon two observations on the algorithm presented in the previous section to highlight its suitability for use in program analysis. The first observation is that the algorithm is easily adapted to compute the integer hull of a k-dimensional projection of constraints C, that is, the smallest polyhedron that contains those points $\langle v_1, \ldots, v_k \rangle$ for which C possesses a corresponding integer solution $\langle v_1, \ldots, v_n \rangle$. Restriction to a subset of variables of interest is an operation commonly required for program analysis. The adaptation is simply achieved by restricting the points $\boldsymbol{p}$, determined by calls to maximise, to the variables of interest. The second observation is that C may contain non-linear constraints, as long as the point oracle can deal with these. Both of these observations are illustrated with a well-known problem from program analysis.

Although the seminal paper on polyhedral analysis [6] identified the problem of approximating non-linear constraints, a widely accepted solution to the problem has not been found. Consider the example of [6, Sect. 4.2.1] to illustrate how to compute a polyhedral approximation of a non-linear assignment. Specifically, suppose the constraint $S = \{-x + y \leq 1, -y \leq -1, -x - y \leq -5\}$, holds when the non-linear assignment $y := xy$ is encountered. The problem is how to systematically compute a polyhedral approximation of the ensuing non-linear space. This problem can be addressed by augmenting S with the constraint $y' = xy$, thereby raising the dimension, then symbolically projecting out y, and replacing y' with y. This gives the shaded space in Fig. 2(a) defined by $\{-x \leq -2, y \leq x + x^2, -y \leq -x, -y \leq x^2 - 5x, y \leq 32767\}$. This approach presupposes that a symbolic projection algorithm is known for the system of augmented constraints, which of course, is not guaranteed in general [7]. Note that the $y \leq 32767$ constraint is imposed by an underlying 16-bit representation where variables range over $[-32768, 32767]$; other machine representations would likewise ensure that integer variables can only lie within a finite range. Note too that the manually derived non-linear constraint suggested in [6, Sect. 4.2.1] omits the inequality $-x \leq -2$ that is necessary to exclude the origin. This inequality follows from a linear combination of $-x+y \leq 1$ and $-x-y \leq -5$, and illustrates the subtlety of manually abstracting non-linear constraints.

Now consider a run of the algorithm where C is instantiated to $S \wedge y' = xy$ and the variables are totally ordered as in the sequence x, y', y. Putting $k = 2$ then eliminates the variable y so that the algorithm computes the integer hull of the projection of C onto the x, y' plane. An initial solution $\boldsymbol{u} = \langle 32767, 32767, 1\rangle$ is computed at line 2 of integerhull. The projection of $\boldsymbol{u}$ onto the x, y' plane is merely $\boldsymbol{u}' = \langle 32767, 32767\rangle$ which can be represented as a system of inequalities $\{x \leq 32767, -x \leq -32767, y' \leq 32767, -y' \leq -32767\}$, which defines the polyhedra P at line 4 and can be seen in Fig. 2(b).

On the first iteration f is chosen to be $-x \leq -32767$, then the cost function at line 8 is $-x$. The net effect is to find the solution $\boldsymbol{v} = \langle 2, 6, 3\rangle$ that minimises the x coordinate whilst satisfying $C \wedge \neg f \equiv C \wedge x < 32767$. Projecting $\boldsymbol{v}$ onto the x, y' plane yields $\boldsymbol{v}' = \langle 2, 6\rangle$. Extending the polyhedra P in Fig. 2(a) with this point by computing the convex hull at line 9 gives the line segment $\{-x \leq -2, x \leq 32767, 32765y' = 32761x + 131068\}$ depicted as P in Fig. 2(b).

On the second iteration, f is chosen to be $32765y' \leq 32761x + 131068$, leading to the triangle depicted in Fig. 2(c). At this stage P has reached full dimensionality and worklisthull will be called. Here, a call to maximise with *ineq* as $-32765y' \leq -32761x - 131068$ will lead to the polyhedra in Fig. 2(d). After this, further iterations will fail to find further points and P will be returned. Notice that P is formulated in terms of y' which represents the value of y after the assignment. The state of the x, y variables after the assignment is obtained by merely replacing y' with y. Again notice that symbolic computation of the projection has been replaced by an integer hull calculation.

