Scalable Detection of Semantic Clones

Mark Gabel
Lingxiao Jiang
Zhendong Su

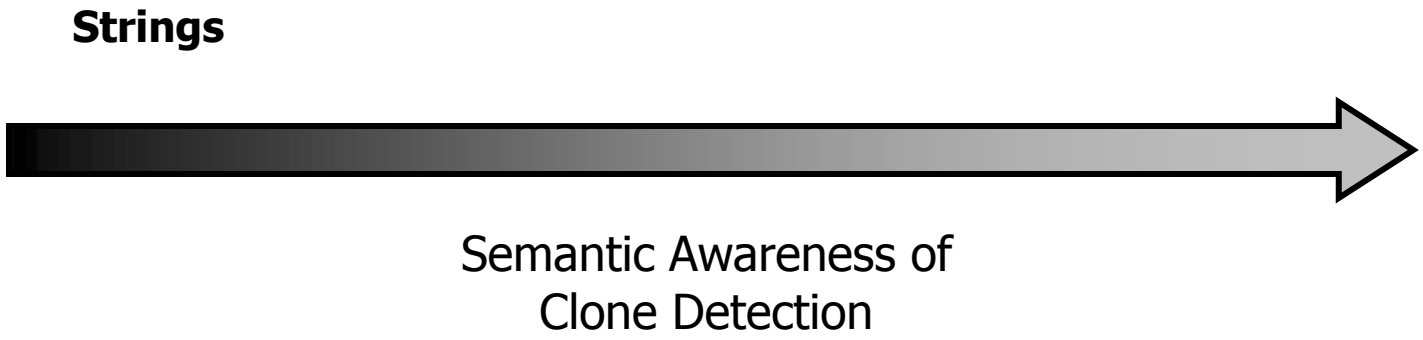
Motivation

- Maintenance problem
  - Refactoring
  - Automated procedure extraction
- Aspect mining
- Program understanding
- Copy/paste bugs

Clone Detection

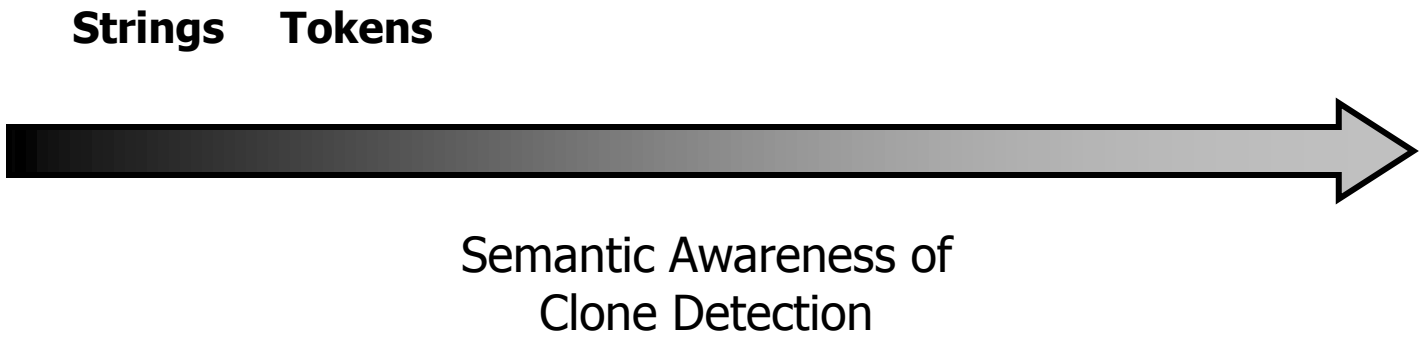
- Definition
  - The enumeration of **similar fragments** of a program or set of programs
- Input:
  - A program or set of programs
- Output:
  - "Clone Groups," sets of equivalent fragments
  - In terms of a **similarity function**

