Polynomial-time Fence Insertion For Structured Programs

M. Taheri, A. Pourdamghani, M. Lesani
University of California, Riverside
Sharif University of Technology
Correctness Dependent on Program Order

Dekker Mutual Exclusion Algorithm

```c
1 bool p0()
2   flag0 = 1
3   if (flag1 == 1)
4       return false;
5   // critical
6   flag0 = 0;
7   return true;
8 }

1 bool p1()
2   flag1 = 1
3   if (flag0 == 1)
4       return false;
5   // critical
6   flag1 = 0;
7   return true;
8 }
```
Weak (relaxed) memory models

Out of order execution
Both x86 and ARMv7
Fence Instructions

```c
1 bool p0()
2   flag0 = 1
3   fence
4   if (flag1 == 1)
5       return false;
6   // critical
7   flag0 = 0;
8   return true;
9 }

1 bool p1()
2   flag1 = 1
3   fence
4   if (flag0 == 1)
5       return false;
6   // critical
7   flag1 = 0;
8   return true;
9 }
```
1 bool p0()
2   flag0 = 1
3   if (flag1 == 1)
4       return false;
5      // critical
6   flag0 = 0;
7   return true;
8 }

{2 -> 3}
Fence Insertion for Straight-line Programs

- Straight-line Programs
- Polynomial greedy algorithm

(read(i))

(write(i, v))

(read(i))

(write(i, v))
Fence Insertion for structured programs

Figure 2
An example of AFG and its constraints. A constraint $\bar{s, t}$ is shown as a dashed arrow from the source $s$ to the sink $t$. The diamonds $\bar{e, m}$ and $\bar{g, n}$ are at level $0$. The diamond $\bar{c, o}$ is at level $1$ and the diamond $\bar{b, p}$ is at level $2$. The constraint $\bar{h, k}$ is an internal constraint for the diamond $\bar{e, m}$, the constraint $\bar{d, j}$ is a spanning constraint for the diamond $\bar{g, n}$, and the constraint $\bar{d, p}$ is a passing constraint for the diamond $\bar{g, n}$.

For example, fence insertion for the graph shown in Figure 1.(a) is reduced to fence insertion for the two paths shown in Figure 1.(b). The high-level idea is that we can incrementally transform a diamond to a single branch by extracting branches. Fences can be independently inserted for the extracted branches. For example, the right branch of Figure 1.(a) is extracted in Figure 1.(b). The orders within a branch can be only preserved by fences inserted within that branch. Thus, the extracted right branch takes the order from $d$ to $f$ with it. Further, the extracted right branch can cover the spanning order from $e$ to $h$ with no extra fences. Thus, it takes in the spanning order from $e$ to $g$ too; it takes it as the shrunk order from $e$ to $g$. Thus, fence insertion for the extracted right branch covers both constraints from $d$ to $f$ and from $e$ to $h$.

The left branch and the vertices above and below the diamond make the second path. The order from $c$ to $h$ overlaps with the left branch and stays within the second extracted path. The result is two paths and fencing for each can be done in polynomial time.

We will elaborate on the algorithm in the following sections.

In the following sections, we first define the problem model (section 2) and then present the greedy fence insertion algorithm for loop-free structured programs and state its optimality and complexity (section 3). We then present a reduction from fence insertion for looping programs to fence insertion for loop-free programs (section 4). Then, we prove the NP-hardness of fence insertion with multiple fence types (section 5). Finally, we discuss the related works (section 6) before we conclude (section 7).
1. A greedy and polynomial-time optimum fence insertion algorithm for Structured programs.
2. The minimum fence insertion problem with multiple types of fence instructions is NP-hard.
Fence Insertion for loop-free programs

1. Constraint Elimination
2. Finding Diamonds
3. Diamond Decomposition
4. Fence Insertion for Simple Paths
Algorithm 3: Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion (q)
   2:   \( q \) is the minimum priority queue ordering diamonds by level
   3: initialize the set \( F \) to \( \emptyset \).
   4:   while \( q \) is not empty do
   5:     extract the innermost diamond \( d \) from \( q \).
   6:     while there is more than one path in \( d \) do
   7:       pick a path \( p \) in \( d \).
   8:       call Algorithm 4 on \( p \) to find the fencing \( f \) and the type \( t \).
   9:     add \( f \) to \( F \).
   10:    if \( t \) is absorbing then
   11:       update the end point of the spanning constraints to the merge point of \( d \).
   12:    else \( t \) is emitting
   13:       update the start point of the spanning constraints to the merge point of \( d \).
   14:       remove \( p \) from \( d \).
   15:   return \( F \).

Figure 4: Transformation of spanning constraints.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are \( \{e, m\} \) and \( \{g, n\} \). For the diamond \( \{e, m\} \), the left simple path \( ehikm \) is extracted. The constraint \( \{k, p\} \) is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge \( \{i, k\} \) that does not cover the spanning constraint \( \{k, p\} \). So the path is emitting and the constraint \( \{k, p\} \) is shrunk to \( \{m, p\} \). Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is \( \{g, n\} \). The left edge \( \{g, n\} \) with no constraint can be simply extracted to reduce the diamond to a simple path. Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a). The only diamond of the next level is \( \{c, o\} \). There are no spanning constraints and simply extracting the right path \( cfl0 \) reduces the diamond to a simple path. Figure 5.(c) has only
Algorithm 3
Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion (q)
2: // q is the minimum priority queue ordering diamonds by level
3: Initialize the set F to ∅.
4: while q is not empty do
5: Extract the innermost diamond d from q.
6: while there is more than one path in d do
7: Pick a path p in d.
8: Call Algorithm 4 on p to find the fencing f and the type t.
9: Add f to F
10: if t is absorbing then
11: Update the end point of the spanning constraints to the merge point of d.
12: else if t is emitting then
13: Update the start point of the spanning constraints to the merge point of d.
14: Remove p from d.
15: return F

Figure 4
Transformation of spanning constraints.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are [e, m] and [g, n]. For the diamond [e, m], the left simple path ehikm is extracted. The constraint [k, p] is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge [i, k] that does not cover the spanning constraint [k, p]. So the path is emitting and the constraint [k, p] is shrunk to [m, p]. Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is [g, n]. The left edge [g, n] with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a). The only diamond of the next level is [c, o]. There are no spanning constraints and simply extracting the right path cflor reduces the diamond to a simple path. Figure 5.(c) has only

Constraint Elimination
**Algorithm 3: Decomposing Diamonds into a Set of Simple Paths**

1. **procedure** `FenceInsertion (q)`
   - `q` is the minimum priority queue ordering diamonds by level

2. Initialize the set `F` to 

3. while `q` is not empty
   - Extract the innermost diamond `d` from `q`.
   - while there is more than one path in `d`
     - Pick a path `p` in `d`.
     - Call Algorithm 4 on `p` to find the fencing `f` and the type `t`.
     - Add `f` to `F`
     - if `t` is absorbing
       - Update the end point of the spanning constraints to the merge point of `d`.
     - else
       - Update the start point of the spanning constraints to the merge point of `d`.
     - Remove `p` from `d`.