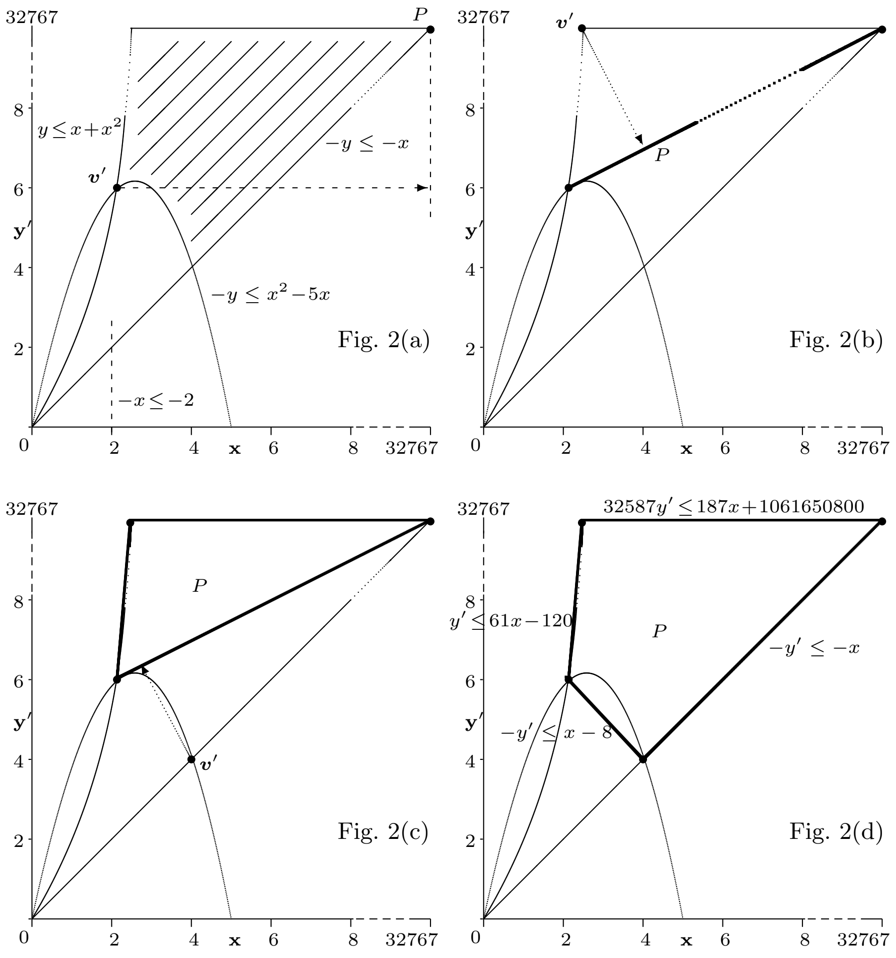

Fig. 2. Polyhedral approximation of a feasible region defined by non-linear constraints

3 Curbing Growth of the Integer Hull

Earlier it was noted that the integer hull algorithm presented allows for anytime approximation from above and below. This is advantageous when the algorithm fails to perform quickly enough, however an approach that allows selected inequalities to be dropped is desirable for a different kind of problem. As the polyhedron representing a solution space grows, the number of inequalities may also grow. This growth can be curbed by either relaxing inequalities in the final polyhedron or dropping them on-the-fly whilst the integer hull algorithm proceeds. These two approaches will henceforth be referred to as off-line and on-line integer hull relaxation. Here, the focus is on off-line relaxation. These techniques can be used to limit the growth of a system of inequalities, ameliorating any ensuing tractability issues (with possible precision cost).

