Solving Large Jigsaw Puzzles

L. Dery and C. Fufa

Abstract—This project attempts to reproduce the genetic algorithm in a paper entitled "A Genetic Algorithm-Based Solver for Very Large Puzzles" by D. Sholomon, O. David, and N. Netanyahu. [3] There are two main challenges in solving jigsaw puzzles. The first is finding the right fitness function to judge the compatibility of two pieces. This has thoroughly been studied and as a result, there are many fitness functions available. This paper explores the second part that is crucial to solving jigsaw puzzles: finding an efficient and accurate way to place the pieces. The genetic algorithm attempts to do just that. The crucial part of the algorithm is in generating a new ordering of pieces called 'child' from two possible orderings of pieces, called 'parents'. Each generation learns from good traits in the parents. After going through a hundred generations, the ordering will reflect the original image to a high accuracy. This paper also makes use of CNN to start with reasonable orderings of 'parents'. This cuts down on the number of generations required to reach the correct ordering of the pieces.

Index Terms—CS231A, Jigsaw Puzzles Algorithms

1 Introduction

The problem of automating the solving of jigsaw puzzles is one that has been around since at least the 1950s. Jigsaw puzzles are image reconstruction problems where the image provided has been cut into non overlapping boxes and shuffled around. The problem is then to reconstruct the original image from the shuffled pieces. For the problem to be tractable, the puzzle pieces are assumed to be of identical dimensions and that no piece has been rotated.

The problem has multiple applications, both in and outside of image reconstruction. Puzzle solution techniques can be applied to broken tiles to simulate the reconstruction of archaeological artifacts. In fall, 2011, DARPA held a competition, with a fifty thousand dollar prize, to automatically reconstruct a collection of shredded documents.Other applications include the molecular docking problem for drug design, DNA/RNA modeling, image base CAPTCHA construction, and speech descrambling.

2 Review of previous work

2.1 Previous Work

In 1964, Freeman and Garder[1] proposed a solution for a 9 piece problem. The shapes were allowed to be of different dimensions. After Freeman and Garder, most of the work has been based on color-based solvers, with the assumption that all pieces are rectangles of the same dimension. Recently, probabilistic puzzle solvers have been developed[2]. These algorithms were solving 432 pieces. These solvers however require apriori knowledge of the puzzle. There are also particle filter-based solvers which are improvements over the probabilistic puzzle solvers.

In 2013 Sholomon et Al[3] introduced a genetic algorithm based technique for solving large jigsaw puzzles. It is our goal in this paper to replicate the results of this paper and also suggest areas where it could be improved.

2.2 This paper's contribution

This paper uses the genetic algorithm as a strategy for piece placement. It uses a standard estimation function. While this is not the

first time that the genetic algorithm has been used to solve the jigsaw puzzle problem, it has only been used to solve puzzles of a limited size. This paper attempts to solve puzzles with larger pieces. In addition to the genetic algorithm, this paper also attempts to use CNN to arrive at the correct reconstruction of the image in less iterations.

3 Technical Details

3.1 Genetic Algorithm

The genetic algorithm as implemented for solving the jigsaw puzzle problems starts out with a thousand different ways to order the pieces. Each way of ordering a piece is called a chromosome. The entire set of a thousand chromosomes is called a population. At each stage of the process, called a generation, we have a population of a thousand chromosomes. Now, the goal is that with each passing generation, i.e. with the next thousand chromosomes or a population, the orderings of the pieces will begin to look more and more like the original or correct image. During each population, the best chromosome will be determined by the estimation function.

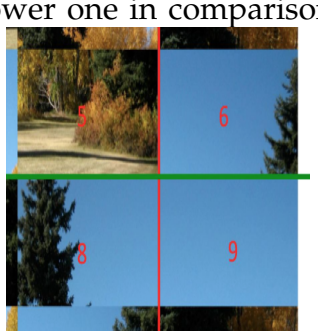
Algorithm 1 Pseudocode of GA framework

```
population ← generate 1000 random chromosomes
for generation_number = 1 → 100 do
    evaluate all chromosomes using the fitness function
    new_population ← NULL
    copy 4 best chromosomes to new_population
    while size(new_population) ≤ 1000 do
        parent1 ← select chromosome
        parent2 ← select chromosome
        child ← crossover(parent1, parent2)
        add child to new_population
    end while
    population ← new_population
end for
```