4. **return** `F`

**Figure 4.(b)**, the constraint `Èf,hÍ` is updated to `Èg, hÍ`. We note that the transformation leaves the internal and passing constraints unchanged.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are `Èe, mÍ` and `Èg, nÍ`. For the diamond `Èe, mÍ`, the left simple path `ehikm` is extracted.

The constraint `Èk, pÍ` is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge `Èi, kÍ` that does not cover the spanning constraint `Èk, pÍ`. So the path is emitting and the constraint `Èk, pÍ` is shrunk to `Èm, pÍ`. Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is `Èg, nÍ`. The left edge `Èg, nÍ` with no constraint can be simply extracted to reduce the diamond to a simple path.

**Figure 5.(c)** shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a). The only diamond of the next level is `Èc, oÍ`. There are no spanning constraints and simply extracting the right path `cflo` reduces the diamond to a simple path. Figure 5.(c) has only
(a) Example diamond 1

(b) The diamond (a) after constraint elimination. The path \( bceg \) is absorbing. The path \( bdfg \) is emitting.

(c) The diamond (b) after the transformation of the spanning constraints. The constraints \( \overline{e, h} \) and \( \overline{e, g} \) are updated to \( \overline{g, h} \).

Figure 4. Transformation of spanning constraints.

Algorithm 3

Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion \( q \)
2: \( q \) is the minimum priority queue ordering diamonds by level
3: Initialize the set \( F \) to \( \emptyset \).
4: while \( q \) is not empty do
5: Extract the innermost diamond \( d \) from \( q \).
6: while there is more than one path in \( d \) do
7: Pick a path \( p \) in \( d \).
8: Call Algorithm 4 on \( p \) to find the fencing \( f \) and the type \( t \).
9: Add \( f \) to \( F \).
10: if \( t \) is absorbing then
11: Update the end point of the spanning constraints to the merge point of \( d \).
12: else \( t \) is emitting
13: Update the start point of the spanning constraints to the merge point of \( d \).
14: Remove \( p \) from \( d \).
15: return \( F \).

Figure 5. (b), the constraint \( \overline{f,h} \) is updated to \( \overline{g,h} \). We note that the transformation leaves the internal and passing constraints unchanged.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are \( \overline{e,m} \) and \( \overline{g,n} \). For the diamond \( \overline{e,m} \), the left simple path \( ehikm \) is extracted.

The constraint \( \overline{k,p} \) is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge \( \overline{i,k} \) that does not cover the spanning constraint \( \overline{k,p} \). So the path is emitting and the constraint \( \overline{k,p} \) is shrunk to \( \overline{m,p} \). Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is \( \overline{g,n} \). The left edge \( \overline{g,n} \) with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a). The only diamond of the next level is \( \overline{c,o} \). There are no spanning constraints and simply extracting the right path \( cflo \) reduces the diamond to a simple path. Figure 5.(c) has only

Constraint Elimination
Algorithm 3: Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion (q)
2: \( q \) is the minimum priority queue ordering diamonds by level
3: Initialize the set \( F \) to \( \emptyset \).
4: while \( q \) is not empty do
5: Extract the innermost diamond \( d \) from \( q \).
6: while there is more than one path in \( d \) do
7: Pick a path \( p \) in \( d \).
8: Call Algorithm 4 on \( p \) to find the fencing \( f \) and the type \( t \).
9: Add \( f \) to \( F \).
10: if \( t \) is absorbing then
11: Update the end point of the spanning constraints to the merge point of \( d \).
12: else if \( t \) is emitting then
13: Update the start point of the spanning constraints to the merge point of \( d \).
14: Remove \( p \) from \( d \).
15: return \( F \)

Figure 4.(b), the constraint \( \overline{e, h} \) is updated to \( \overline{g, h} \). We note that the transformation leaves the internal and passing constraints unchanged.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are \( \overline{e, m} \) and \( \overline{g, n} \). For the diamond \( \overline{e, m} \), the left simple path \( ehikm \) is extracted.

The constraint \( \overline{k, p} \) is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge \( \overline{i, k} \) that does not cover the spanning constraint \( \overline{k, p} \). So the path is emitting and the constraint \( \overline{k, p} \) is shrunk to \( \overline{m, p} \). Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is \( \overline{g, n} \). The left edge \( \overline{g, n} \) with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a). The only diamond of the next level is \( \overline{c, o} \). There are no spanning constraints and simply extracting the right path \( cflo \) reduces the diamond to a simple path. Figure 5.(c) has only
Algorithm 3
Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion (q)
2:  q is the minimum priority queue ordering diamonds by level
3:  Initialize the set F to Ø.
4:  while q is not empty do
5:     Extract the innermost diamond d from q.
6:     while there is more than one path in d do
7:         Pick a path p in d.
8:         Call Algorithm 4 on p to find the fencing f and the type t.
9:         Add f to F
10:     if t is absorbing then
11:        Update the end point of the spanning constraints to the merge point of d.
12:     else if t is emitting then
13:        Update the start point of the spanning constraints to the merge point of d.
14:     Remove p from d.
15:  return F

Figure 4 (b), the constraint \( \bar{e}, h \) is updated to \( \bar{g}, h \). We note that the transformation leaves the internal and passing constraints unchanged.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5 (a). Figure 5 (b) shows the result of Algorithm 3 on Figure 5 (a) after processing the diamonds of level 0. The diamonds of level 0 are \( \bar{e}, m \) and \( \bar{g}, n \). For the diamond \( \bar{e}, m \), the left simple path ehikm is extracted. The constraint \( \overline{k, p} \) is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge \( \overline{i, k} \) that does not cover the spanning constraint \( \overline{k, p} \). So the path is emitting and the constraint \( \overline{k, p} \) is shrunk to \( \bar{m}, p \). Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is \( \bar{g}, n \). The left edge \( \overline{g, n} \) with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5 (c) shows the result of Algorithm 3 on the graph of Figure 5 (b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5 (a). The only diamond of the next level is \( \bar{c}, o \). There are no spanning constraints and simply extracting the right path cflo reduces the diamond to a simple path. Figure 5 (c) has only
Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex $a$ is 0. The labels of the subsequent branch vertices $b$, $c$, and $e$ are 1, 2, and 3 respectively. The labels of the merge vertices $m$ and $o$ are both 3. Thus, the level of the diamond $e, m$ is 0, and the level of the diamond $c, o$ is 1. The algorithm finds four diamonds with the following levels: level $e, m = 0$, level $g, n = 0$, level $c, o = 1$, and level $b, p = 2$.

