APPLYING A GRAPH TO A GRAPH (GRAPH MULTIPLICATION)

William Bricken
March 1998

Abstract

Using the example of three-coloring an arbitrary graph, this rough and unformatted note explores the idea of using one graph as an operator on another graph to produce a "product graph" which embodies the structure/constraints of each of the component graphs. The example graph to be colored is a tetrahedron. The symmetries of the tetrahedral group are exposed as BM transformations on the structure, when link-pairs are interpreted as bounded forms. To model the problem in logic, links (unordered node pairs) are interpreted as BL forms: the logical/computational form of a link (X Y) is NOR. The structural complexity of maximally connected graphs is counted, as an upper bound for possible logic complexity.

A graph is a collection of links between named-nodes. The running example will be the tetrahedron 3COLORing graph. I'll use boundary concepts throughout (i.e. boundary viewpoint), transcribing between the intuition of coloring a spatial graph structure, the logic of predicates, the NOR-graph, and the boundary transformation tools.

The problem is decomposed into structural (graph linking) and coloring (node properties) components.

This is an example of **applying a graph to a graph**. The color graph is applied to every node of the structure graph to generate a *Cartesian product graph* which incorporates color and structure in a single graph.

The **structure graph** can be read as a spatial description of the object; that is, literally a 2D projection of the actual 3D structure with nodes aligning with points and links aligning with edges. The resulting form can be read as describing the plane faces of the object. Thus the structure graph is defined for all 3D tessellated objects.

The **color graph** has color names as nodes and links defined as "not-3majority", a variation of "paired-not-equal". Link definition is obviously problem specific.

THE SPATIAL PROBLEM

Describe the structure of a tetrahedron.

It has four symmetrical nodes, each connected to the other. There are six links.

Let each link be represented by a bounded pair of nodes (i j)

The symmetry

```
choose[K 2]
```

counts the number of nodes in a maximally connected graph and identifies the structure of the forms.

Explicitly for the tetrahedron, the links are

```
(1 2)(1 3)(1 4)(2 3)(2 4)(3 4)        = choose[4 2] = 6
```

Naming this structure:

```
tet = (1 2)(1 3)(1 4)(2 3)(2 4)(3 4)
```

If we constructed a set of four actual nodes, aligning these links with their nodes would structurally force a tetrahedron shape.

This is also a collection of spatial boundaries which can be read from either side of the outermost container (i.e. the container is either marked or void). The spatial boundary reading is close to a group symmetry approach; we have bounded pairs in all combinations.

We have a set of structures sharing the outer space. Each structure is a pair of nodes sharing the interior space. All pairings are represented.

The structure can be deepened to reduce the variable references:

```
(1 2)(1 3)(1 4)(2 3)(2 4)(3 4)  ==>

[ ([1][2]) ([1][3]) ([1][4]) ([2][3]) ([2][4]) ([3][4]) ]   demorgan

( ((1)(2)) ((1)(3)) ((1)(4))
           ((2)(3)) ((2)(4))
                    ((3)(4)) )

( ((1)(2 3 4))
  ((2)  (3 4))
  ((3)    (4)) )
```

Note the tetrahedral structure in the links.

```
form-of-choose[K 2]  =  demorgan[  form-of-choose[K 3]  ]
```

This demonstration is at the heart of the idea that structural descriptions (i.e. graphs, sets of link-bounds) are 2SAT. Each when framed as 3CNF, they reduce to 2DNF.

[The follow demonstration is visually balanced with redundant logic.]

Original form:

```
            ( ((1)(2)) ((1)(3)) ((1)(4))
              ((2)(1)) ((2)(3)) ((2)(4))
              ((3)(1)) ((3)(2)) ((3)(4)) )
```

Distribute outwards, i.e. *factor* each line:

```
            ( ((1)(2 3 4))
              ((2)(1 3 4))
              ((3)(1 2 4)) )
```

Add contextually void-equivalent symmetric forms:

```
      ( ((1) (2 3 4) (1 2 4) (1 3 4) )
        ((2) (2 3 4) (1 2 4) (1 3 4) )
        ((3) (2 3 4) (1 2 4) (1 3 4) ) )
```

Note here that (1) = (1) (1 2 4) (1 3 4)

Distribute outwards:

```
       (2 3 4) (1 2 4) (1 3 4) (1 2 3)
```

This nested form emphasizes the recursion over choose[n n-1]:

```
  (( (1 2 3) (2 3 4) (1 2 4) (1 3 4) ))

  (( (1 2 3) (4 ((1 2) (1 3) (2 3))) ))                    factor 4

  (( (1 2 3) (4 ((1 2) (3 ((1) (2))))))                    factor 3

  (( (1 2 3) (4 ((1 2) (3 ((1) (2 (( ) (1   ))))))))       factor 2
```

