An Algorithm for Finding Input-Output Constrained Convex Sets in an Acyclic Digraph

Gregory Gutin, Adrian Johnstone, Joseph Reddington, Elizabeth Scott, and Anders Yeo

Department of Computer Science, Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK

Abstract. A set X of vertices of an acyclic graph is convex if any vertex on a directed path between elements of X is itself in X. We construct an algorithm for generating all input-output constrained convex (IOCC) sets in an acyclic digraph, which uses several novel ideas. We show that our algorithm is more efficient than algorithms described in the literature in both the worst case and computational experiments. IOCC sets of acyclic digraphs are of interest in the area of modern embedded processor technology.

1 Introduction

In this paper we consider an algorithm for generating all input-output constrained convex sets in an acyclic digraph N. There is an immediate application for this algorithm in the field of embedded systems design. One of the major design choices for any new processor is the selection of the machine instruction set. In an embedded system, the processor will only execute a single fixed program during its lifetime, and significant efficiency gains can be made by choosing the machine instruction set, and associated hardware, to support the program that will be executed.

In particular there exist *extensible* general purpose processors such as the ARM OptimoDE, the MIPS Pro Series and the Tensilica Xtensa that can be customized for specific applications by the addition of custom-designed machine instructions and supporting hardware. The approach is to choose a set of application specific machine instructions by examination of the target program; candidate instructions are likely to involve the combination of several basic computations. For example, a program solving simultaneous linear equations may find it useful to have a single instruction to perform matrix inversion on a set of values held in registers.

Candidate instruction identification is carried out on *data dependency graphs* (DDGs), which are obtained from the application program by first splitting it into *basic blocks*, regions of sequential computation with no control transfer into their bodies, and then creating vertices for each instruction. There is an arc to each vertex u from those vertices whose instructions compute input operands of u. The resulting DDGs are acyclic and any convex subset of vertices is a candidate for a custom instruction which could be implemented in hardware. (A vertex set X is *convex* if it has the property that any vertex which lies on a path between vertices in X is itself in X, and convexity ensures that all of the inputs for the proposed instruction are available at the start of the instruction execution.)

We have given [4] an algorithm which efficiently finds all the connected convex vertex sets of an acyclic digraph N. However, in practice a given hardware application will have specific, and usually small, input and output constraints. This significantly reduces the size of the solution space and thus presents an opportunity for a more efficient enumeration algorithm. Furthermore, certain instructions, such as writes to main memory, cannot be combined into a custom instruction, thus certain vertices in the acyclic digraph can be designated as *forbidden* from the point of view of inclusion in a candidate set. Thus we are interested in finding all convex sets which have specified bounds, n_{in} and n_{out}, on the numbers of input and output vertices and which do not contain any vertices from a specified forbidden set F. For a convex set S, a vertex $i \in V(N) - S$ ($o \in S$) is called an *input vertex* (*output vertex*) if there is an arc from i to a vertex in S (there is an arc from o to a vertex not in S).

Bonzini and Pozzi [1] and Chen, Maskell and Sun [2] proved that with the two constraints above there are only polynomial number, $O(n^{n_{in}+n_{out}})$, of valid convex sets in an acyclic digraph N with n vertices provided n_{in} and n_{out} are constants (as they are in practice). The algorithm given in [1], the BP algorithm, has running time $O(n^{n_{in}+n_{out}+1})$. For an acyclic digraph N with unique source s (which is a vertex of in-degree zero) and a vertex set Q, a vertex set C is a *generalized dominator of* Q if each path from s to Q passes through a vertex in C, and for each vertex $c \in C$ there is a path from s to Q which contains only c and no other members of C. It was observed in [1] that if C is a generalized dominator of B in N then there is a convex set S in N with the set of input vertices C and the set of output vertices containing B. However, the converse it not true and, as a result, the BP algorithm does not generate all valid convex sets (in our experiments up to 25% of all valid convex sets were not generated by the BP algorithm); for a more detailed discussion, see [6].

Moreover, the BP algorithm is efficient only when the number $c(N)$ of valid convex sets in N is close to $\Theta(n^{n_{in}+n_{out}})$. In practice many acyclic digraphs N have significantly fewer valid convex sets. In such cases our valid convex set generation algorithm $\mathcal{A}$ described below, which is of time complexity $O(m \cdot n_{in}^2(c(N) + n^{n_{out}}) + m)$, is significantly faster (m is the number of arcs in N) than the BP algorithm. More importantly, $\mathcal{A}$ generates *all* valid convex sets.

In computational experiments, we have compared $\mathcal{A}$ with the state-of-the-art algorithm of Chen, Maskell and Sun [2] (CMS algorithm) and the well-known algorithm of Atasu, Pozzi and Ienne [5] (API algorithm). Our experiments clearly demonstrate that $\mathcal{A}$ is significantly faster than both the CMS and API algorithms.

For more information on modern embedded processors technology and convex set generating algorithms, see, e.g. [1,2,3,4,5].

In what follows, N denotes the acyclic digraph under consideration and F_0 is the initial set of forbidden vertices. By adding extra vertices to N and F_0 if necessary, without loss of generality we shall assume that N has a unique vertex s (*source*) with in-degree zero and a unique vertex t (*sink*) with out-degree zero. We assume that $s, t \in F_0$. Thus, every vertex lies on a directed path between two elements of F_0.