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path. Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows
Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex \( a \) is 0. The labels of the subsequent branch vertices \( b, c \) and \( e \) are 1, 2 and 3 respectively. The labels of the merge vertices \( m \) and \( o \) are both 3. Thus, the level of the diamond \( [e, m] \) is 0 and the level of the diamond \( [c, o] \) is 1. The algorithm finds four diamonds with the following levels:

- Level \( (e, m) = 0 \)
- Level \( (g, n) = 0 \)
- Level \( (c, o) = 1 \)
- Level \( (b, p) = 2 \)

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path. Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows
Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex \( a \) is 0. The labels of the subsequent branch vertices \( b, c, e \) are 1, 2 and 3 respectively. The labels of the merge vertices \( m, o \) are both 3. Thus, the level of the diamond \( \{e, m\} \) is 0 and the level of the diamond \( \{c, o\} \) is 1. The algorithm finds four diamonds with the following levels:

- Level \( \{e, m\} = 0 \)
- Level \( \{g, n\} = 0 \)
- Level \( \{c, o\} = 1 \)
- Level \( \{b, p\} = 2 \)

### 3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows...
Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

- \text{level}(\{e, m\}) = 0
- \text{level}(\{g, n\}) = 0
- \text{level}(\{c, o\}) = 1
- \text{level}(\{b, p\}) = 2

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The labels of the subsequent branch vertices \(b, c, e\) are 1, 2, and 3 respectively. The labels of the merge vertices \(m, o\) are both 3. Thus, the level of the diamond \(\{e, m\}\) is 0 and the level of the diamond \(\{c, o\}\) is 1. The algorithm finds four diamonds with the following levels:

- \text{level}(\{e, m\}) = 0
- \text{level}(\{g, n\}) = 0
- \text{level}(\{c, o\}) = 1
- \text{level}(\{b, p\}) = 2

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows...
Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex $a$ is 0. The labels of the subsequent branch vertices $b, c$, and $e$ are 1, 2, and 3 respectively. The labels of the merge vertices $m$ and $o$ are both 3. Thus, the level of the diamond $(c, o)$ is 1 and the level of the diamond $(e, m)$ is 0. The algorithm finds four diamonds with the following levels:

- Level $(c, o) = 1$
- Level $(e, m) = 0$
- Level $(g, n) = 0$
- Level $(b, p) = 2$

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path. Accordingly, there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows
Figure 3
An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

- level \((\text{È} e, m \text{Í}) = 0\)
- level \((\text{È} g, n \text{Í}) = 0\)
- level \((\text{È} c, o \text{Í}) = 1\)
- level \((\text{È} b, p \text{Í}) = 2\)

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The labels of the subsequent branch vertices \(b, c, e\) are 1, 2, and 3 respectively. The labels of the merge vertices \(m, o\) are both 3. Thus, the level of the diamond \((\text{È} e, m \text{Í})\) is 0 and the level of the diamond \((\text{È} c, o \text{Í})\) is 1. The algorithm finds four diamonds with the following levels:

- level \((\text{È} e, m \text{Í}) = 0\)
- level \((\text{È} g, n \text{Í}) = 0\)
- level \((\text{È} c, o \text{Í}) = 1\)
- level \((\text{È} b, p \text{Í}) = 2\)

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path. Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows
Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex $a$ is 0. The labels of the subsequent branch vertices $b$, $c$ and $e$ are 1, 2 and 3 respectively. The labels of the merge vertices $m$ and $o$ are both 3. Thus, the level of the diamond $\{e, m\}$ is 0, and the level of the diamond $\{c, o\}$ is 1. The algorithm finds four diamonds with the following levels:

- $\{e, m\}$ level = 0
- $\{g, n\}$ level = 0
- $\{c, o\}$ level = 1
- $\{b, p\}$ level = 2

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.
Figure 3
An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2.

The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

level \((\bar{e}, m)\) = 0,
level \((\bar{g}, n)\) = 0,
level \((\bar{c}, o)\) = 1,
level \((\bar{b}, p)\) = 2.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex \(a\) is 0. The labels of the subsequent branch vertices \(b, c, e\) are 1, 2, and 3 respectively. The labels of the merge vertices \(m, o\) are both 3. Thus, the level of the diamond \((\bar{e}, m)\) is 0 and the level of the diamond \((\bar{c}, o)\) is 1. The algorithm finds four diamonds with the following levels:

level \((\bar{e}, m)\) = 0,
level \((\bar{g}, n)\) = 0,
level \((\bar{c}, o)\) = 1,
level \((\bar{b}, p)\) = 2.

3.3 Decomposing Diamonds into Simple Paths
In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows...
Figure 3
An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2.
The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

- level \((\bar{E}, m)\) = 0,
- level \((\bar{E}, g)\) = 0,
- level \((\bar{E}, c)\) = 1,
- level \((\bar{E}, b)\) = 2.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2.
The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex \(a\) is 0.
The labels of the subsequent branch vertices \(b\), \(c\) and \(e\) are 1, 2 and 3 respectively. The labels of the merge vertices \(m\) and \(o\) are both 3. Thus, the level of the diamond \((\bar{E}, m)\) is 0 and the level of the diamond \((\bar{E}, c)\) is 1. The algorithm finds four diamonds with the following levels:

- level \((\bar{E}, m)\) = 0,
- level \((\bar{E}, g)\) = 0,
- level \((\bar{E}, c)\) = 1,
- level \((\bar{E}, b)\) = 2.

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows...
3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly, there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows...
Diamonds

Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex is 0. The labels of the subsequent branch vertices b, c, and e are 1, 2, and 3 respectively. The labels of the merge vertices m and o are both 3. Thus, the level of the diamond \(\langle e, m \rangle\) is 0 and the level of the diamond \(\langle c, o \rangle\) is 1. The algorithm finds four diamonds with the following levels:

- \(\text{level}(\langle e, m \rangle) = 0\)
- \(\text{level}(\langle g, n \rangle) = 0\)
- \(\text{level}(\langle c, o \rangle) = 1\)
- \(\text{level}(\langle b, p \rangle) = 2\)

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows
Figure 3. An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The calculated label of each vertex is shown close to it and the numbers with the dark background show the level of the enclosing diamonds. The label of the start vertex $a$ is 0. The labels of the subsequent branch vertices $b$, $c$, and $e$ are 1, 2, and 3 respectively. The labels of the merge vertices $m$ and $o$ are both 3. Thus, the level of the diamond $(e, m)$ is 0 and the level of the diamond $(c, o)$ is 1. The algorithm finds four diamonds with the following levels:

- Level $(e, m)$ = 0
- Level $(g, n)$ = 0
- Level $(c, o)$ = 1
- Level $(b, p)$ = 2

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly, there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows...
Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows
Figure 3: An example execution of Algorithm 2 that finds diamonds on the graph in Figure 2. The calculated label of each vertex is shown close to it. Four diamonds are found. The numbers with the dark background show the level of the enclosing diamonds.

The difference of the labels of the merge and branch vertices. When a diamond is found, it is added to a priority queue based on its level. Finally, the priority queue is returned.

As an example, Figure 3 shows the execution of Algorithm 2 on the graph in Figure 2. The labels of the subsequent branch vertices $b$, $c$, and $e$ are 1, 2, and 3, respectively. The labels of the merge vertices $m$ and $o$ are both 3. Thus, the level of the diamond $\{e, m\}$ is 0, and the level of the diamond $\{c, o\}$ is 1. The algorithm finds four diamonds with the following levels:

- $\{e, m\}$ at level 0
- $\{g, n\}$ at level 0
- $\{c, o\}$ at level 1
- $\{b, p\}$ at level 2

3.3 Decomposing Diamonds into Simple Paths

In this step, we present an algorithm (Algorithm 3) that decomposes each diamond into simple paths and finds the optimum fencing for them. The algorithm iterates the diamonds from the innermost to the outermost. For each diamond, it incrementally extracts simple paths until only a simple path remains in the diamond. Therefore, the degree of the nesting diamond decreases from one to zero. This makes the nesting diamond a simple diamond. As the algorithm iterates all the nested diamonds before the nesting one, diamonds are visited when they are already simple.

