Report

Description of Algorithm:

My algorithm goes through and generates cost for possible transitions and then employs DFS in order to determine the best possible 2nd order transformation that yields the best solution to the geometric analogy problem.

Firstly, in order to analyze the properties of the various shapes, I run the solution from A1 on the input files, and employ the results from finalList in order to get the various shapes with their attributes (i.e. size, shape, vertices, etc.). Furthermore, I generate an output directory containing the interpretation files for the input files and parse those to get the relations between the various shapes. In order to distinguish between shapes across files and various interpretations, I also change the shape attribute for every object to the name of the file + interpretation + shape number. For instance, shape 1 in file A.txt for the second interpretation would be named Ab1 if a polygon, and Abc1 if circle, or Abd1 if dot. This allows for me to be able to distinguish the various shapes in an easier fashion. From here, I create separate lists of my shapes in A, B, C, and k. I then go through all the possible pairings of shapes in A and B, and C and k, respectively and calculate all 1st order transformation costs by passing the lists into my Generate_Transformations function. My Generate_Transformation function handles the relations for inside, and outside as well as delete/add by placing a place-holder character, *, to represent the shape that is being deleted or added. This function's final result will return back two dictionaries one for A → B transformation costs and for C → k transformation costs, where the key is the transformation and the value the cost.

The next part of my algorithm employs a DFS search through these two dictionaries by starting with the transformations that have the lowest costs. This search is to compare the second order transformation costs that come from the possible transformation pairs of T(T(AB), T(Ck)). The algorithm does this by comparing the difference between the costs for T(A, B) and T(C, k), since this would logically mean that the transformation between the two are very similar if their difference in costs is the least. It is important to note that I do have a cost heuristic for a maximum cost for this algorithm so that at some point it can distinguish between too many transformations that yield a low difference in cost. For instance, it is possible to change every shape in both T(A, B) and T(C, k), which will yield a very high cost for both, but will ultimately be the wrong transformation because it undergoes so many changes.

Finally, my algorithm returns the second order transformation with the lowest cost and outputs the correct k.

Metric on the Complexities of the first order and second order transformations:

Below are my various penalty/cost heuristics and values:
cost_size = 7
cost_delete = 24
cost_add = 29.1
cost_change_shape = 19

cost_area = 7.3
hloc_left_cost = 3.0
hloc_right_cost = 3.1
hloc_center_cost = 3.2
vloc_bottom_cost = 3.3
vloc_top_cost = 3.4
vloc_middle_cost = 3.5
overlap_cost = 6.7
inside_cost = 6.0

MAX_2nd_Order_Transformation_Cost = 67.1

During the course of my algorithm, I went through MANY iterations of adjusting my costs in order to be able to distinguish the best 1st and 2nd order transformations. I selected the above costs based on the logic of changes that happen throughout the geometric analysis problem. It makes logical sense that intrinsic changes to shapes should hold a heavier cost than extrinsic ones. Therefore, changes in shape, area, size cost at least twice as much as changes in top, right, bottom, left. There is also the consideration of relations that matter. I needed to be able to distinguish the difference between an inner or outer or overlapping shape being deleted/added or changed in general. Therefore, relation penalties held greater weight than when a shape was moved around the grid. The cost for deleting and adding shapes was the highest since that is a very large change, and so should have a large cost/penalty. In addition, it was necessary to distinguish costs for each individual hloc and vloc since it is important in some geometric analogy problems to distinguish between the answer choices of a shape moving to the top, left or top, right, for instance. Finally, for my search algorithm I also have a max 2nd order transformation cost that I do not wish to exceed. This was selected based on the summation of my two highest costs (adding a shape and changing the shape). This made logical sense, since I wanted to rule out second order transformation that seem the best only because they undergo a large number of transformations.