For a fixed pair n_{in}, n_{out} of positive integers and a set F of forbidden vertices, we say that a convex set S is *valid* if $S \cap F = \emptyset$ and the numbers of its input and output vertices are at most n_{in} and n_{out}, respectively. For vertex sets Y, Z, an arc yz with $y \in Y$ and $z \in Z$ is called an (Y, Z)-*arc* and a path (walk) starting in a vertex of Y and terminating in a vertex of Z is called a (Y, Z)-*path* ((Y, Z)-*walk*). In this paper, all walks (and, thus, paths and cycles) are directed.

2 Preliminary Results

Let O be an arbitrary set of vertices such that $|O| \leq n_{out}$ and the following holds: for every vertex $o \in O$ there is an (o, F)-path in $N - (O - \{o\})$. The condition guarantees that there is no convex set containing O with the output set $O' \subset O$.

Definition 1. *For a set Y of vertices in a digraph D, the* convex closure Y_D^{cl} *(or just Y^{cl} if D is clear from the context) is defined as $Y_D^{cl} = \{u \mid \exists (u, Y) - path \ \& \ (Y, u) - path\}$. That is, Y_D^{cl} contains all vertices with a path in D into Y and a path in D from Y. In particular $Y \subseteq Y^{cl}$.*

Let X and F be arbitrary sets of vertices in N, such that $O_N^{cl} \subseteq X$, $F_0 \subseteq F$ and $X \cap F = \emptyset$. We will give a recursive algorithm that finds all convex sets S with O as the output vertices, with at most n_{in} input vertices and with $X \subseteq S \subseteq V(N) - F$. However, before doing this we need the following definitions and lemmas.

Definition 2. *Given the above definitions, let N^* be obtained from N by deleting all arcs out of the vertices in O. Let D be obtained from N^* by coloring every arc $xy \in A(N^*)$ red and adding the blue-colored arc yx.*

Given a multiset $\mathcal{B}$ of arcs in D, let $D_{\mathcal{B}}$ denote the directed multigraph with $V(D_{\mathcal{B}}) = V(D)$ and $A(D_{\mathcal{B}}) = \mathcal{B}$. Note that $\mathcal{B}$ may contain several copies of the same arc in D and $D_{\mathcal{B}}$ may therefore contain parallel arcs.

Definition 3. *A multiset $\mathcal{W}$ of arcs in D is* $(D; F, X)$-feasible *if the following conditions hold.*

(i) $d^+_{D_{\mathcal{W}}}(f) \geq d^-_{D_{\mathcal{W}}}(f)$ *for all* $f \in F$;

(ii) $d^-_{D_{\mathcal{W}}}(x) \geq d^+_{D_{\mathcal{W}}}(x)$ *for all* $x \in X$;

(iii) *$d^-_{D_\mathcal{W}}(y) = d^+_{D_\mathcal{W}}(y)$ for all $y \in V(D_\mathcal{W}) - F - X$;*
(iv) *No distinct red arcs in $D_\mathcal{W}$ have the same initial vertex;*
(v) *There are no 2-cycles in $D_\mathcal{W}$.*

Define $R(\mathcal{W})$ as follows.

$$R(\mathcal{W}) = \sum_{f \in F} (d^+_{D_\mathcal{W}}(f) - d^-_{D_\mathcal{W}}(f)) = \sum_{x \in X} (d^-_{D_\mathcal{W}}(x) - d^+_{D_\mathcal{W}}(x))$$

Definition 4. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D and let $W = w_1 w_2 \dots w_k$ be a walk in D. Let $\mathcal{R}$ denote all vertices in D with a red arc out of them in $\mathcal{W}$ and let $RED(D)$ denote all red arcs in D. A vertex $w_i \in V(W)$ is said to be either $(\mathcal{W}, W)$-special, $(\mathcal{W}, W)$-normal or $(\mathcal{W}, W)$-forbidden depending on the following.*

(a) *w_i is $(\mathcal{W}, W)$-special if and only if $1 < i < k$, $w_i w_{i-1} \in \mathcal{W} \cap RED(D)$ and $w_i w_{i+1} \in RED(D)$.*
(b) *w_i is $(\mathcal{W}, W)$-normal if and only if w_i is not $(\mathcal{W}, W)$-special and the following holds: $i = k$ or $w_i w_{i+1} \notin RED(D)$ or $w_{i+1} w_i \in \mathcal{W}$ or $w_i \notin \mathcal{R}$.*
(c) *w_i is $(\mathcal{W}, W)$-forbidden if it is not $(\mathcal{W}, W)$-special or $(\mathcal{W}, W)$-normal. In other words, w_i is $(\mathcal{W}, W)$-forbidden if and only if $i < k$ and $w_i w_{i-1} \notin \mathcal{W} \cap RED(D)$ (or $i = 1$) and $w_i w_{i+1} \in RED(D)$ and $w_{i+1} w_i \notin \mathcal{W}$ and $w_i \in \mathcal{R}$.*