For each path of a diamond, the algorithm calls the fence insertion algorithm for simple paths (that we will see in Algorithm 4) to obtain an optimum fence placement for the internal constraints of the path. The rationale for the separation of paths is that the internal constraints of a path can be covered by only fences inside the path. Thus, the optimum fencing for the internal constraints can be locally determined. The algorithm then checks if the resulting fence placement can cover the spanning constraints of the path.

Accordingly, there are two path types: absorbing and emitting. We use an example in Figure 4 to illustrate these types. Figure 4.(a) shows a simple diamond. Figure 4.(b) shows...
Decomposing Diamonds to Simple Paths

(a) The graph of Figure 2 after constraint elimination.

(b) Decomposing diamonds of level 0

(c) Decomposing diamonds of level 1

Figure 5

Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6

The Final Set of Simple Paths for Figure 5.

Figure 7

Fence insertion for a simple path. The algorithm visits the constraints in the order \( a, c \), \( b, d \), \( c, e \), \( f, i \), \( g, i \), and \( h, i \) and inserts the fences \( b, c \), \( d, e \), and \( h, i \). The inserted fences cover the spanning constraint starting from \( g \).

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
Algorithm 3: Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion (q)
2: \( q \) is the minimum priority queue ordering diamonds by level
3: Initialize the set \( F \) to \( \emptyset \).
4: while \( q \) is not empty do
5: \( d \) is the innermost diamond from \( q \).
6: while there is more than one path in \( d \) do
7: \( p \) is a path in \( d \).
8: Call Algorithm 4 on \( p \) to find the fencing \( f \) and the type \( t \).
9: Add \( f \) to \( F \).
10: if \( t \) is absorbing then
11: Update the end point of the spanning constraints to the merge point of \( d \).
12: else if \( t \) is emitting then
13: Update the start point of the spanning constraints to the merge point of \( d \).
14: Remove \( p \) from \( d \).
15: return \( F \).

Figure 4.

Transformation of spanning constraints.

Diamond Decomposition: Absorbing and Emitting Paths

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are \( e,m \) and \( g,n \). For the diamond \( e,m \), the left simple path \( ehikm \) is extracted. The constraint \( k,p \) is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge \( i,k \) that does not cover the spanning constraint \( k,p \). So the path is emitting and the constraint \( k,p \) is shrunk to \( m,p \). Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is \( g,n \). The left edge \( g,n \) with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a). The only diamond of the next level is \( c,o \). There are no spanning constraints and simply extracting the right path \( c,lo \) reduces the diamond to a simple path. Figure 5.(c) has only...
Diamond Decomposition: Absorbing and Emitting Paths

Algorithm 3

Decomposing Diamonds into a Set of Simple Paths

1:
procedure FenceInsertion (q)
2:
\( q \) is the minimum priority queue ordering diamonds by level
3:
Initialize the set \( F \) to \( \emptyset \).
4:
while \( q \) is not empty do
5:
Extract the innermost diamond \( d \) from \( q \).
6:
while there is more than one path in \( d \) do
7:
pick a path \( p \) in \( d \).
8:
call Algorithm 4 on \( p \) to find the fencing \( f \) and the type \( t \).
9:
add \( f \) to \( F \).
10:
if \( t \) is absorbing then
11:
update the end point of the spanning constraints to the merge point of \( d \).
12:
else
13:
\( t \) is emitting
14:
update the start point of the spanning constraints to the merge point of \( d \).
15:
remove \( p \) from \( d \).
16:
return \( F \).

Figure 4.
Transformation of spanning constraints.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are \( 
\{ e, m \} \) and \( 
\{ g, n \} \). For the diamond \( 
\{ e, m \} \), the left simple path \( e, f, h, k, m \) is extracted. The constraint \( 
\{ k, p \} \) is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge \( 
\{ i, k \} \) that does not cover the spanning constraint \( 
\{ k, p \} \). So the path is emitting and the constraint \( 
\{ k, p \} \) is shrunk to \( 
\{ m, p \} \). Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is \( 
\{ g, n \} \). The left edge \( 
\{ g, n \} \) with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a). The only diamond of the next level is \( 
\{ c, o \} \). There are no spanning constraints and simply extracting the right path \( c, f, l, o \) reduces the diamond to a simple path. Figure 5.(c) has only
Example diamond 1

The diamond (a) after constraint elimination. The path $bceg$ is absorbing. The path $bdfg$ is emitting.

(b) The diamond (b) after the transformation of the spanning constraints. The constraints $e, h, e, g$ are updated to $e, g, g, h$.

Figure 4. Transformation of spanning constraints.

Algorithm 3: Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion ($q$)
2: $q$ is the minimum priority queue ordering diamonds by level
3: Initialize the set $F$ to $\emptyset$.
4: while $q$ is not empty do
5: Extract the innermost diamond $d$ from $q$.
6: while there is more than one path in $d$ do
7: Pick a path $p$ in $d$.
8: Call Algorithm 4 on $p$ to find the fencing $f$ and the type $t$.
9: Add $f$ to $F$.
10: if $t$ is absorbing then
11: Update the end point of the spanning constraints to the merge point of $d$.
12: else
13: Update the start point of the spanning constraints to the merge point of $d$.
14: Remove $p$ from $d$.
15: end if
16: return $F$

Figure 4.(b), the constraint $f, h$ is updated to $g, h$. We note that the transformation leaves the internal and passing constraints unchanged.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are $e, m$ and $g, n$. For the diamond $e, m$, the left simple path $ehikm$ is extracted. The constraint $k, p$ is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge $i, k$ that does not cover the spanning constraint $k, p$. So the path is emitting and the constraint $k, p$ is shrunk to $m, p$. Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is $g, n$. The left edge $g, n$ with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a). The only diamond of the next level is $c, o$. There are no spanning constraints and simply extracting the right path $cflo$ reduces the diamond to a simple path. Figure 5.(c) has only one diamond.
Algorithm 3
Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion (q)
2: \( q \) is the minimum priority queue ordering diamonds by level
3: Initialize the set \( F \) to \( \emptyset \).
4: while \( q \) is not empty do
5: Extract the innermost diamond \( d \) from \( q \).
6: while there is more than one path in \( d \) do
7: Pick a path \( p \) in \( d \).
8: Call Algorithm 4 on \( p \) to find the fencing \( f \) and the type \( t \).
9: Add \( f \) to \( F \).
10: if \( t \) is absorbing then
11: Update the end point of the spanning constraints to the merge point of \( d \).
12: else
13: Update the start point of the spanning constraints to the merge point of \( d \).
14: Remove \( p \) from \( d \).
15: \text{return} \( F \).