<u>Transformation – Test case 1</u>

A →
triangle(Aa1)
vloc(Aa1,middle)
hloc(Aa1,center)
vloc(Aa2,middle)
hloc(Aa2,center)
inside(Aa2,Aa1)
small(Aa2)
large(Aa1)

B →
triangle(Ba1)
vloc(Ba1,middle)
hloc(Ba1,center)

C →
circle(Cac1)
vloc(Cac1,middle)
hloc(Cac1,center)
square(Ca1)
vloc(Ca1,middle)
hloc(Ca1,center)
inside(Ca1,Cac1)
inside(Ca1,Cac1)
small(Ca1)
large(Cac1)

k = 4 →
vloc(c1,middle)
hloc(c1,center)

T(AB) → (Aa1,Ba1)(Aa2,*)
delete(Aa2)
delete inside Aa2

T(Ck) → (Cac1,k4ac1)(Ca1,*)
delete(Ca1)
delete inside Ca1

T(T(AB), T(Ck)) → ((Aa1, Cac1)(Ba1, k4ac1)(Aa2, Ca1))

Transformation – Test Case 2

A →
circle(Aac1)
vloc(Aac1,top)
hloc(Aac1,center)
circle(Aac2)
vloc(Aac2,top)
hloc(Aac2,center)
square(Aa1)
vloc(Aa1,bottom)
hloc(Aa1,center)
inside(Aac2,c1)
small(Aac2)
large(Aac1)
above(Aac1,Aa1)
above(Aac2,Aa1)

B →
circle(Bac1)
vloc(Bac1,bottom)
hloc(Bac1,center)
circle(Bac2)
vloc(Bac2,top)
hloc(Bac2,center)
square(Ba1)
vloc(Ba1,bottom)
hloc(Ba1,center)
below(Bac1,Bac2)
inside(Ba1,Bac1)
inside(Ba1,Bac1)
small(Ba1)
large(Bac1)
above(Bac2,Ba1)

C →
circle(Cac1)
vloc(Cac1,bottom)
hloc(Cac1,center)
triangle(Ca1)
vloc(Ca1,top)
hloc(Ca1,center)
square(Ca2)
vloc(Ca2,top)
hloc(Ca2,center)
below(Cac1,Ca1)

below(Cac1,Ca2)
above(Ca1,Ca2)
inside(Ca2,Ca1)
small(Ca2)
large(Ca1)

k = 3 →
circle(k3ac1)
vloc(k3ac1, bottom)
hloc(k3ac1, center)
triangle(p1)
vloc(k3a1, bottom)
hloc(k3a1, center)
square(k3a2)
vloc(k3a2, top)
hloc(k3a2, center)
inside(k3ac1, k3a1)
small(k3ac1)
large(k3a1)
below(k3ac1, k3a2)
below(k3a1, k3a2)

T(AB) → (Aac1,Bac1)(Aac2,Bac2)(Aa1,Ba1)
move(top(Aac1),bottom(Bac1))
change(inside(['Aac1', 'Bac1']), inside(['Aac2', 'Bac2']))
change(inside(['Aac1', 'Bac1']), inside(['Aa1', 'Ba1']))

T(Ck) → (Cac1,k3ac1)(Ca1,k3a1)(Ca2,k3a2)
move(top(Ca1),bottom(k3a1))
change(inside(['Cac1', 'k3ac1']), inside(['Ca1', 'k3a1']))
change(inside(['Ca1', 'k3a1']), inside(['Ca2', 'k3a2']))

T(T(AB), T(Ck)) →
(Aac1, Cac1)(Bac1, k3ac1)(Aac2, Ca1)(Bac2, k3a1)(Aa1, Ca2)(Ba1, k3a2)

Transformation – Test Case 3

A (Best transformation Ac)→
triangle(Ac1)
vloc(Ac1,top)
hloc(Ac1,right)
rectangle(Ac2)
vloc(Ac2,middle)
hloc(Ac2,left)
right_of(Ac1,Ac2)
above(Ac1,Ac2)

overlap(Ac1,Ac2)

B (Best transformation Ba) →
triangle(Ba1)
vloc(Ba1,bottom)
hloc(Ba1,left)
rectangle(Ba2)
vloc(Ba2,middle)
hloc(Ba2,center)
left_of(Ba1,Ba2)
below(Ba1,Ba2)
inside(Ba1,Ba2)
small(Ba1)
large(Ba2)