Suppose that $S = \{0 \leq x, 0 \leq y, y \leq -x+6\}$, describes the relative values of variables x and y when the non-linear assignment $y := xy$ is reached. Applying the integer hull algorithm to the system of non-linear constraints $S \wedge y' = xy$ with the variable ordering x, y', y and $k = 2$ gives the projected integer hull, H, enclosing only those integer points that satisfy both the linear inequality $-y' \leq 0$ and the non-linear inequality $y' \leq -x^2 + 6x$. H is defined by 7 inequalities c_1: $-y' \leq 0$, c_2: $y' \leq 5x$, c_3: $y' \leq 3x+2$, c_4: $y' \leq x+6$, c_5: $y' \leq -x+12$, c_6: $-3x+20$, c_7: $y' \leq -5x+30$. Rank these inequalities according to some measure of their suitability for relaxation, and discard as appropriate. For example, if the ranking was $c_3, c_6, c_4, c_5, c_2, c_7, c_1$ then relaxing the highest two ranked would give a small increase the volume, but this slightly larger polytope contains no additional integer points. Reranking, this process could be continued to a final result given by c_2, c_7, c_1.

This approach is parameterised by the function ranking the inequalities. The method chosen here for ranking is to calculate a Monte Carlo approximation, [15], of the volume of $H \wedge \neg c_i$ that represents the increase in volume resulting from the relaxation. A bounding box is constructed and sampled until the sampling error ($\sigma/\sqrt{n}$, where $\sigma^2 = (r^2+n^2)/(r+n)^2$, r is the number of samples in the region and n the number of samples not in the region) is beneath a given value. The proportion of sample within the polytope multiplied by the volume of the bounding box gives an approximation to the volume, as required.

Alternative rankings are possible: the volume of a polytope (with rational vertices) can be computed in polynomial time [2] and, rather surprisingly, so can the number of integer points in such a polytope [3, 8, 22], which is exactly what is required when describing integral properties. However, despite their complexity these remain difficult problems, particularly in high dimension and sampling based methods seem more suitable to guiding the quick relaxation of constraints.

A natural generalisation of off-line relaxation is on-line relaxation which discards inequalities as soon as their number exceeds some pre-defined threshold in the main loop of the integer hull algorithm. This approach is problematic for the integerhull algorithm given earlier as relaxation will lead to faces whose vertices are not known and the hull method will not work. However, the integerhull′ method with its reliance on a more general convex hull algorithm can incorporate this – simply follow the call to convexhull with as many relaxation steps as required. Anecdotal evidence suggests that this might useful, particularly in discarding inequalities with large coefficients that are both problematic for performance and less likely to be useful for program analysis.

4 Experimental Evaluation

The algorithm described in Section 2 and the off-line approximation technique described in Section 3 were implemented and tested.

As mentioned in the introduction, the target of this work is abstract interpretation based analyses, where a fixpoint describing the values and relationships between program variables is calculated. This fixpoint is a point in the meet semi-lattice of polyhedra over the variables. Fixpoints arise because of loops: the values of the variables after an iteration of a loop serve as the values that are input to the next iteration. The semantic equations that express the values that variables can assume are thus recursive. A fixpoint of these equations can be interpreted as expressing invariants that hold over all iterations of a loop. The fixpoint may not necessarily be the unique least fixpoint; the requirement for correctness is merely that if the fixpoint summarises values that hold in one iteration, then it also summarises values that hold in the next. By tightening to integer polyhedra at certain analysis points the analysis is strengthened. The tightening to the integer hull could be applied with differing levels of granularity: after each domain operation, at the end of the fixpoint calculation, or after the analysis of each loop structure. The benchmarks best represent the last of these.

The benchmarks come from the Stanford Invariant Generator (Sting) and FAST [17, 1]. The Sting analyser discovers linear invariants of transition systems that represent iterations of loops where all the variables are integer. The benchmarks are invariants generated by Sting and are representive of the program analysis problems that this work is aimed at. The implementation is in Java and the experiments were run on a single core of a MacBook with a 2.4GHz Intel Core 2 Duo processor and 4GB of memory.