*We now define a $(D; \mathcal{W}; F, X)$-*feasible walk*, W, in D as any (F, X)-walk where for every vertex $x \in V(D)$, x appears at most once on W as a $(\mathcal{W}, W)$-special vertex, it appears at most once on W as a $(\mathcal{W}, W)$-normal vertex and it does not appear at all as a $(\mathcal{W}, W)$-forbidden vertex.*

Lemma 1. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D and let $W = w_1 w_2 \dots w_k$ be an (F, X)-walk in D. If no vertex on W is $(\mathcal{W}, W)$-forbidden then there exists a $(D; \mathcal{W}; F, X)$-feasible walk, W', in D from w_1 to w_k.*

Proof: Assume without loss of generality that W is the shortest walk from w_1 to w_k in D without any $(\mathcal{W}, W)$-forbidden vertices. For the sake of contradiction assume that W is not a $(D; \mathcal{W}; F, X)$-feasible walk, which implies that some vertex $x \in V(D)$ appears on W at least twice as a $(\mathcal{W}, W)$-special vertex or at least twice as a $(\mathcal{W}, W)$-normal vertex.

First assume that $x = w_i = w_j$ and $1 < i < j < k$ and both w_i and w_j are $(\mathcal{W}, W)$-special. This means that $w_1 w_2 \dots w_i w_{j+1} w_{j+2} \dots w_k$ is a walk from w_1 to w_k containing no $(\mathcal{W}, W)$-forbidden vertices (as w_i is still $(\mathcal{W}, W)$-special and no other vertex changes status). This contradicts the minimality of W.

So now assume that $x = w_i = w_j$ and $1 \le i < j \le k$ and both w_i and w_j are $(\mathcal{W}, W)$-normal. Again we note that $W' = w_1 w_2 \dots w_i w_{j+1} w_{j+2} \dots w_k$ is a walk from w_1 to w_k containing no $(\mathcal{W}, W)$-forbidden vertices (if $j = k$, then $W' = w_1w_2\dots w_i$). This again contradicts the minimality of W. These contradictions imply that W is a $(D;\mathcal{W};F,X)$-feasible walk. $\square$

Corollary 1. *Let $\mathcal{W}$ be a $(D;F,X)$-feasible multiset of arcs in D. If $W = w_1w_2\dots w_k$ and $W' = w_kw_{k+1}\dots w_l$ are $(D;\mathcal{W};F,X)$-feasible walks in D, then there exists a $(D;\mathcal{W};F,X)$-feasible walk from w_1 to w_l in D.*

Proof: Let $W^* = w_1w_2\dots w_kw_{k+1}\dots w_l$. As w_k is not a $(\mathcal{W},W^*)$-forbidden vertex on W^* (as otherwise w_k would be $(\mathcal{W},W')$-forbidden on W') we note that there are no $(\mathcal{W},W^*)$-forbidden vertices on W^*. We are now done by Lemma 1. $\square$

Recall that if $\mathcal{W}$ is a multiset of arcs and W is a walk, then if some arc appears i times in $\mathcal{W}$ and j times in $A(W)$ then it appears $i+j$ times in $A(W)\cup\mathcal{W}$.

Lemma 2. *Let $\mathcal{W}$ be a $(D;F,X)$-feasible multiset of arcs in D and let W be a $(D;\mathcal{W};F,X)$-feasible walk in D. Let $\mathcal{W}'$ be obtained from $A(W)\cup\mathcal{W}$ after deleting pairs xy, yx of arcs until there is no 2-cycles anymore. Then $\mathcal{W}'$ is a $(D;F,X)$-feasible multiset of arcs in D with $R(\mathcal{W}') = R(\mathcal{W})+1$.*

Proof: Let $\mathcal{W}'$ be defined as in the statement of the lemma and let $\mathcal{W}'' = A(W)\cup\mathcal{W}$. As W is a walk from F to X we note that (i), (ii) and (iii) in Definition 3 hold for $\mathcal{W}''$ and $R(\mathcal{W}'') = R(\mathcal{W})+1$. However this implies that (i), (ii) and (iii) in Definition 3 also hold for $\mathcal{W}'$ and $R(\mathcal{W}') = R(\mathcal{W})+1$, as deleting 2-cycles have no effect on $d^-(y)-d^+(y)$ for any $y\in V(D)$. By the definition of a $(D;\mathcal{W};F,X)$-feasible walk in D we note that (iv) in Definition 3 holds for $\mathcal{W}'$ (as the only way a vertex can increase the number of red arcs leaving it is if x is a $(\mathcal{W},W)$-normal vertex in W and x did not have any red arcs leaving it in $\mathcal{W}$). By the construction of $\mathcal{W}'$ we also note that (v) in Definition 3 holds for $\mathcal{W}'$. $\square$

We say that a set $\mathcal{S} = \{Q_1,Q_2,\dots,Q_p\}$ of paths and cycles in a directed multigraph M is a *decomposition* of M if each arc of M belongs to exactly one element of $\mathcal{S}$.