Similarity of Program Fragments



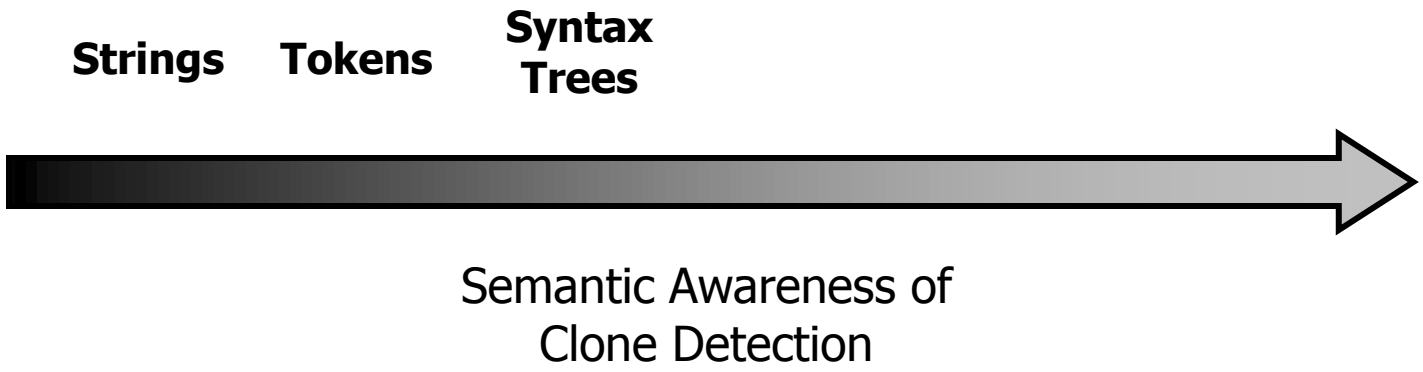
- 1992: Baker, parameterized string algorithm
- Current open source tools: Checkstyle, PMD

Similarity of Program Fragments



- 2002: Kamiya et al., CCFinder
- 2004: Li et al., CP-Miner
- 2007: Basit et al., Repeated Tokens Finder

Similarity of Program Fragments



- 1998: Baxter et al., CloneDR
- 2004: Wahler et al., XML-based
- 2007: Jiang et al., Deckard

Interleaved Clones

```
int func(int i, int j) {
    int k = 10;
    while (i < k) {
        i++;
    }
    j = 2 * k;
    printf("i=%d, j=%d\n", i, j);
    return k;
}
```

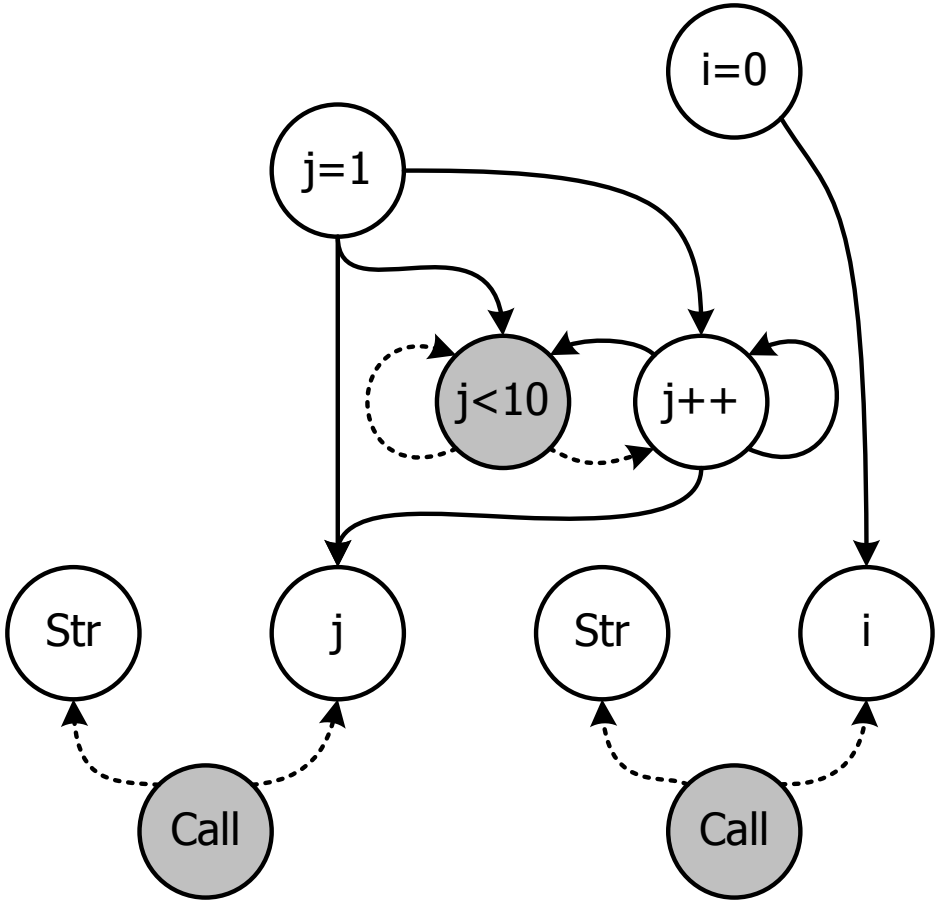
```
int func_timed(int i, int j) {
    int k = 10;
    long start = get_time_millis();
    long finish;
    while (i < k) {
        i++;
    }
    finish = get_time_millis();
    printf("loop took %dms\n",
        finish – start);
    j = 2 * k;
    printf("i=%d, j=%d\n", i, j);
    return k;
}
```