Calculating Integer Hulls The algorithm is coded in Java, with the oracle provided by the CBC MILP solver [13]. CBC is coded in C++ and called via the Java Native Interface. The integer hull is only defined for bounded problems and as noted earlier inequalities need to be augmented with variable bounds. In these experiments the problem constraints were augmented with variables bounds of [0,64]. The results can be seen in Fig. 3: for each named benchmark, Var gives the number of variables in the benchmarks, Ineqs the number of input constraints (including the variable bounds), Time gives the execution time in seconds for calculating the integer hull, Opts the number of calls to the ILP solver and Sol Size the number of inequalities in the integer hull.

The *barvinok* package for integer point counting [22] and the *Polylib* package for manipulating integer polyhedra [23] have been used to check the integer hull calculations given in this paper. *barvinok* has been run on the input constraints and the calculated integer hull to check that the number of lattice points are indentical for both and *Polylib* has been used to convert the calculated constraints to a vertices and rays representation, the test being that the vertices are all at integer points.

Off-line Approximation Fig. 4 tabulates results from applying the Monte Carlo approximation of Section 3 to the results of the integer hull calculations. Benchmarks with no bounded relaxation have been omitted. The first set of results gives data on ranking and relaxing one constraint: T1 is the time taken in seconds, Best is the number of sample points needed to calculate the volume associated with the constraint dropped, Total is the total number of sample points and Max is the largest sample size needed to approximate a volume arising

Problem	Var	Ineqs	Time	Opts	Sol Size
barber.inv	8	29	3.417	207	15
berkeley-nat.inv	4	13	0.075	29	9
berkeley.inv	4	11	0.069	28	9
cars.inv	5	19	0.15	61	13
efm.inv	6	22	0.111	39	12
efm1.inv	6	21	0.086	19	9
heap.inv	5	16	0.08	26	10
lifo-nat.inv	7	24	0.297	87	13
lifo.inv	4	14	0.047	15	6
robot.inv	3	10	0.031	10	5
scheduler-2p.invl1	7	27	0.15	46	15
scheduler-2p.invl2	7	27	0.279	65	17
scheduler-3p.invl1	10	40	10.881	273	42
scheduler-3p.invl2	10	46	194.022	1037	125
scheduler-3p.invl3	10	38	23.034	388	26
see-saw.inv	2	6	0.022	10	5
swim-pool-1.inv	9	32	0.255	62	16
swim-pool.inv	9	31	0.248	57	15
train-beacon.invlate1	3	11	0.026	12	7
train-beacon.invonbrake	3	10	0.037	14	6
train-beacon.invontime	3	12	0.027	14	8
train-beacon.invstopped	3	11	0.026	12	7
train-rm03.inv	6	20	0.12	38	12

Fig. 3. Benchmarking of the integer hull algorithm

from a single constraint. The second set of results details relaxing as many constraints as possible, recalculating the ranking at each step: TM is the time taken in seconds, Cons is the number of constraints in the input, Size is the final number of constraints and Sam is total number of samples taken in this process.

Discussion The results are promising. The implementation (not tuned to the problems) returns the integer hull for all benchmarks up to 10 dimensions in an acceptable time. These are the first experiments of this kind performed on program analysis benchmarks (indeed, the authors know of no integer hull benchmarking work at all). However, at 10 dimensions and beyond performance degenerates (the implementation was unable to solve nine further suitable benchmarks over more than 10 variables in a reasonable time). This is in part because some calls to the ILP oracle become slow, in part owing to the amount of factoring performed and in part owing to the growth in the number of simplicies to be handled. The largest benchmarks in the suite have 15 variables; this is real loop data and being able to handle between 10 to 20 variable problems would allow the analysis of many programs. The authors believe that further work on the implementation will yield improvements in scalability. Size of the constraint coefficients is also a problem as the bounding box increases in size – most benchmarks run equally well with larger bounding boxes, but not all.