Lemma 3. *Let $\mathcal{W}$ be a $(D;F,X)$-feasible multiset of arcs in D. Then $D_{\mathcal{W}}$ can be decomposed into $W_1,W_2,\dots,W_{R(\mathcal{W})}$, $C_1,C_2,\dots,C_k$ (for some $k\ge 0$), such that W_i is a path from F to X in $D_{\mathcal{W}}$ for all $i=1,2,\dots,R(\mathcal{W})$ and C_j is a cycle in $D_{\mathcal{W}}$ for all $j=1,2,\dots,k$.*

Proof: If $R(\mathcal{W})=0$, $D_{\mathcal{W}}$ is eulerian and it is well-known that $D_{\mathcal{W}}$ can be decomposed into a number of cycles. So assume that $R(\mathcal{W})>0$. We will use induction on $|A(D_{\mathcal{W}})|$. Let $u_1\in F$ be any vertex with $d^+_{D_{\mathcal{W}}}(u_1) > d^-_{D_{\mathcal{W}}}(u_1)$. Starting at u_1 and moving to a successor vertex until we reach an already visited vertex, in which case we obtain a cycle, or we reach a vertex in X, in which case we obtain an (F,X)-path in $D_{\mathcal{W}}$. Remove the arcs of this cycle or path from $\mathcal{W}$ and use induction if $R(\mathcal{W})>0$ or the above case when $R(\mathcal{W})=0$. It is not difficult to see that this results in the desired decomposition, with exactly $R(\mathcal{W})$ (F,X)-paths in $D_{\mathcal{W}}$. $\square$

Lemma 4. *Let $\mathcal{W}$ be a $(D;F,X)$-feasible multiset of arcs in D, such that $R(\mathcal{W}) \leq n_{in}$. Assume that $O_N^{cl} \subseteq X$ and that for every vertex $o \in O$ there is an (o,F)-path in $N-(O-\{o\})$. Let Q be a set of vertices such that $X \subseteq Q \subseteq V(D)-F$. If there is no $(D;\mathcal{W};V(D)-Q,Q)$-feasible walk in D then Q is a convex set in N satisfying the input and output constraints. Furthermore, O is the set of output vertices of Q.*

Proof: Let Q be defined as stated in the lemma. Assume that Q is not convex, implying that there is a $w \notin Q$ such that there exists a (w,Q)-path, P, in N and a (Q,w)-path, P', in N. Let P'' be the reverse of P' and note that if no vertex of P' belongs to O, then P'' is a blue path in D from $w \in V(D)-Q$ to Q. As there is no $(D;\mathcal{W};V(D)-Q,Q)$-feasible walk in D we must have that some vertex on P' does indeed belong to O. Let $o_1 \in V(P') \cap O$ be arbitrary.

Let p be the terminal vertex of P and note that $p \in Q$. As there is a path, P^*, in N from p to F (recall that there exist (u,F)-paths for all $u \in V(N)$), we get a blue path from F to Q unless some vertex on the P^* belongs to O. Let this vertex be $o_2 \in V(P^*) \cap O$. By the above construction, we have an (o_1,w)-path in N and an (w,o_2)-walk in N (by merging P and part of P^*). However this implies that $w \in O_N^{cl} \subseteq X \subseteq Q$, a contradiction.

We will now prove that the input and output constraints are satisfied. Assume that there is some arc, xy, out of Q in N where $x \notin O$. Thus, yx is a blue arc in D and yx is a $(D;\mathcal{W};V(D)-Q,Q)$-feasible walk in D, a contradiction. So the only arcs out of Q in N come from O. Let $o \in O$ be arbitrary and recall that there is an (o,F)-path in $N-(O-\{o\})$, which implies that some vertex on this path is an output vertex for Q. Therefore, this vertex must be o, so we have now shown that O is exactly the output vertices for Q.

Assume that the input constraint is not satisfied and that $\{x_1,x_2,\ldots,x_r\}$ is a set of vertices in $V(N)-Q$ with arcs into Q in N and $r > n_{in}$. Let $\{y_1,y_2,\ldots,y_r\}$ be defined such that x_iy_i is an $(V(D)-Q,Q)$-arc in N for all $i \in \{1,2,\ldots,r\}$. By Lemma 3 let $W_1,W_2,\ldots,W_{R(\mathcal{W})}$, $C_1,C_2,\ldots,C_k$ be a decomposition of $\mathcal{W}$, such that W_i is a path from F to X in $D_{\mathcal{W}}$ for all $i=1,2,\ldots,R(\mathcal{W})$ and C_j is a cycle in $D_{\mathcal{W}}$ for all $j=1,2,\ldots,k$.

Assume that there is some $x_i \in \{x_1,x_2,\ldots,x_r\}$ such that there is no red (x_i,Q)-arc in $\mathcal{W}$. If there is no red arc out of x_i in $\mathcal{W}$ at all, then the arc x_iy_i contradicts the fact that there is no $(D;\mathcal{W};V(D)-Q,Q)$-feasible walk in D. So let x_iu be a red arc in $\mathcal{W}$ where $u \notin Q$. However the path ux_iy_i again contradicts the fact that there is no $(D;\mathcal{W};V(D)-Q,Q)$-feasible walk in D. So for every vertex in $\{x_1,x_2,\ldots,x_r\}$ there exists a red (x_i,Q)-arc in $\mathcal{W}$. Without loss of generality we may assume that $\{y_1,y_2,\ldots,y_r\}$ was chosen such that x_iy_i is red and $x_iy_i \in \mathcal{W}$ for all $i=1,2,\ldots,k$. If for some $i \in \{1,2,\ldots,k\}$ the arc x_iy_i belongs to a cycle $C_a \in \{C_1,C_2,\ldots,C_k\}$, then there is a $(Q,V(D)-Q)$-arc, uv, in C_a. However, the path vu is a $(D;\mathcal{W};V(D)-Q,Q)$-feasible walk in D, a contradiction.