Clones:

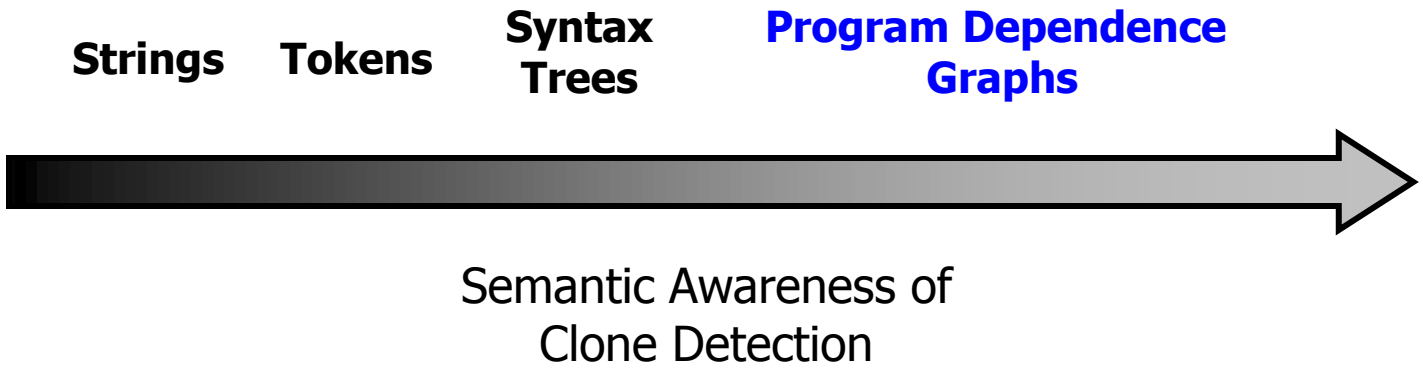
Separate Computations

Program Dependence Graphs

```
void bar() {
  int j = 1;
  int i = 0;
  while (j < 10)
    j++;
  printf("%d", i);
  printf("%d", j);
}
```