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level \( 0 \). The diamonds of level \( 0 \) are \( \langle e, m \rangle \) and \( \langle g, n \rangle \). For the diamond \( \langle e, m \rangle \), the left simple path \( ehikm \) is extracted. The constraint \( \langle k, p \rangle \) is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge \( \langle i, k \rangle \) that does not cover the spanning constraint \( \langle k, p \rangle \). So the path is emitting and the constraint \( \langle k, p \rangle \) is shrunk to \( \langle m, p \rangle \). Extracting the left path reduces the diamond to a simple path. The other diamond of level \( 0 \) is \( \langle g, n \rangle \). The left edge \( \langle g, n \rangle \) with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level \( 1 \) in the original graph of Figure 5.(a). The only diamond of the next level is \( \langle c, o \rangle \). There are no spanning constraints and simply extracting the right path \( cflo \) reduces the diamond to a simple path. Figure 5.(c) has only
Diamond Decomposition: Absorbing and Emitting Paths

(a) Example diamond 1

(b) The diamond (a) after constraint elimination. The path $bceg$ is absorbing. The path $bdfg$ is emitting.

(c) The diamond (b) after the transformation of the spanning constraints. The constraints $\mathcal{E}_{e,h}$ and $\mathcal{E}_{e,g}$ are updated to $\mathcal{E}_{g,h}$.

Figure 4

Algorithm 3: Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion ($q$)
2: $q$ is the minimum priority queue ordering diamonds by level
3: Initialize the set $F$ to $\emptyset$.
4: while $q$ is not empty do
5: Extract the innermost diamond $d$ from $q$.
6: while there is more than one path in $d$ do
7: Pick a path $p$ in $d$.
8: Call Algorithm 4 on $p$ to find the fencing $f$ and the type $t$.
9: Add $f$ to $F$.
10: if $t$ is absorbing then
11: Update the end point of the spanning constraints to the merge point of $d$.
12: else if $t$ is emitting then
13: Update the start point of the spanning constraints to the merge point of $d$.
14: Remove $p$ from $d$.
15: return $F$

Figure 4.(b), the constraint $\mathcal{E}_{f,h}$ is updated to $\mathcal{E}_{g,h}$. We note that the transformation leaves the internal and passing constraints unchanged.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level $0$. The diamonds of level $0$ are $\mathcal{E}_{e,m}$ and $\mathcal{E}_{g,n}$. For the diamond $\mathcal{E}_{e,m}$, the left simple path $ehikm$ is extracted. The constraint $\mathcal{E}_{k,p}$ is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge $\mathcal{E}_{i,k}$ that does not cover the spanning constraint $\mathcal{E}_{k,p}$. So the path is emitting and the constraint $\mathcal{E}_{k,p}$ is shrunk to $\mathcal{E}_{m,p}$. Extracting the left path reduces the diamond to a simple path. The other diamond of level $0$ is $\mathcal{E}_{g,n}$. The left edge $\mathcal{E}_{g,n}$ with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level $1$ in the original graph of Figure 5.(a). The only diamond of the next level is $\mathcal{E}_{c,o}$. There are no spanning constraints and simply extracting the right path $cflo$ reduces the diamond to a simple path. Figure 5.(c) has only...
Diamond Decomposition: Absorbing and Emitting Paths

Example diamond 1

The diamond (a) after constraint elimination. The path bceg is absorbing. The path bdhg is emitting.

The diamond (b) after the transformation of the spanning constraints. The constraints f,e, h and g,e, h are updated to g, h and e, g.

Figure 4

Algorithm 3

Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion (q)
2: \( q \) is the minimum priority queue ordering diamonds by level
3: Initialize the set \( F \) to \( \phi \).
4: while \( q \) is not empty do
5: Extract the innermost diamond \( d \) from \( q \).
6: while there is more than one path in \( d \) do
7: Pick a path \( p \) in \( d \).
8: Call Algorithm 4 on \( p \) to find the fencing \( f \) and the type \( t \).
9: Add \( f \) to \( F \).
10: if \( t \) is absorbing then
11: Update the end point of the spanning constraints to the merge point of \( d \).
12: else
13: Update the start point of the spanning constraints to the merge point of \( d \).
14: Remove \( p \) from \( d \).
15: return \( F \).

Figure 4. (b), the constraint f,h is updated to g, h. We note that the transformation leaves the internal and passing constraints unchanged.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level 0. The diamonds of level 0 are e,m and g,n. For the diamond e,m, the left simple path ehikm is extracted. The constraint k,p is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge i,k that does not cover the spanning constraint k,p. So the path is emitting and the constraint k,p is shrunk to m,p. Extracting the left path reduces the diamond to a simple path. The other diamond of level 0 is g,n. The left edge g,n with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level 1 in the original graph of Figure 5.(a).

The only diamond of the next level is c,o. There are no spanning constraints and simply extracting the right path cflor reduces the diamond to a simple path. Figure 5.(c) has only
Figure 4. (b), the constraint $\bar{e}, h \bar{I}$ is updated to $\bar{e}, g \bar{I}$. We note that the transformation leaves the internal and passing constraints unchanged.

We illustrate the iteration over diamonds of different levels in Figure 5. Constraint elimination on the graph in Figure 2 results in Figure 5.(a). Figure 5.(b) shows the result of Algorithm 3 on Figure 5.(a) after processing the diamonds of level $0$. The diamonds of level $0$ are $\bar{e}, m \bar{I}$ and $\bar{g}, n \bar{I}$. For the diamond $\bar{e}, m \bar{I}$, the left simple path $ehikm$ is extracted. The constraint $\bar{k}, p \bar{I}$ is a spanning constraint for this path. The optimum fencing for the internal constraints of this path is one fence on the edge $\bar{i}, k \bar{I}$ that does not cover the spanning constraint $\bar{k}, p \bar{I}$. So the path is emitting and the constraint $\bar{k}, p \bar{I}$ is shrunk to $\bar{m}, p \bar{I}$. Extracting the left path reduces the diamond to a simple path. The other diamond of level $0$ is $\bar{g}, n \bar{I}$. The left edge $\bar{g}, n \bar{I}$ with no constraint can be simply extracted to reduce the diamond to a simple path.

Figure 5.(c) shows the result of Algorithm 3 on the graph of Figure 5.(b) after processing the diamonds of the next level, that has been level $1$ in the original graph of Figure 5.(a). The only diamond of the next level is $\bar{c}, o \bar{I}$. There are no spanning constraints and simply extracting the right path $cflo$ reduces the diamond to a simple path. Figure 5.(c) has only

```
Algorithm 3
Decomposing Diamonds into a Set of Simple Paths

1: procedure FenceInsertion (q)
2: q is the minimum priority queue ordering diamonds by level
3: Initialize the set $F$ to $\emptyset$.
4: while $q$ is not empty do
5: Extract the innermost diamond $d$ from $q$.
6: while there is more than one path in $d$ do
7: Pick a path $p$ in $d$.
8: Call Algorithm 4 on $p$ to find the fencing $f$ and the type $t$.
9: Add $f$ to $F$.
10: if $t$ is absorbing then
11: Update the end point of the spanning constraints to the merge point of $d$.
12: else $t$ is emitting
13: Update the start point of the spanning constraints to the merge point of $d$.
14: Remove $p$ from $d$.
15: return $F$
```
The graph of Figure 2 after constraint elimination.