As $r > n_{in} \geq R(\mathcal{W})$ there must be some path in $\{W_1,W_2,\ldots,W_{R(\mathcal{W})}\}$ that contains at least two arcs from $\{x_1y_1,x_2y_2,\ldots,x_ry_r\}$. Without loss of generality assume that x_1y_1 is the first such arc on W_1 and x_2y_2 is the second such arc on W_1. This implies that there is a walk from $y_1 \in Q$ to $x_2 \notin Q$, containing a $(Q, V(D) - Q)$-arc, uv. However the path vu is a $(D; \mathcal{W}; V(D) - Q, Q)$-feasible walk in D, a contradiction. This contradiction against $r > n_{in}$ implies that the input constraint is satisfied. □

Lemma 5. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D, such that $R(\mathcal{W}) > n_{in}$. Then there is no convex set of vertices, Q, in N, such that $X \subseteq Q \subseteq V(D) - F$ and Q satisfies the input constraint and has O as its output vertices.*

Proof: For the sake of contradiction assume that there exists a convex set of vertices, Q, in N, such that $X \subseteq Q \subseteq V(D) - F$ and Q satisfies the input constraint and has O as its output vertices. Let $I = \{i_1, i_2, \dots, i_r\}$ be the input vertices for Q and $r \leq n_{in}$. By Lemma 3 let $W_1, W_2, \dots, W_{R(\mathcal{W})}, C_1, C_2, \dots, C_k$ be a decomposition of $\mathcal{W}$, such that W_i is a path from F to X in $D_{\mathcal{W}}$ for all $i = 1, 2, \dots, R(\mathcal{W})$ and C_j is a cycle in $D_{\mathcal{W}}$ for all $j = 1, 2, \dots, k$. As $r \leq n_{in} < R(\mathcal{W})$ there must be some path W_i, without loss of generality say W_1, which does not contain a red arc out of any vertex in I (as each vertex in I has at most one red arc out of it in $\mathcal{W}$). Let uv be a $(V(D) - Q, Q)$-arc on W_1. If uv is a red arc then $u \notin I$, contradicting the fact that I is the set of input vertices. So uv is a blue $(V(D) - Q, Q)$-arc in D. Hence $u \in N^*$ and $v \in O$ contrary to the definition of N^*. □

Lemma 6. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D. In time $O(|V(N)| + |A(N)|)$ we can find a $(D; \mathcal{W}; F, X)$-feasible walk in D if it exists or determine that it does not exist. If it does not exist we can also determine the following two sets:*

$$S = \{u \mid \textit{there is a } (D; \mathcal{W}; F, \{u\})\textit{-feasible walk in } D\}$$
$$T = \{v \mid \textit{there is a } (D; \mathcal{W}; \{v\}, X)\textit{-feasible walk in } D\}$$

Proof: We will define a digraph D' as follows. Let $\mathcal{R}$ contain all vertices in D which have a red arc out of them in $\mathcal{W}$. Let the vertex set of D' be $V(D') = V(D) \cup \{r' \mid r \in \mathcal{R}\}$ (that is we duplicate all vertices in $\mathcal{R}$). For all arcs $uv \in A(D)$ add the following arcs to D'.

(R1) If uv is red and $vu \notin \mathcal{W}$ and $u \in \mathcal{R}$, then add $u'v$ to D'.
(R2) If uv is red and $vu \in \mathcal{W}$ or $u \notin \mathcal{R}$, then add uv to D'.
(B1) If uv is blue and $vu \in \mathcal{W}$, then add uv and uv' to D'.
(B2) If uv is blue and $vu \notin \mathcal{W}$, then add uv to D'.

Now use any algorithm such as depth (or breadth) first search to find an (F, X)-path in D' if it exists (we start and end in vertices of the form u and not u'). First assume that such a path $P' = x_1x_2 \dots x_k$ exists. Replacing vertices of the form u' with u, we obtain a walk $W = w_1w_2 \dots w_k$ in D, which we will show is a $(D; \mathcal{W}; F, X)$-feasible walk. If $x_i = w_i'$ then, since there are no arcs of the form $y'z'$ in D', we must have $x_{i-1} = w_{i-1}$ and $x_{i+1} = w_{i+1}$. Thus, it is not difficult to see that w_i a $(\mathcal{W}, W)$-special vertex in W. Whereas if $w_i = x_i$ then either $i = k$ or **R2**, **B1** or **B2** holds for $w_i w_{i+1}$ and so w_i is $(\mathcal{W}, W)$-normal in W. Thus, since P' is a path, W is indeed a $(D; \mathcal{W}; F, X)$-feasible walk in D.

Now assume that some $(D; \mathcal{W}; F, X)$-feasible walk, W, in D exists. Let $W = w_1 w_2 w_3 \cdots w_l$. If w_i is $(\mathcal{W}, W)$-special in W then change it to w_i' in D'. After doing this for all special vertices we note that we get a path from F to X in D'. So we have now shown that there exists a $(D; \mathcal{W}; F, X)$-feasible walk in D if and only if there exists an (F, X)-path in D'. This gives us the correct time complexity as $|V(D')| \leq 2|V(D)|$ and $|A(D')| \leq 2|A(D)|$.