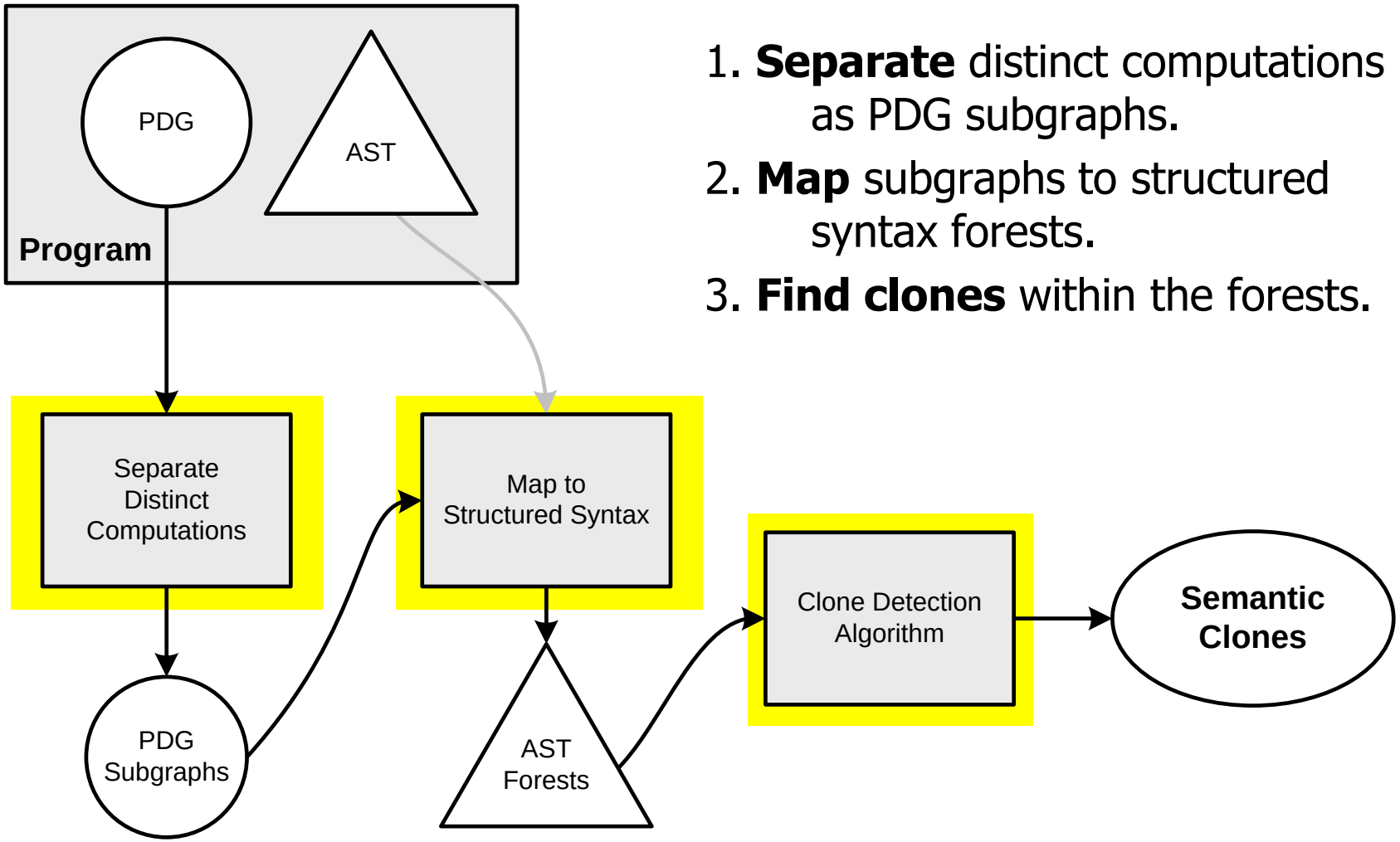


Similarity of Program Fragments



- 2000, 2001: Komondoor and Horwitz
- 2006: Liu et al., GPLAG
- **This work – first scalable technique**

Approach

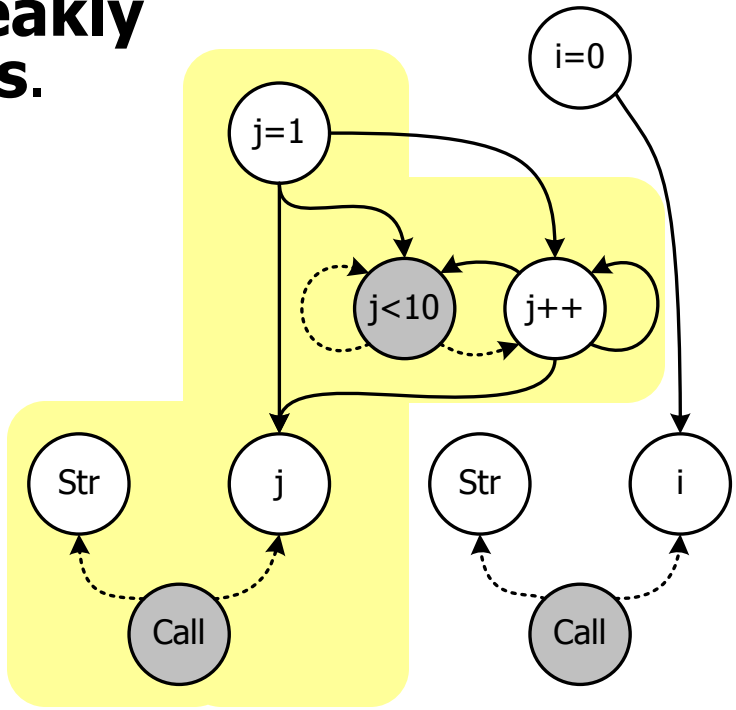


1. **Separate** distinct computations as PDG subgraphs.
2. **Map** subgraphs to structured syntax forests.
3. **Find clones** within the forests.