C (Best transformation Ca) →
rectangle(Ca1)
vloc(Ca1,top)
hloc(Ca1,right)
scc(Ca2)
vloc(Ca2,middle)
hloc(Ca2,left)
right_of(Ca1,Ca2)
above(Ca1,Ca2)
overlap(Ca1,Ca2)

k = 3 →
scc(k3a1)
vloc(k3a1,middle)
hloc(k3a1,left)
rectangle(k3a2)
vloc(k3a2,bottom)
hloc(k3a2,left)
right_of(k3a1,k3a2)
above(k3a1,k3a2)
inside(k3a2,k3a1)
small(k3a2)
large(k3a1)

T(AB) → (Ac1,Ba1)(Ac2,Ba2)
move(right(Ac1),left(Ba1))
move(top(Ac1),bottom(Ba1))
move(left(Ac2),center(Ba2))
change(overlap(['Ac1', 'Ba1']), inside(['Ac2', 'Ba2']))

T(Ck) → (Ca1,k3a2)(Ca2,k3a1)

move(right(Ca1),left(k3a2))
move(top(Ca1),bottom(k3a2))
change(overlap(['Ca1', 'k3a2']), inside(['Ca2', 'k3a1']))

T(T(AB), T(Ck)) →
((Ac1, Ca1)(Ba1, k3a2)(Ac2, Ca2)(Ba2, k3a1))
move(left(Ac2), left(Ca2))
move(center(Ba2), center(k3a1))

<u>Transformation – Test Case 4</u>

A →
circle(Aac1)
vloc(Aac1,middle)
hloc(Aac1,center)
triangle(Aa1)
vloc(Aa1,middle)
hloc(Aa1,center)
inside(Aac1,Aa1)
small(Aac1)
large(Aa1)

B →
circle(Bac1)
vloc(Bac1,top)
hloc(Bac1,center)
triangle(Ba1)
vloc(Ba1,bottom)
hloc(Ba1,center)
above(Bac1,Ba1)

C →
circle(Cac1)
vloc(Cac1,middle)
hloc(Cac1,center)
square(Ca1)
vloc(Ca1,middle)
hloc(Ca1,center)
inside(Ca1,Cac1)
inside(Ca1,Cac1)
small(Ca1)
large(Cac1)

k = 2 →
circle(k2ac1)
vloc(k2ac1,bottom)

hloc(k2ac1,center)
square(k2a1)
vloc(k2a1,top)
hloc(k2a1,center)
below(k2ac1,k2a1)

T(AB) → (Aac1,Bac1)(Aa1,Ba1)
change(small(Aac1),large(Bac1))
move(middle(Aac1),top(Bac1))
change(large(Aa1),small(Ba1))
move(middle(Aa1),bottom(Ba1))
change(inside(['Aac1', 'Bac1']), inside(['Aa1', 'Ba1']))

T(Ck) → (Cac1,k2ac1)(Ca1,k2a1)
change(large(Cac1),small(k2ac1))
move(middle(Cac1),bottom(k2ac1))
change(small(Ca1),large(k2a1))
move(middle(Ca1),top(k2a1))
change(inside(['Cac1', 'k2ac1']), inside(['Ca1', 'k2a1']))

T(T(AB), T(Ck)) →
(Aac1, Cac1)(Bac1, k2ac1)(Aa1, Ca1)(Ba1, k2a1)

<u>Transformation – Test Case 5</u>

A →
dot(Aad1)
vloc(Aad1,top)
hloc(Aad1,center)
square(Aa1)
vloc(Aa1,bottom)
hloc(Aa1,center)
above(Aad1,Aa1)

B →
square(Ba1)
vloc(Ba1,bottom)
hloc(Ba1,center)

C →
dot(Cad1)
vloc(Cad1,middle)
hloc(Cad1,left)
square(Ca1)
vloc(Ca1,middle)
hloc(Ca1,right)

left_of(Cad1,Ca1)

k = 5 →
square(k5a1)
vloc(k5a1,middle)
hloc(k5a1,right)

T(AB) → (Aad1,*)(Aa1,Ba1)
delete(Aad1)

T(Ck) → (Cad1,*)(Ca1,K5a1)
delete(Cad1)

T(T(AB), T(Ck)) →
(Aad1, Cad1)(Ba1, K5a1)(Aa1, Ca1)