If there is no (F, X)-path in D' then it is not difficult to find the set of vertices $Z \subseteq V(D')$, such that there is an (F, z)-path in D' if and only if $z \in Z$. Now let $S = \{u \in V(D) \mid u \in Z\}$ (note that $\{u \in V(D) \mid u \in Z \text{ or } u' \in Z\}$ is an equivalent definition of S). It is not difficult to see that S is the set of vertices in $V(D)$ for which there exists a $(D; \mathcal{W}; F, \{s\})$-feasible walk in D. Analogously we can find T. □

3 The Algorithm

The algorithm $\mathcal{A}(N, F)$ described below makes a call to $\mathcal{B}(\emptyset, F, X)$ for all possible output sets O (see A.2) where $X = O^{cl}_N$. The procedure $\mathcal{B}(\mathcal{W}, F, X)$ will then find all convex sets, Q, satisfying the input constraint and having O as the output vertices and satisfying $X \subseteq Q \subseteq V(N) - F$.

Lemma 7. *The sets saved by $\mathcal{A}(N, F)$ are precisely the valid convex sets of N and furthermore no such set is saved more than once.*

Proof: We only save solutions in B.4 or B.5.2. In both cases, using Corollary 1 and Lemma 4, it can be seen that the saved set is a valid convex set.

Now let Q' be a valid convex set. Let O' be the output vertices of Q'. Assume that for some $o' \in O'$ there is no (o', F)-path in $N - (O' - \{o'\})$. Let $y \in N^+_N(o')$ be arbitrary and assume that $y \notin Q'$. However there is no (y, F)-path in $N - (O' - \{o'\})$ but there is a (y, F)-path in N. This implies that there is a $(y, O' - \{o'\})$-path in N. Therefore $y \in Q'$ (as Q' is convex), a contradiction. So for every $o' \in O'$ there is a (o', F)-path in $N - (O' - \{o'\})$. Note that $X = (O')^{cl}_N \subseteq Q'$, as Q' is convex and we will make a call to $\mathcal{B}(\emptyset, F, X)$ in A.2.3 of $\mathcal{A}(N, F)$.

If we return in B.1, then we did not have $X \subseteq Q' \subseteq V(D) - F$, by Lemma 5. If we make recursive calls in B.3, then the desired recursive call is $\mathcal{B}(\mathcal{W}, F, (X \cup \{u\})^{cl}_N)$ if $u \in Q'$ (note that if $X \cup \{u\} \subseteq Q'$ and Q' is convex, then $(X \cup \{u\})^{cl}_N \subseteq Q'$) and $\mathcal{B}(\mathcal{W}, F \cup \{u\}, X)$ if $u \notin Q'$. Now assume that T is saved in $B.4$. We note that $S \cap V(Q') = \emptyset$ by Lemma 5 (as otherwise we obtain a $(D; F, Q')$-feasible multiset, $\mathcal{W}'$, with $R(\mathcal{W}') > n_{in}$, by adding a $(D; \mathcal{W}; F, \{s\})$-feasible walk to $\mathcal{W}$, where $s \in S \cap V(Q')$). Analogously $T - V(Q') = \emptyset$ by Lemma 5 (as again we obtain a $(D; V(D) - Q', Q')$-feasible multiset, $\mathcal{W}'$, with $R(\mathcal{W}') > n_{in}$, by adding a $(D; \mathcal{W}; \{t\}, X)$-feasible walk to $\mathcal{W}$, where $t \in T - V(Q')$). Therefore $Q' = T$ and Q' is saved.

```
Algorithm $\mathcal{A}(N, F)$
  A.1 Find an acyclic order $u_1, u_2, \ldots, u_n$ of the vertices in $N$ (that
      is, if $u_iu_j \in A(N)$ then $i < j$).
  A.2 For all sets $O \subseteq V(N)$, where for every vertex $o \in O$ there is
      an $(o, F)$-path in $N-(O-\{o\})$ and $|O| \le n_{out}$ do the following.
    A.2.1 Find $N^*$ and $D$ as in Definition 2.
    A.2.2 Let $X = O^{cl}_N$ (see Definition 1).
    A.2.3 Make a call to $\mathcal{B}(\emptyset, F, X)$ (see below).
```

```
Algorithm $\mathcal{B}(\mathcal{W}, F, X)$
  B.1 If $R(\mathcal{W}) > n_{in}$ or if $X \cap F \neq \emptyset$, then there is no solution so return.
  B.2 Use Lemma 6 to determine if there is a $(D; \mathcal{W}; F, X)$-feasible walk in $D$.
      If there is and $R(\mathcal{W}) \le n_{in}$ then add it to $\mathcal{W}$ using the approach in Lemma
      2 and go to B.1. Otherwise determine $S$ and $T$ as in Lemma 6.
  B.3 If $V(D) \neq S \cup T$ then let $u \in V(D) - S - T$ be arbitrary. Now make
      recursive calls $\mathcal{B}(\mathcal{W}, F, (X \cup \{u\})^{cl}_N)$ and $\mathcal{B}(\mathcal{W}, F \cup \{u\}, X)$ and return.
  B.4 If $R(\mathcal{W}) = n_{in}$, then by B.3 we have $V(D) = S \cup T$. In this case save $T$
      as a solution and return.
  B.5 If $R(\mathcal{W}) < n_{in}$ then consider the following possibilities. Note that $V(D) =$
      $S \cup T$.
    B.5.1 If $X \neq T$ then let $u_a \in T - X$ be chosen such that $a$ is minimum.
          Make the recursive calls $\mathcal{B}(\mathcal{W}, F, (X \cup \{u_a\})^{cl}_N)$ and $\mathcal{B}(\mathcal{W}, F \cup \{u_a\}, X)$ and
          return.
    B.5.2 If $X = T$, then save $T$ as a solution. Let $i_1, i_2, \ldots, i_k$ be the vertices
          in $V(D) - T$ with red arcs into $T$ in $D$. Make the following recursive calls.
          $\mathcal{B}(\mathcal{W}, F, (X \cup \{i_1\})^{cl}_N)$,
          $\mathcal{B}(\mathcal{W}, F \cup \{i_1\}, (X \cup \{i_2\})^{cl}_N)$,
          $\mathcal{B}(\mathcal{W}, F \cup \{i_1, i_2\}, (X \cup \{i_3\})^{cl}_N)$,....,
          $\mathcal{B}(\mathcal{W}, F \cup \{i_1, i_2, \ldots, i_{k-1}\}, (X \cup \{i_k\})^{cl}_N)$.
```