Separating Computations

- Connected vertices have a semantic relationship
- Break implicit control dependences and partition the PDG into **weakly connected components**.

```
void bar() {
  int j = 1;
  int i = 0;
  while (j < 10)
    j++;
  printf("%d", i);
  printf("%d", j);
}
```



Semantic Threads

```
struct file_stat *compute_statistics() {
    struct file_stat *result = malloc(sizeof(struct file_stat));
    int avg_temp_file_size = 0;
    int avg_data_file_size = 0;
    /* iterate the temp files */
    ...
    /* iterate the data files */
    ...
    /* avg results and store in avg_temp_file_size */
    ...
    /* avg results and store in avg_data_file_size */
    ...
    result->temp_size = avg_temp_file_size;
    result->data_size = avg_data_file_size;
    return result;
}
```

Semantic Threads

```
int count_list_nodes(struct list_node *head) {
    int i = 0;
    struct list_node *tail = head->prev;
    while (head != tail && i < MAX) {
        i++;
        head = head->next;
    }
    return i;
}
```

Enumerating Semantic Threads

- **Semantic thread**:
  - Forward slice or union of forward slices
- **Interesting semantic threads**:
  - Overlap by at most γ nodes
  - Set of maximal size
  - No fully subsumed threads

Semantic Threads in Practice

	Procedures	**Procs** w/ interleaved γ=**0 STs**	**Procs** w/ interleaved γ=**3 STs**
GIMP	13,337	903	3,008
GTK	13,284	697	2,380
MySQL	14,408	1,618	2,441
Postgres	9,276	1,221	2,267
Linux	136,480	10,609	22,514

Mapping and Solving

- Syntactic Image: $\mu : G \rightarrow \{ AST \}$
  - Interesting Semantic Threads ➔ Interesting AST Forests
- Clone Detection: **DECKARD**
  - Numerical vector approximation of trees
  - Clustering as a near-neighbor problem
  - Scalable solution

Implementation

- PDGs, ASTs
  - Grammatech CodeSurfer: C/C++
- Semantic Threads, Clone Detection
  - Parallel Java
- Clustering
  - MIT Locality Sensitive Hashing (native)

Analysis Times

	Size (MLoc)	PDG Build Time	PDG Dump Time
GIMP	0.78	25m 57s	20m 40s
GTK	0.88	12m 50s	16m 54s
MySQL	1.13	16m 56s	12m 36s
PostgreSQL	0.74	9m 12s	21m 48s
Linux	7.31	296m 1s	241m 4s

	AST Only		AST/PDG	
	VGen	**Cluster**	**VGen**	**Cluster**
GIMP	0m37s	1m11s	0m44s	1m45s
GTK	0m31s	0m57s	0m34s	0m53s
MySQL	0m27s	1m16s	0m29s	1m34s
PostgreSQL	0m40s	1m50s	0m51s	2m30s
Linux	8m42s	6m1s	9m48s	7m24s

Quantitative Results

Min. Nds.	AST Only	PDG/AST
4	935203	940497
8	350804	354079
14	150694	152484
22	65275	66489
32	30039	30367

(a) Cloned LOC

Min. Nds.	AST Only	PDG/AST
4	160934	170544
8	49003	54761
14	16114	18918
22	5692	7439
32	2295	3446

(b) Num. of Clone Groups

Min. Nds.	AST Only	PDG/AST
4	13.9	14.1
8	15.5	16.2
14	20.8	22.5
22	26.5	30.1
32	31.9	38.9

(c) Avg. Cloned LOC / Group

Example

```
void os_event_free(os_event_t event)
{
   ut_a(event);
   os_fast_mutex_free(&(event->os_mutex));
   ut_a(0 == pthread_cond_destroy(&(event->cond_var)));
            /* Remove from the list of events */
   os_mutex_enter(os_sync_mutex);
   UT_LIST_REMOVE(os_event_list, os_event_list, event);
   os_event_count--;
   os_mutex_exit(os_sync_mutex);
   ut_free(event);
}
```

Example

```
static void os_event_free_internal(os_event_t event)
{
   ut_a(event);
   /* This is to avoid freeing the mutex twice */
   os_fast_mutex_free(&(event->os_mutex));
   ut_a(0 == pthread_cond_destroy(&(event->cond_var)));
   /* Remove from the list of events */
   UT_LIST_REMOVE(os_event_list, os_event_list, event);
   os_event_count--;
   ut_free(event);
}
```