Problem	T1	Best	Total	Max	Volume	TM	Cons	Size	Sam
berkeley-nat.inv	0.061	100	401	101	260.0	0.089	9	8	405
berkeley.inv	0.049	107	518	108	5.981	0.084	9	8	514
cars.inv	0.099	134	1403	190	0.0	0.455	13	8	4596
efm.inv	0.114	200	635	200	0.0	0.255	12	10	973
efm1.inv	0.080	200	200	200	0.0	0.127	9	8	200
heap.inv	0.077	200	594	200	0.0	0.179	10	8	984
lifo-nat.inv	0.132	184	560	184	5.565	0.235	13	12	525
robot.inv	0.022	148	148	148	11000.372	0.032	5	4	149
scheduler-2p.invl1	0.155	200	1368	200	0.0	0.709	15	10	3673
scheduler-2p.invl2	0.198	200	1586	200	0.0	1.076	17	10	6776
scheduler-3p.invl1	1.025	200	6170	200	0.0	16.282	42	14	92934
scheduler-3p.invl2	47.755	200	22848	200	0.0	1556.5	125	14	1326519
scheduler-3p.invl3	0.499	200	2572	200	0.0	4.312	26	14	17962
see-saw.inv	0.012	107	452	142	2.505	0.027	5	3	639
swim-pool-1.inv	0.223	200	800	200	0.0	0.745	16	13	1598
swim-pool.inv	0.208	200	400	200	0.0	0.521	15	13	600
train-beacon.invlate1	0.026	109	331	112	1.431	0.045	7	6	349
train-beacon.invonbrake	0.030	104	206	104	3.308	0.038	6	5	214
train-beacon.invontime	0.034	134	716	134	30.09	0.100	8	5	1205
train-beacon.invstopped	0.031	114	346	120	1.263	0.064	7	5	441
train-rm03.inv	0.095	179	710	182	4.148E9	0.305	12	9	1219

Fig. 4. Off-line relaxation of constraints

With two or three exceptions approximation results give the desired behaviour – sensible constraints to relax can be quickly identified. The slower benchmarks indicate a need to augment ranking with a timeout. A further issue is that when relaxing more than one constraint, dropping one or other equally ranked constraint can change the total number of constraints relaxed.

5 Related Work

The algorithm presented in Section 2 was first outlined by Hartmann in [4, 10] and is also mentioned in [9]. However, its relationship with projection, non-linearity and program analysis have not previously been commented on. An alternative algorithm has been proposed by Meister based on the concept of periodic polyhedra [14]. Eisenbrand's work [9] also deals with approximating the integer hull from above, a counterpart to the work here that is also useful for program analysis. In terms of implementation, the *barvinok* package for integer point counting [22] includes an integer hull algorithm also based on [10]; initial experimentation suggests that this does not scale as well as the implementation described here. The *iB4e* system [11] implements the beneath/beyond convex hull algorithm over reals and uses similar concepts; it is also presented with an oracle for solving LP problems, this could be an ILP solver.

The work on polynomial algorithms for lattice point counting from Barvinok and others [2, 3, 8, 12, 22] gives precisely what is required for assessing the importance of a candidate constraint for relaxation. This work has been extended to count points in the projection of a constrained space [21]. The results of these systems are impressive, but are still slower than sampling based techniques for estimating the number of integer points in a polytope as used in the approximation thread of this work. Recent work on loop nest analysis [19, 22] utilises the algorithmic results on point counting. However, abstract interpretation based analysis requires constraints between variables, not lattice point counts.

6 Conclusion

This paper has presented work on the application of algorithms from computational geometry and linear programming to data arising in program analysis. An existing algorithm has been detailed and elucidated, and features that make its adaptation to problems of non-linearity and integrality in program analysis easy and natural have been identified. It has also been implemented and empirically evaluated. The results of this evaluation underline that this novel approach to dealing with integer variables and non-linear constraints in program analysis is promising. The approach is coupled with a method for approximating the increase in volume associated with relaxing a constraint from a system in order to control the size of that system. This too has been implemented and again the results suggests that the methods will be of importance in program analysis.

The implementation described in this paper represents the state-of-the-art for integer hull calculation. The results are promising, but also invite further work on increasing the speed of the implementation and the size of problem that can be dealt with. Future work will focus on improvements to the current implementation, whilst also investigating alternative approaches avoiding large numbers of calls to an ILP solver. The current implementation is not tuned to program analysis benchmarks and a further line of work is to investigate whether the structure of these lead to practical or theoretical improvements.