Decomposing diamonds of level 0.

Decomposing diamonds of level 1.

Figure 5 Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6 The Final Set of Simple Paths for Figure 5.

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom...
Diamond Decomposition, Level 0

The graph of Figure 2 after constraint elimination.

Decomposing diamonds of level 0.

Decomposing diamonds of level 1.

Figure 5
Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6
The Final Set of Simple Paths for Figure 5.

Figure 7
Fence insertion for a simple path. The algorithm visits the constraints in the order $\bar{a}, \bar{c}$, $\bar{b}, \bar{d}$, $\bar{c}, \bar{e}$, $\bar{f}, \bar{i}$, $\bar{g}, \bar{i}$, and $\bar{h}, \bar{i}$ and inserts the fences $\bar{b}, \bar{c}$, $\bar{d}, \bar{e}$, and $\bar{h}, \bar{i}$. The inserted fences cover the spanning constraint starting from g.

3.4 Fence Insertion for Simple Paths
Diamond Decomposition, Level 0

(a) The graph of Figure 2 after constraint elimination.

(b) Decomposing diamonds of level 0

(c) Decomposing diamonds of level 1

Figure 5

Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6

The Final Set of Simple Paths for Figure 5.

Figure 7

Fence insertion for a simple path. The algorithm visits the constraints in the order $\overline{a, c}$, $\overline{b, d}$, $\overline{c, e}$, $\overline{f, i}$, $\overline{g, i}$, and $\overline{h, i}$ and inserts the fences $\overline{b, c}$, $\overline{d, e}$, and $\overline{h, i}$. The inserted fences cover the spanning constraint starting from $g$.  

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
Diamond Decomposition, Level 0

(a) The graph of Figure 2 after constraint elimination.
(b) Decomposing diamonds of level 0
(c) Decomposing diamonds of level 1

Figure 5
Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6
The Final Set of Simple Paths for Figure 5.

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
Diamond Decomposition, Level 1

(a) The graph of Figure 2 after constraint elimination.

(b) Decomposing diamonds of level 0

(c) Decomposing diamonds of level 1

Figure 5
Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6
The Final Set of Simple Paths for Figure 5.

Figure 7
Fence insertion for a simple path. The algorithm visits the constraints in the order \( \overline{a, c}, \overline{b, d}, \overline{c, e}, \overline{f, i}, \overline{g, i}, \overline{h, i} \) and inserts the fences \( \overline{b, c}, \overline{d, e}, \overline{h, i} \). The inserted fences cover the spanning constraint starting from \( g \).

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom disk, the algorithm visits the constraints in the order \( a, c \), \( b, d \), \( c, e \), \( f, i \), \( g, i \), and \( h, i \). The inserted fences cover the spanning constraint starting from \( g \).
3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom 

Figure 7

Fence insertion for a simple path. The algorithm visits the constraints in the order $a, c$, $b, d$, $c, e$, $f, i$, $g, i$, and $h, i$, and inserts the fences $b, c$, $d, e$, and $h, i$. The inserted fences cover the spanning constraint starting from $g$. 

3.4 Fence Insertion for Simple Paths
3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom constraint, the algorithm visits the constraints in the order $a, c, b, d, c, e, f, i, g, i, h, i$ and inserts the fences $b, c, d, e, h, i$. The inserted fences cover the spanning constraint starting from $g$. 

Figure 5: Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6: The Final Set of Simple Paths for Figure 5.

Figure 7: Fence insertion for a simple path. The algorithm visits the constraints in the order $a, c, b, d, c, e, f, i, g, i, h, i$ and inserts the fences $b, c, d, e, h, i$. The inserted fences cover the spanning constraint starting from $g$. 

### 3.4 Fence Insertion for Simple Paths

The graph of Figure 2 after constraint elimination. 

Decomposing diamonds of level 0. 

Decomposing diamonds of level 1.
Fence Insertion for Simple Paths

The graph of Figure 2 after constraint elimination.

Decomposing diamonds of level 0.

Decomposing diamonds of level 1.

Figure 5
Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6
The Final Set of Simple Paths for Figure 5.

Figure 7
Fence insertion for a simple path. The algorithm visits the constraints in the order $\mathcal{E} a, c \mathcal{I}$, $\mathcal{E} b, d \mathcal{I}$, $\mathcal{E} c, e \mathcal{I}$, $\mathcal{E} f, i \mathcal{I}$, $\mathcal{E} g, i \mathcal{I}$, and $\mathcal{E} h, i \mathcal{I}$ and inserts the fences $\mathcal{E} b, c \mathcal{I}$, $\mathcal{E} d, e \mathcal{I}$, and $\mathcal{E} h, i \mathcal{I}$. The inserted fences cover the spanning constraint starting from $g$.

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom...
Fence Insertion for Simple Paths

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom constraint (the one being checked), the algorithm operates as follows:

1. It first identifies the bottom constraint and checks if it is a simple path constraint. If it is not, the algorithm moves to the next constraint.
2. The algorithm then checks if the constraint is a simple path constraint and if it is not, it proceeds to the next constraint.
3. If the constraint is a simple path constraint, the algorithm checks if it is the bottom constraint. If it is, the algorithm inserts a fence at the bottom constraint.
4. If the constraint is not the bottom constraint, the algorithm checks if it is a simple path constraint and if it is not, it proceeds to the next constraint.
5. If the constraint is a simple path constraint, the algorithm inserts a fence at the bottom constraint.
6. The algorithm then checks if there are any more constraints to be processed. If there are, the algorithm repeats the process from step 1.

The algorithm visits the constraints in the order of the bottom constraint, the constraints at levels 1, 2, and 3, and inserts fences at the bottom constraint and the constraints at levels 1, 2, and 3. The inserted fences cover the spanning constraint starting from the bottom constraint.
In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
Fence Insertion for Simple Paths

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom...

The graph of Figure 2 after constraint elimination.

Decomposing diamonds of level 0

Decomposing diamonds of level 1

Figure 5

Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6

The Final Set of Simple Paths for Figure 5.

Figure 7

Fence insertion for a simple path. The algorithm visits the constraints in the order a, c, b, d, c, e, f, i, g, i, h, i and inserts the fences b, c, d, e, and h, i. The inserted fences cover the spanning constraint starting from g.
Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom

3.4 Fence Insertion for Simple Paths

The graph of Figure 2 after constraint elimination.

Decomposing diamonds of level 0

Decomposing diamonds of level 1

Figure 5

Decomposition of Nested Diamonds into a Set of Simple Paths

Figure 6

The Final Set of Simple Paths for Figure 5.

Figure 7

Fence insertion for a simple path. The algorithm visits the constraints in the order $\overline{a, c}, \overline{b, d}, \overline{c, e}, \overline{f, i}, \overline{g, i},$ and $\overline{h, i}$ and inserts the fences $\overline{b, c}, \overline{d, e},$ and $\overline{h, i}$. The inserted fences cover the spanning constraint starting from $g$. 

Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
Fence Insertion for Simple Paths

3.4 Fence Insertion for Simple Paths

In this section, we present an algorithm (Algorithm 4) that finds the optimum fencing for simple paths. More precisely, given the internal constraints of a simple path and a bottom
Optimality and Complexity

1. Decomposing Diamonds
   1. Absorbing path: The spanning constraint can be covered with no extra fence in the path.
   2. Emitting: If the extra fence is put outside the path, it may cover other overlapping constraints.

2. Simple Paths
   1. The size of the optimum solution is at least the size of every set of non-overlapping constraints.
   2. The constraints that lead to addition of fences are non-overlapping.

\[ O(|C| \log |C| + |C||V| + |V|\log|V|) \] time and \[ O(|C| + |V|) \] space complexity.
We now sum the complexity of the steps. The time complexity of Algorithm 1 is $O(|V| + 3|E| \log |E| + |C| \log |C| + |C| |E|)$. The space complexity of Algorithm 1 is $O(|C| + |E| + |V|)$. To further simplify these orders, we show that $|E| \sim O(|V|)$. It is easy to see that the sum of the degree of all the merge vertices of an AFG is $O(|V|)$. Similarly, the sum of the degrees of all the branch vertices of an AFG is $O(|V|)$. Also, the sum of the degrees of all non-merge non-branch vertices is $O(|V|)$. As a result, the sum of degrees of all the vertices is $O(|V|)$ thus, $|E| \sim O(|V|)$. Therefore, the time complexity of Algorithm 1 is $O(|V| \log |V| + |C| \log |C| + |C| |E|)$ and its space complexity is $O(|C| + |V|)$. In this section, we present a transformation for loops in a given CFG with loops to an AFG. Therefore, we can reduce fence insertion for any CFG to an AFG and use Algorithm 1 to find an optimal fence insertion.

We illustrate the transformation using an example. Figure 9.(a) shows a CFG with a loop. The vertex $b$ is the branch instruction: it jumps either to the body of loop at the vertex $c$ or out of the loop to vertex $g$. We call the edge $\bar{b}, c$ that jumps from the branch vertex to the loop body, the start edge. The body of the loop is a CFG in general. In this example, it is the simple path $cef$. We call the edge $\bar{f}, b$ that jumps from the end of the loop body back to the branch vertex, the return edge. We call the edge $\bar{b}, g$ that jumps from the branch vertex out of the loop, the exit edge. We now transform the CFG in Figure 9.(a) to the AFG in Figure 9.(b). The graph has two internal constraints in the loop body: $\bar{e}, c$ and $\bar{c}, f$, and the constraint $\bar{f}, h$ from inside the loop body to outside of the loop. The constraint $\bar{e}, c$ is upwards: it requires the execution of $e$ in one iteration of the loop to be executed before the execution of $c$ in the next iteration of the loop. We notice that the branch instruction $b$ is executed between...
We now sum the complexity of the steps. The time complexity of Algorithm 1 is $O(|V| + 3|E| \log |E| + |C| \log |C| + ||E||)$. The space complexity of Algorithm 1 is $O(|C| + |E| + |V|)$. To further simplify these orders, we show that $|E| = O(|V|)$.

It is easy to see that the sum of the degree of all the merge vertices of an AFG is $O(|V|)$. Similarly, the sum of the degrees of all the branch vertices of an AFG is $O(|V|)$. Also, the sum of the degrees of all non-merge non-branch vertices is $O(|V|)$. As a result, the sum of degrees of all the vertices is $O(|V|)$ thus, $|E| = O(|V|)$.

Therefore, the time complexity of Algorithm 1 is $O(|V| \log |V| + |C| \log |C| + ||C|| |V|)$ and its space complexity is $O(|C| + |V|)$.

In this section, we present a transformation for loops in a given CFG with loops to an AFG. Therefore, we can reduce fence insertion for any CFG to an AFG and use Algorithm 1 to find an optimal fence insertion.

We illustrate the transformation using an example. Figure 9.(a) shows a CFG with a loop. The vertex $b$ is the branch instruction: it jumps either to the body of loop at the vertex $c$ or out of the loop to vertex $g$. We call the edge $\langle b, c \rangle$ that jumps from the branch vertex to the loop body, the start edge. The body of the loop is a CFG in general. In this example, it is the simple path $cfe$. We call the edge $\langle f, b \rangle$ that jumps from the end of the loop body back to the branch vertex, the return edge. We call the edge $\langle b, g \rangle$ that jumps from the branch vertex out of the loop, the exit edge.

We now transform the CFG in Figure 9.(a) to the AFG in Figure 9.(b). The graph has two internal constraints in the loop body: $\langle e, c \rangle$ and $\langle c, f \rangle$, and the constraint $\langle f, h \rangle$ from inside the loop body to outside of the loop. The constraint $\langle e, c \rangle$ is upwards: it requires the execution of $e$ in one iteration of the loop to be executed before the execution of $c$ in the next iteration of the loop. We notice that the branch instruction $b$ is executed between...
Converting a loop to a diamond

We now sum the complexity of the steps. The time complexity of Algorithm 1 is $O(|V| + 3|E| \log |E| + |C| \log |C| + ||E||)$. The space complexity of Algorithm 1 is $O(|C| + |E| + |V|)$. To further simplify these orders, we show that $|E| \sim O(|V|)$. It is easy to see that the sum of the degree of all the merge vertices of an AFG is $O(|V|)$. Similarly, the sum of the degrees of all the branch vertices of an AFG is $O(|V|)$. Also, the sum of the degrees of all non-merge non-branch vertices is $O(|V|)$. As a result, the sum of degrees of all the vertices is $O(|V|)$, thus, $|E| \sim O(|V|)$. Therefore, the time complexity of Algorithm 1 is $O(|V| \log |V| + |C| \log |C| + ||E||)$ and its space complexity is $O(|C| + |V|)$.
Converting a loop to a diamond

We now sum the complexity of the steps. The time complexity of Algorithm 1 is $O(|V| + 3|E| \log |E| + |C| \log |C| + |C||E|)$. The space complexity of Algorithm 1 is $O(|C| + |E| + |V|)$. To further simplify these orders, we show that $|E| \leq O(|V|)$. It is easy to see that the sum of the degree of all the merge vertices of an AFG is $O(|V|)$. Similarly, the sum of the degrees of all the branch vertices of an AFG is $O(|V|)$. Also, the sum of the degrees of all non-merge non-branch vertices is $O(|V|)$. As a result, the sum of degrees of all the vertices is $O(|V|)$, thus, $|E| \leq O(|V|)$. Therefore, the time complexity of Algorithm 1 is $O(|V| \log |V| + |C| \log |C| + |C||E|)$ and its space complexity is $O(|C| + |V|)$. 

In this section, we present a transformation for loops in a given CFG with loops to an AFG. Therefore, we can reduce fence insertion for any CFG to an AFG and use Algorithm 1 to find an optimal fence insertion.