Another Example

```
GimpVectors *vectors = GIMP_VECTORS (item);
GList *list;
GimpMatrix3 matrix;

gimp_transform_matrix_flip (flip_type, axis, &matrix);

gimp_vectors_freeze (vectors);
gimp_image_undo_push_vectors_mod (gimp_item_get_image (
        item), _("Flip Path"), vectors);

for (list = vectors->strokes; list; list = g_list_next (list))
  {
    GimpStroke *stroke = list->data;
    gimp_stroke_transform (stroke, &matrix);
  }
gimp_vectors_thaw (vectors);
```

```
GimpVectors *vectors = GIMP_VECTORS (item);
GimpMatrix3 local_matrix;
GList *list;

gimp_vectors_freeze (vectors);
gimp_image_undo_push_vectors_mod (gimp_item_get_image (
        item), _("Transform Path"), vectors);

local_matrix = *matrix;

if (direction == GIMP_TRANSFORM_BACKWARD)
  gimp_matrix3_invert (&local_matrix);

for (list = vectors->strokes; list; list = g_list_next (list))
  {
    GimpStroke *stroke = list->data;
    gimp_stroke_transform (stroke, &local_matrix);
  }
gimp_vectors_thaw (vectors);
```

```
GimpVectors *vectors = GIMP_VECTORS (item);
GList *list;
GimpMatrix3 matrix;
gdouble angle = 0.0;

switch (rotate_type)
  {
  case GIMP_ROTATE_90:
    angle = G_PI_2;
    break;
  case GIMP_ROTATE_180:
    angle = G_PI;
    break;
  case GIMP_ROTATE_270:
    angle = - G_PI_2;
    break;
  }

gimp_transform_matrix_rotate_center (center_x, center_y, angle,
        &matrix);

gimp_vectors_freeze (vectors);
gimp_image_undo_push_vectors_mod (gimp_item_get_image (
        item), _("Rotate Path"), vectors);

for (list = vectors->strokes; list; list = g_list_next (list))
  {
    GimpStroke *stroke = list->data;
    gimp_stroke_transform (stroke, &matrix);
  }
gimp_vectors_thaw (vectors);
```

Fragment 1

```
GimpVectors *vectors = GIMP_VECTORS (item);
GList *list;
GimpMatrix3 matrix;

gimp_transform_matrix_flip (flip_type, axis, &matrix);

gimp_vectors_freeze (vectors);
gimp_image_undo_push_vectors_mod (gimp_item_get_image (
        item), _("Flip Path"), vectors);

for (list = vectors->strokes; list; list = g_list_next (list))
   {
      GimpStroke *stroke = list->data;
      gimp_stroke_transform (stroke, &matrix);
   }
gimp_vectors_thaw (vectors);
```

Fragment 2

```
GimpVectors *vectors = GIMP_VECTORS (item);
GimpMatrix3 local_matrix;
GList *list;

gimp_vectors_freeze (vectors);
gimp_image_undo_push_vectors_mod (gimp_item_get_image (
        item), _("Transform Path"), vectors);

local_matrix = *matrix;

if (direction == GIMP_TRANSFORM_BACKWARD)
   gimp_matrix3_invert (&local_matrix);

for (list = vectors->strokes; list; list = g_list_next (list))
   {
      GimpStroke *stroke = list->data;
      gimp_stroke_transform (stroke, &local_matrix);
   }
gimp_vectors_thaw (vectors);
```

```
GimpVectors *vectors = GIMP_VECTORS (item);
GList *list;
GimpMatrix3 matrix;
gdouble angle = 0.0;

switch (rotate_type)
  {
  case GIMP_ROTATE_90:
    angle = G_PI_2;
    break;
  case GIMP_ROTATE_180:
    angle = G_PI;
    break;
  case GIMP_ROTATE_270:
    angle = – G_PI_2;
    break;
  }

gimp_transform_matrix_rotate_center (center_x, center_y, angle,
        &matrix);

gimp_vectors_freeze (vectors);
gimp_image_undo_push_vectors_mod (gimp_item_get_image (
        item), _("Rotate Path"), vectors);

for (list = vectors->strokes; list; list = g_list_next (list))
  {
    GimpStroke *stroke = list->data;
    gimp_stroke_transform (stroke, &matrix);
  }
gimp_vectors_thaw (vectors);
```

Fragment 3

Summary

- First scalable clone detection algorithm based on PDGs
  - Reduction to a simpler tree-based problem
  - Scalable, effective
- New classes of clones
  - Demonstrated to exist
  - Enabling technology: new applications

Complete PDG