Acknowledgements The authors thank Tim Hopkins, Mathias Köppe and Axel Simon for useful discussions. They would further like to thank the anonymous referees for their insightful comments and pointers to related work. The work was supported by EPSRC projects EP/E033105/1 and EP/E034519/1. Andy King's secondment to Portcullis is supported by a Royal Society Industrial Fellowship. Jacob Howe would like to thank the University of St Andrews for supporting him as a visiting scholar.

References

1. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast Acceleration of Symbolic Transition Systems. In *Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 118–121. Springer-Verlag, 2003.
2. A. Barvinok. Computing the Volume, Computing Integral Points, and Exponential Sums. In *Computational Geometry*, pages 161–170. ACM Press, 1992.

3. A. Barvinok. A Polynomial Time Algorithm for Counting Integral Points in Polyhedra When the Dimension is Fixed. *Mathematics of Operations Research*, 19(4):769–779, 1994.
4. W. Cook, M. Hartmann, R. Kannan, and C. McDiarmid. On Integer Points in Polyhedra. *Combinatorica*, 12(1):27–37, 1992.
5. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTREÉ analyzer. In *European Symposium on Programming*, volume 3444 of *Lecture Notes in Computer Science*, pages 21–30. Springer-Verlag, 2005.
6. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *Principles of Programming Languages*, pages 84–96. ACM Press, 1978.
7. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms*. Springer-Verlag, 1996.
8. J. A. Deloera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective Lattice Point Counting in Rational Convex Polytopes. *Journal of Symbolic Computation*, 38(4):1273–1302, 2004.
9. F. Eisenbrand. *Gomory-Chvátal Cutting Planes and the Elementary Closure of Polyhedra*. PhD thesis, Universität des Saarlandes, 2000.
10. M. E. Hartmann. *Cutting Planes and the Complexity of the Integer Hull*. PhD thesis, School of Operations Research and Industrial Engineering, Cornell University, 1988. Technical Report 819.
11. P. Huggins. iB4e: A Software Framework for Parametrizing Specialized LP Problems. In *International Congress on Mathematical Software*, volume 4151 of *Lecture Notes in Computer Science*, pages 245–247. Springer-Verlag, 2006.
12. M. Köppe. A Primal Barvinok Algorithm Based on Irrational Decompositions. *SIAM Journal on Discrete Mathematics*, 21(1):220–236, 2007.
13. R. Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–66, 2003.
14. B. Meister. Periodic Polyhedra. In *Compiler Construction*, volume 2985 of *Lecture Notes in Computer Science*, pages 134–149. Springer-Verlag, 2004.
15. H. L. Ong, H. C. Huang, and W. M. Huin. Finding the Exact Volume of a Polyhedron. *Advances in Engineering Software*, 34:351–356, 2003.
16. P. Quinton, S. V. Rajopadhye, and T. Risset. On Manipulating $\mathbb{Z}$-polyhedra using a Canonical Representation. *Parallel Processing Letters*, 7(2):181–194, 1997.
17. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constraint Based Linear Relations Analysis. In *Static Analysis Symposium*, volume 3148 of *Lecture Notes in Computer Science*, pages 53–68. Springer-Verlag, 2004.
18. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.
19. R. Seghir and V. Loechner. Memory Optimization by Counting Points in Integer Transformations of Parametric Polytopes. In *Compilers, Architectures, and Synthesis for Embedded Systems*, pages 74–82. ACM Press, 2006.
20. A. Simon. *Value-Range Analysis of C Programs*. Springer, 2008.
21. S. Verdoolaege, K. Beyls, M. Bruynooghe, and F. Catthoor. Experiences with Enumeration of Integer Projections of Parametric Polytopes. In *Compiler Construction*, volume 3443 of *Lecture Notes in Computer Science*, pages 91–105. Springer-Verlag, 2005.
22. S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe. Counting Integer Points in Parametric Polytopes Using Barvinok's Rational Functions. *Algorithmica*, 48(1):37–66, 2007.
23. D. K. Wilde. A Library for Doing Polyhedral Operations. Technical Report PI-785, IRISA, 1993.