We illustrate the transformation using an example. Figure 9.(a) shows a CFG with a loop. The vertex $b$ is the branch instruction: it jumps either to the body of loop at the vertex $c$ or out of the loop to vertex $g$. We call the edge $(b, c)$ that jumps from the branch vertex to the loop body, the start edge. The body of the loop is a CFG in general. In this example, it is the simple path $cefb$. We call the edge $(f, b)$ that jumps from the end of the loop body back to the branch vertex, the return edge. We call the edge $(b, g)$ that jumps from the branch vertex out of the loop, the exit edge.

We now transform the CFG in Figure 9.(a) to the AFG in Figure 9.(b). The graph has two internal constraints in the loop body: $(e, c)$ and $(c, f)$, and the constraint $(f, h)$ from inside the loop body to outside of the loop. The constraint $(e, c)$ is upwards: it requires the execution of $e$ in one iteration of the loop to be executed before the execution of $c$ in the next iteration of the loop. We notice that the branch instruction $b$ is executed between
Converting a loop to a diamond

We now sum the complexity of the steps. The time complexity of Algorithm 1 is $O(|V| + 3|E| \log |E| + |C| \log |C| + ||E||)$. The space complexity of Algorithm 1 is $O(|C| + |E| + |V|)$. To further simplify these orders, we show that $|E| \sim O(|V|)$. It is easy to see that the sum of the degree of all the merge vertices of an AFG is $O(|V|)$. Similarly, the sum of the degrees of all the branch vertices of an AFG is $O(|V|)$. Also, the sum of the degrees of all non-merge non-branch vertices is $O(|V|)$. As a result, the sum of degrees of all the vertices is $O(|V|)$ thus, $|E| \sim O(|V|)$. Therefore, the time complexity of Algorithm 1 is $O(|V| \log |V| + |C| \log |C| + ||E||)$ and its space complexity is $O(|C| + |V|)$.

In this section, we present a transformation for loops in a given CFG with loops to an AFG. Therefore, we can reduce fence insertion for any CFG to an AFG and use Algorithm 1 to find an optimal fence insertion.

We illustrate the transformation using an example. Figure 9.(a) shows a CFG with a loop. The vertex $b$ is the branch instruction: it jumps either to the body of loop at the vertex $c$ or out of the loop to vertex $g$. We call the edge $\langle b, c \rangle$ that jumps from the branch vertex to the loop body, the start edge. The body of the loop is a CFG in general. In this example, it is the simple path $\langle c, e, f \rangle$. We call the edge $\langle f, b \rangle$ that jumps from the end of the loop body back to the branch vertex, the return edge. We call the edge $\langle b, g \rangle$ that jumps from the branch vertex out of the loop, the exit edge.

We now transform the CFG in Figure 9.(a) to the AFG in Figure 9.(b). The graph has two internal constraints in the loop body: $\langle e, c \rangle$ and $\langle c, f \rangle$, and the constraint $\langle f, h \rangle$ from inside the loop body to outside of the loop. The constraint $\langle e, c \rangle$ is upwards: it requires the execution of $e$ in one iteration of the loop to be executed before the execution of $c$ in the next iteration of the loop. We notice that the branch instruction $b$ is executed between $e$ and $f$. The vertex $h$ is an input/output instruction.
Converting a loop to a diamond

We now sum the complexity of the steps. The time complexity of Algorithm 1 is $O(|V| + 331|E| \log |E| + |C| \log |C| + ||E||)$. The space complexity of Algorithm 1 is $O(|C| + |E| + |V|)$. To further simplify these orders, we show that $|E| \sim O(|V|)$. It is easy to see that the sum of the degree of all the merge vertices of an AFG is $O(|V|)$. Similarly, the sum of the degrees of all the branch vertices of an AFG is $O(|V|)$. Also, the sum of the degrees of all non-merge non-branch vertices is $O(|V|)$. As a result, the sum of degrees of all the vertices is $O(|V|)$ thus, $|E| \sim O(|V|)$. Therefore, the time complexity of Algorithm 1 is $O(|V| \log |V| + |C| \log |C| + ||E||)$ and its space complexity is $O(|C| + |V|)$.  

Fence Insertion for Loops

In this section, we present a transformation for loops in a given CFG with loops to an AFG. Therefore, we can reduce fence insertion for any CFG to an AFG and use Algorithm 1 to find an optimal fence insertion.

We illustrate the transformation using an example. Figure 9.(a) shows a CFG with a loop. The vertex $b$ is the branch instruction: it jumps either to the body of loop at the vertex $c$ or out of the loop to vertex $g$. We call the edge $\langle b, c \rangle$ that jumps from the branch vertex to the loop body, the start edge. The body of the loop is a CFG in general. In this example, it is the simple path $c e f$. We call the edge $\langle f, b \rangle$ that jumps from the end of the loop body back to the branch vertex, the return edge. We call the edge $\langle b, g \rangle$ that jumps from the branch vertex out of the loop, the exit edge.

We now transform the CFG in Figure 9.(a) to the AFG in Figure 9.(b). The graph has two internal constraints in the loop body: $\langle e, c \rangle$ and $\langle c, f \rangle$, and the constraint $\langle f, h \rangle$ from inside the loop body to outside of the loop. The constraint $\langle e, c \rangle$ is upwards: it requires the execution of $e$ in one iteration of the loop to be executed before the execution of $c$ in the next iteration of the loop. We notice that the branch instruction $b$ is executed between
We now transform the CFG in Figure 9.(a) to the AFG in Figure 9.(b). The graph has two internal constraints in the loop body: \(\overline{e, c}\) and \(\overline{c, f}\), and the constraint \(\overline{f, h}\) from inside the loop body to outside of the loop. The constraint \(\overline{e, c}\) is upwards: it requires the execution of \(e\) in one iteration of the loop to be executed before the execution of \(c\) in the next iteration of the loop. We notice that the branch instruction \(b\) is executed between

---

**Figure 8** Reduction Example

**Figure 9** Converting a Loop to a Diamond

We now sum the complexity of the steps. The time complexity of Algorithm 1 is

\[
O(|V| + 3|E| \log |E| + |C| \log |C| + |E||E|).
\]

The space complexity of Algorithm 1 is

\[
O(|C| + |E| + |V|).
\]

To further simplify these orders, we show that

\[
|E| \sim O(|V|).
\]

It is easy to see that the sum of the degree of all the merge vertices of an AFG is \(O(|V|)\). Similarly, the sum of the degrees of all the branch vertices of an AFG is \(O(|V|)\). Also, the sum of the degrees of all non-merge non-branch vertices is \(O(|V|)\). As a result, the sum of degrees of all the vertices is

\[
O(|V|)
\]

thus,

\[
|E| \sim O(|V|).
\]

Therefore, the time complexity of Algorithm 1 is

\[
O(|V| \log |V| + |C| \log |C| + |E||E|)
\]

and its space complexity is

\[
O(|C| + |V|).
\]
Multi-type Fence Insertion Problem is NP-hard

\[
\langle CT, FT, G, C \rangle
\]

\[U = \{u_1, u_2, \ldots, u_n\}\]
\[S = \{S_1, S_2, S_3, \ldots, S_k\}\]

\[CT = U\]
\[FT = S\]
Polynomial-time Fence Insertion For Structured Programs

M. Taheri, A. Pourdamghani, M. Lesani
University of California, Riverside
Sharif University of Technology