Extruding this result into a second dimension emphasizes the recursive form:

```
  (                                                 )
    ((1 2 3) (4                                   ))
                ((1 2) (3                       ))
                           ((1) (2            ))
                                  (( ) (1   ))
```

A function to generate this form is:

```
(defun make-ch-eg (n)
  (if (= n 0)
    '(( ))
    `(( ,make-list[n-1] (,n ( ,choose[n n-1] ))) )  ))
```

where make-ch-eg[n] = choose[n+1 n]

We can turn it into a recursive function:

```
(defun make-2exclude (n)             ;n>1
  (if (= n 0)
    `(( ) (1))
    `( ,(make-list (1- n)) (,n ,(make-2exclude (1- n))) )  ))
```

We have interpreted **tet** as a spatial structure, as a group symmetry, as a collection of links, and as a boundary structure. Now we read the forms as logic:

tet = (((1 2)(1 3)(1 4)(2 3)(2 4)(3 4)))

(not (and (1 or 2) (1 or 3) (1 or 4) (2 or 3) (2 or 4) (3 or 4)))

It is not the case that every pairing of nodes possesses some arbitrary property. Some pairing does not share this (whatever) property, otherwise the graph would be fully connected.

The bounding collection of link-bounds reads for logic as the conjunction of a set of paired disjunctions. An available meaning for this logical form is "Some property is true for every pair". Another interpretation is "Select one or the other node. I.e.. select one end of each link."

The deMorganized form describes faces rather than edges:

tet = (((2 3 4) (1 2 4) (1 3 4) (1 2 3)))

Some face also does not have this (whatever) property. It is the face that does not contain the property that the pair has.

Note: no structural description is in TAUT. Since all node-names are positive, no reductions can occur. (This does not imply that some links are logically redundant.)

Counting Structural Complexity

Maximal connectivity (K is the number of nodes):

choose[K 2] bounded pairs = K(K-1)/2

The maximal number of variable references for this form is

2*choose[K 2] = K(K-1)

The nested form

(((1)(2 3 4)) ((2)(3 4)) ((3)(4)))

has

(sum[K] - 1) = K*(K+1)/2 - 1 = (K^2 + K - 2)/2

variable references and

(3*(K-1) + 1)

distinction nodes.

For 2ary NOR nodes, each list of m inputs (m>2), requires m-2 gates. For K nodes, this would be

structure + listing + top-level

(3*(K-1) + 1) + sum[K-2] + (K-1)

= (K-2)(K-1)/2 + 3K - 2 + K - 1
= (K^2 - 3K + 2 + 8K - 6)/2
= (K^2 + 5K - 4)/2

If 2AND gates are permitted, the NOR gate count reduces around 2K.

Count Summary for the spatial problem

Maximal graph connectivity:

nodes	K	
links=boundary clauses	choose[K 2]	= K(K-1)/2
variable references flat	2*choose[K 2]	= K(K-1)
variable references deep	(sum[K] - 1)	= (K + 2)(K - 1)/2
distinctions deep	(3*(K-1) + 1)	
depth-if-deep	3	
2ary gates	(K^2 + 5K - 4)/2	

THE COLORING PROBLEM

Describe the exclusion property over nodes.

We use predicates asserted about the coloring of each node {1, 2, 3, 4} with colors {B, G, R} to describe the CNF form of 3COLOR.

The form of exclusion over N nodes (objects, locales, times, etc) is

((X Y) (X Z) (Y Z) ((X)(Y)(Z)))

"Exclusion" is also known as "exactly one" quantification

This form is a bounded collection of boundaries, where

(X Y) (X Z) (Y Z)

again represents links between nodes.

The new term,

((X)(Y)(Z))

can be read as universal quantification over all colors:

For all COLORS , the structure that shares the same space defines the color relation. The new term says that the rest of bounds in the collection pertain to each of the specific labels.

Spatially, we have defined an equilateral triangle and declared that all the nodes on this 3graph are constrained in some way.

K item complete exclusion, a declaration of uniqueness across items, has

choose[K 2] + 1

bounds, with

2*choose[K 2] + K

variable mentions.

For 3COLOR, in general:

all[nodes]: (((B)(G)(R)) (B G) (B R) (G R))

This description says: "Nodes are only one color."