If we make recursive calls in B.5.1, then the desired recursive call is $\mathcal{B}(\mathcal{W}, F, (X \cup \{u_a\})^{cl}_N)$ if $u_a \in Q'$ and $\mathcal{B}(\mathcal{W}, F \cup \{u_a\}, X)$ if $u_a \notin Q'$. If we perform B.5.2, then we return Q' if $\{i_1, i_2, \ldots, i_k\} \cap Q' = \emptyset$. If $\{i_1, i_2, \ldots, i_k\} \cap Q' \neq \emptyset$ then let j be the minimum index such that $i_j \in \{i_1, i_2, \ldots, i_k\} \cap Q'$ and note that the desired recursive call is $\mathcal{B}(\mathcal{W}, F \cup \{i_1, i_2, \ldots, i_{j-1}\}, (X \cup \{i_j\})^{cl}_N)$.

We have shown that Q' will be saved and we will now prove that Q' cannot be saved twice. As we only consider sets O with the property that for every vertex $o \in O$ there is an (o, F)-path in $N - (O - \{o\})$, we note that Q' cannot be saved in two distinct calls in A.2.3 (only when O is exactly the output set of Q'). Furthermore as all recursive calls either add a vertex to the forbidden set or that same vertex to X the same set cannot be saved in two different recursive calls (in B.5.2. we noted above exactly in which recursive call Q' would be saved if $\{i_1, i_2, \ldots, i_k\} \cap Q' \neq \emptyset$ and otherwise it would not be saved in any recursive call, but only in the current call). $\square$

We omit proof of the following due to the limited space.

Lemma 8. *If N is a connected acyclic digraph of order n, size m and containing $c(N)$ valid convex sets, then $\mathcal{A}(N, F)$ has time complexity $O(m \cdot N_{in}^2(c(N) + n^{N_{out}}) + m)$.*

The last two lemmas imply the following:

Theorem 1. *Let N be an acyclic digraph with n vertices and m arcs. The algorithm $\mathcal{A}(N, F)$ finds all valid convex sets in N in time $O(m \cdot n_{in}^2(c(N) + n^{n_{out}}) + m)$, where $c(N)$ is the number of valid convex sets.*

4 Experiments

We have implemented $\mathcal{A}$ and tested it against the state-of-the-art algorithm of Chen, Maskell and Sun [2] (the CMS algorithm) and the well-known algorithm of Atasu, Pozzi and Ienne [5] (the API algorithm) using both synthetic examples and DDGs generated from real world applications. The source of the CMS implementation was kindly provided by its authors. All algorithms were implemented in C++ and experimental data were produced using Dual Core AMD Opteron 265 1.8GHz processors with 4Gbyte of RAM, running 64-bit SUSE Linux 10.2.

Figure 1 shows the performance of these algorithms on synthetic tree and acyclic lattice digraphs with $n_{in} = 3$ and $n_{out} = 2$. In both cases, algorithm $\mathcal{A}$ consistently outperforms the current state of the art with the performance of CMS only slightly superior to the API algorithm on tree-like graphs.

Table 1 shows results from five real world C++ programs in the MiBench benchmark suite [3]. We selected a large (150–1800 lines of intermediate code) basic block from within a critical loop of each program: typically the compiler will have unrolled this block to some degree. The resulting DDGs were augmented

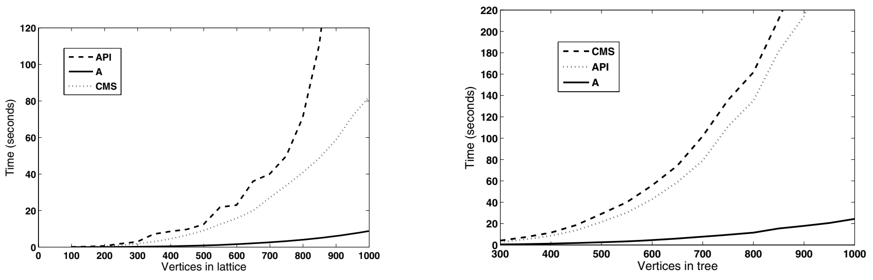

Fig. 1. Performance on lattice and tree synthetic digraphs under I/O constraints

Table 1. Comparative performance on real world examples

Input	n_{in}	n_{out}	$c(N)$ sets	Time CMS	Time API06	Time $\mathcal{A}$	Calls CMS	Calls API06	Calls $\mathcal{A}$
bf	2	1	482	0.04	2.78	0.01	24,009	796,775	631
	4	1	1,920	0.07	15.71	0.04	34,084	4,467,923	3,507
	6	1	7,669	0.11	34.61	0.17	55,374	9,816,778	15,005
	3	2	7,831	0.35	91.51	0.15	176,631	25,169,197	12,302
	5	2	40,714	0.79	352.95	0.92	383,570	101,091,122	75,376
	4	3	105,599	2.31	DNF	2.81	1,122,520	DNF	189,037
	6	3	570,197	7.02	DNF	14.57	3,342,391	DNF	1,085,505
	8	3	2,155,103	17.37	DNF	71.95	8,329,766	DNF	4,253,251
cjpeg	2	1	406	0.02	0.10	0.00	21,907	61,832	694
	4	1	544	0.02	0.10	0.00	22,003	70,216	970
	6	1	550	0.01	0.11	0.00	22,003	70,216	982
	3	2	41,363	0.61	13.86	0.28	677,813	9,880,064	76,889
	5	2	113,611	0.82	19.30	1.02	875,155	13,460,590	220,929
	7	2	140,335	0.94	20.15	1.53	896,688	13,721,462	274,377
	4	3	2,201,568	20.50	DNF	18.78	18,454,621	DNF	4,236,388
rijndael	2	1	1241	2.79	51.43	0.05	697778	5473096	1636
	4	1	4,787	3.51	253.30	0.17	786,732	27,471,175	8,728
	6	1	15,236	4.09	DNF	0.59	878,083	DNF	29,626
	3	2	75,241	83.96	DNF	3.98	11,575,641	DNF	145,477
	5	2	648,748	201.41	DNF	23.88	31,777,459	DNF	1,207,733
sha	2	1	1,546	3.76	DNF	0.13	300,752	DNF	1,632
	4	1	4,372	4.23	DNF	0.24	345,994	DNF	7,284
	6	1	10,152	5.30	DNF	0.49	432,350	DNF	18,844
	3	2	78,132	85.14	DNF	6.75	6,450,724	DNF	117,159
	5	2	293,259	164.97	DNF	15.37	12,652,418	DNF	494,521
md5	2	1	893	1.54	DNF	0.04	329,373	DNF	969
	4	1	2,304	1.66	DNF	0.08	342,133	DNF	3,791
	6	1	3,546	1.70	DNF	0.12	349,486	DNF	6,275
	3	2	54,476	51.45	DNF	3.24	6,109,809	DNF	102,389

with forbidden vertices, which represent values external to the basic block to give the following examples: the BlowFish encryption algorithm (bf) with 467 vertices of which 134 are forbidden; JPEG image compression (cjpeg) with 152 vertices, 34 forbidden; AES encryption (rijndael) with 1237 vertices, 391 forbidden; secure message digest hashing (sha) with 1811 vertices, 351 forbidden; and MD5 with 1170 vertices, 353 forbidden.

The API algorithm is not competitive with either $\mathcal{A}$ or CMS for these graphs, supporting the conclusions in [2]. Although the CMS algorithm performs better on some small to medium examples, on large examples the effect of the better asymptotic complexity is clear — on the rijindel, sha and md5 benchmarks, $\mathcal{A}$ clearly has a performance advantage. We also note that in every case, the number of recursions made by $\mathcal{A}$ was significantly lower.

References

1. Bonzini, P., Pozzi, L.: Polynomial-time subgraph enumeration for automated instruction set extension. In: DATE 2007, pp. 1331–1336 (2007)
2. Chen, X., Maskell, D.L., Sun, Y.: Fast identification of custom instructions for extensible processors. IEEE Trans. Computer-Aided Design Integr. Circuits Syst. 26, 359–368 (2007)
3. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: A free, commercially representative embedded benchmark suite. In: Proceedings of the WWC-4, 2001 IEEE International Workshop on Workload Characterization, pp. 3–14 (2001)
4. Gutin, G., Johnstone, A., Reddington, J., Scott, E., Soleimanfallah, A., Yeo, A.: An algorithm for finding connected convex subgraphs of an acyclic digraph. In: Algorithms and Complexity in Durham, 2007. College Publications (2008)
5. Pozzi, L., Atasu, K., Ienne, P.: Exact and approximate algorithms for the extension of embedded processor instruction sets. IEEE Trans. on CAD of Integrated Circuits and Systems 25, 1209–1229 (2006)
6. Reddington, J.: Improvements to instruction identification for custom instruction set design. PhD Thesis, Royal Holloway, University of London (2008)