Each node has one of these four clause descriptors. The n-th node is identified by:

```
    ( ((Bn) (Gn) (Rn)) (Bn Gn) (Bn Rn) (Gn Rn) )
      1                2       3       4
```

Counting the Unique Color Constraint

For 3 colors, the "exactly one" color restriction requires

```
      4K  bounds
      9K  variable references
      (3K + 1) + 3K + 1 = 2*(3K + 1)  distinctions
```

where K is the number of nodes. 2-ary gates need cover only the fanout at the top level and within the universal conjunction, so the entire count is:

structure	+	*and*	+	*top*
2*(3K + 1)	+	K	+	2K	=	9K + 2

Count Summary for the Unique Color Restriction

For 3color uniqueness, we need:

nodes	K
links=boundary clauses	4K
variable references flat	9K
variable references deep	9K
distinctions deep	6K + 2
depth-if-deep	3
2ary gates	9K + 2

Four Color Example

The four color extension of the uniqueness property is

```
      ( ((B)(G)(R)(Y)) (B G)(B R)(B Y)(G R)(G Y)(R Y) )
```

Compare this to the structure of the tetrahedron, which has a node symmetry. The pattern abstraction is TRUE for any property:

```
      (               (1 2)(1 3)(1 4)(2 3)(2 4)(3 4) )
```

This exclusion description says: "No pair of nodes share the same property."

Abstract the tetrahedron structure term, **tet**,

tet = (1 2)(1 3)(1 4)(2 3)(2 4)(3 4)

to get

(((1)(2)(3)(4)) tet)

For four colors, the tetrahedron abstraction is included as part of the coloring constraint, it asserts uniqueness of the property for the tetrahedron form.

GRAPH PRODUCT: The Form of 3 Coloring a Graph

To constrain the four-node structural graph with the three-node color constraint, we just place the bound descriptions in the same space. The actual linking of colors and nodes is done at the level of forming a node-color predicate. We construct a variable set which is the cross-product of nodes-by-colors. This predicate is "Node # has color X."

We build the structure description, one for each color.

We build color constraint bounds, one for each node.

```
  (
    (B1 B2)(B1 B3)(B1 B4)(B2 B3)(B2 B4)(B3 B4)
    (G1 G2)(G1 G3)(G1 G4)(G2 G3)(G2 G4)(G3 G4)
    (R1 R2)(R1 R3)(R1 R4)(R2 R3)(R2 R4)(R3 R4)

    ((B1) (G1) (R1)) (B1 G1) (B1 R1) (G1 R1)
    ((B2) (G2) (R2)) (B2 G2) (B2 R2) (G2 R2)
    ((B3) (G3) (R3)) (B3 G3) (B3 R3) (G3 R3)
    ((B4) (G4) (R4)) (B4 G4) (B4 R4) (G4 R4)
  )
```

Count Summary for the graph product

3color constraint on maximally connected K graph:

nodes	K
variables	3*K
links=boundary clauses	3*choose[K 2] + 4K = K(3K + 5)/2
variable references flat	6*choose[K 2] + 9K = 3K(K + 2)
variable references deep	none
distinctions deep	3*(K-1) + 1 + 6K + 2 - 1 = 9K - 1
depth-if-deep	3
2ary gates	(K^2 + 5K - 4)/2 + 9K + 2 - 1 = (K^2 + 23K - 2)/2

Before constructing a deep representation, it is appropriate to examine the minimal structure of the form, which as we know is <void>.

The **first step** fro graph multiplication is to insert the subgraph into each positive bound. This requires counting the types of bounds in the expression.

Given K nodes to color, we have the following count on bounds:

ternary-negative	T	=	K
binary-positive	B	=	3K(K+1)/2
all-bounds	A	=	K(3K + 5)/2

Insert (A-1) context bounds into each of B positive-bounds. The first reduction pass constructs B bound-clauses with (2 + A - 1) forms in each positive-bound.

Each virtual inserted form consists of

$$(A - 1) = K(3K + 5)/2 - 1 = 3K(K + 1)/2$$

bounds and

$$3K(K + 5)/2$$

bounds+2atoms.

The variable count is

virtual - self-inner + self-outer

3K(K + 2) - 2 + 2 = 3K(K+2)

Count Summary fro positive bound reduction

First reduction of positive-bounds, initial form

nodes	K
variables	3K
links=boundary clauses A	K(3K + 5)/2
positive bounds B	3K(K+1)/2
negative bounds T	K
variables each flat positive	3K(K + 2)
variables total	(3K(K+1)/2)*3K(K + 2) = 9K^2*(K+1)(K+2)/2
2ary gates, each positive	(K^2 + 23K - 2)/2 - 1
2ary gates, total	(3K(K+1)/2)*((K^2 + 23K - 2)/2 - 1) + 2K
	= 3K(K-1)(K^2 + 23K - 4)/4 + 2K

On the first INSERT pass, all variables which match the two contextual variables in each positive-bound are erased. The new virtual form is then reduced from the top level of the algorithm. Thus there are B calls with a smaller data structure. The algorithm to this point is called

1 + 3K(K+1)/2

times at most. Any time the algorithm recurs, it is with a smaller structure by at least by one positive clause and by the removal of two variables.

With 3K variables, there are at most 3K/2 recursions of this type for each positive bound, yielding

(3K/2)*3K(K+1)/2 + 1 = 9K^2(K+1)/2 + 1

calls to the top level on the first pass.

If a tautology exists (i.e., if the graph cannot be three colored), the form will rapidly reduce in size. If the graph can be three colored, then failure for a bound-clause to reduce indicates how a particular link must be colored, relative to all other links on the graph.

In the case of **tet**, the

K*choose[3 2] = 4*K

factor vanishes as redundant. This corresponds to links that have the following pattern -- for eg, (B1 G1):

(<letter.number> <different-letter.same-number>)

They are all the positive-bounds associated with the color structure constraint. That is, the color triangle reduces to a universal color assertion: "each of the available colors". The structure of the **tet** graph is redundant with the color triangle-graph.

Note that only color constraint positive-bounds can vanish; erasing links would violate the structure of the graph/problem.

The **second step** is to insert what remains into the negative-bounds.

In the **tet** case, the total context clauses remaining for negative-clause insertion is

3*choose[K 2] + K - 1

In the worst case, the total clauses remaining is all the clauses but one. There are 3K negative literals in the negative-bounds, one for each variable.

Each virtual inserted form consists of

$$(A - 1) = K(3K + 5)/2 - 1 = 3K(K + 1)/2$$

bounds giving the whole single negative literal size of

$$3K(K + 3)/2$$

bounds+1atom. There are still 3K(K+2) variables in each negative-bound.

Thus the variable size for all negative forms, with virtual insertion, is

$$3K*(3K(K+2)) = 9K^2(K+2)$$

This gives a total size of the initial virtual form as

positive-vars		*negative-vars*		*vars*
9K^2*(K+1)(K+2)/2	+	9K^2(K+2)	=	9K^2(K+2)(K+3)/2

Each negative bound shrinks by at least one variable, so there are a maximum of 3K recursions on this form.

Of the

$$3*choose[K\ 2] + K - 1$$

clauses inserted into each negative-literal of **tet**, there are choose[K 2] which erase, all those associated with the color of the negative-literal.

The remaining

$$(2*choose[K\ 2] + K - 1)$$

clauses are a 2SAT TAUT.

Summary of Counting

To determine 3COLOR requires the following resources:

nodes	K
variables	3K
links=bounds A	K(3K + 5)/2
positive bounds B	3K(K+1)/2
negative bounds T	K
variables total (insert)	9K^2(K+2)(K+3)/2
Recursive calls:	9K^2(K+1)/2 + 3K + 1

If each recursive call processed every variable, then the number of tests for variable equality would have a maximum bound of

(9K^2(K+1)/2 + 3K + 1)*(9K^2(K+2)(K+3)/2) = O[K^7]

Smaller Still

Note that redundant positive bounds will be removed during the negative-bound reduction, so in practice, none of the positive reductions need occur until the recursive context of each negative bound insertion. By replicating the work to reduce positive bounds 3K times, the algorithm resources can be reduced by the entire first step, i.e. by

9K^2*(K+1)(K+2)/2 variable references and

9K^2(K+1)/2 + 1   recursive calls

The remaining sizes:

variable references counting inserted virtual forms

9K^2(K+2) + K(K-1) = K*(9K^2 + 19K - 1)

recursive calls

3K

variable equality tests

(3K)*(K*(9K^2 + 19K - 1)) = 3K^2(9K^2 +19K - 1) = O[K^4]

Existence Proof

Prove: Pervasion by Insertion identifies three-coloring-impossibility for an arbitrary graph.

The form consists of at most

K(K-1)/2

positive bounds; these identify the connectivity structure of the graph. An additional

3K positive-bounds and K negative-bounds

identify the colorability constraint.

Only the negative-bounds can vanish to identify a TAUT, and to do so, all of them for a specific node must vanish. Eg: if ((Bi)(Gi)(Ri)) vanishes, then node i is not colorable.

For this to happen, the virtual form inserted into each negative-literal must reduce to mark. The smaller virtual graph to be reduced consists of the original form with these things removed:

1. the universal-color constraint for the node
2. the particular current color of that node, individually, and
3. any redundant clauses in the positive-bounds

Removing the color constraint for a node suggests that the algorithm can be proved by induction on the nodes. This convenience provides a three recurrence halt to the proof.