Above is the higher level pseudo code for the Genetic Algorithm framework. The four best scorers according to the estimation or fitness function will automatically be placed into the next generation. The rest of the chromosomes for the next generation are going to be hybrids of chromosomes from the current one. Two chromosomes from the current population are selected, and a function called crossover generates a child chromosome that learns from the parents, and has a better reordering of the pieces, and hence, a better fitness score. It is via this mechanism that each generation gets a better fitness score than the previous generation. The selection process of which parents to choose to give birth to a new child chromosome discriminates towards parents with a better fitness score. The selection process is called a roulette selection. The likelihood of being selected is directly proportional to how good the fitness score is. This way, the algorithm makes sure that selected parent chromosomes have good traits (as evidenced by their fitness scores) to be passed on to the children.

Fitness Function

The estimation function utilizes the fact that adjacent pieces in the original image will most likely share similar colors along their edges. Hence, computing the sum of the squared color differences along pixels that are adjacent to each other (between two different pieces) will give us an indication of whether the two pieces belong adjacently in the direction they shared the pixels. Hence, the less the specific sum is , the more likely they are to be adjacent to each other. From the image below for example, we can expect the fitness function to give us a high score for piece 5 and 6 as the color difference in the edges seem to be high, while piece 8 and 9 will have a very low fitness score. We can further assume that piece 5 and 8 will have a high fitness score while 6 and 9 will have a lower one in comparison.



The fitness function for a given chromosome will compute the sum of the score for every edge in the chromosome. Below are examples of functions which compute the compatibility

score of two pieces in a left-right adjacency relationship, and a function which computes the fitness score of a given chromosome (i.e. computes the score for all edges and directions.)

$$D(x_i, x_j, r) = \sqrt{\sum_{k=1}^{K}\sum_{b=1}^{3}(x_i(k, K, b) - x_j(k, 1, b))^2}.$$

K is the number of pixels in each piece in the vertical direction.

$$\sum_{i=1}^{N}\sum_{j=1}^{M-1}(D(x_{i,j}, x_{i,j+1}, r)) + \sum_{i=1}^{N-1}\sum_{j=1}^{M}(D(x_{i,j}, x_{i+1,j}, d))$$

This way, it covers all the available edges in a chromosome. Note that D is the fitness score for the compatibility of piece xj to the direction (left, right, down, or up) of xi. In the selection process of the algorithm (roulette selection) make sure that a lower fitness score is treated as more likely to be chosen.

Crossover

Crossover can be considered the heart of the algorithm. Crossover receives two parent chromosomes and creates a child chromosome. It allows ”good traits” to be transmitted from the parents to the child. The goal is to have the child with a better fitness score than both parents. The fitness function does a good job of discriminating between adjacent pieces, but does not give any indication of whether the pieces are placed at the correct absolute position in the image. The implementation of crossover then must allow for independence in the placement of pieces. (It should be a dynamic process. Just because a piece was at some point assigned to say, (2,3) of the the image, it must not remain there, it should be able to transition into a different place based on how the pieces build up around it.)

The implementation of crossover suggested starts out with a single piece and then gradually joins other pieces at available boundaries.The image is always contiguous since new pieces are only added adjacent to existing ones. Keeping track of the pieces used and the dimensions of the child being formed is important so that the dimensions of the child are similar to that of the parents. The process of growing the kernel will go on until all the pieces have been used.

The final absolute location of a given piece is only determined after all the pieces have been used. This is because as recommended earlier, the kernel growing process must allow for independence or flexibility in the placement as the algorithm plays out. To begin, crossover selects a random piece from either parent and places it in the kernel. After that, it keeps track of all the available boundaries for a new piece to be added to the kernel. An available boundary can be thought of as a piece and the direction in which a new piece can be placed adjacent to it. There are three main phases involved in crossover.

Phase One

It goes through the boundary pieces in the kernel. Let’s say that piece x_i in the direction d, for example is selected. Phase one checks to see if both parents have the same piece x_j in the direction of d of x_i. If it so, x_i is added to the kernel. If x_i has been already added, it will of course be skipped. The only pieces under consideration should be unused pieces (pieces not in the kernel). This phase keeps on going until there is no boundary on which both parents agree.

Phase Two

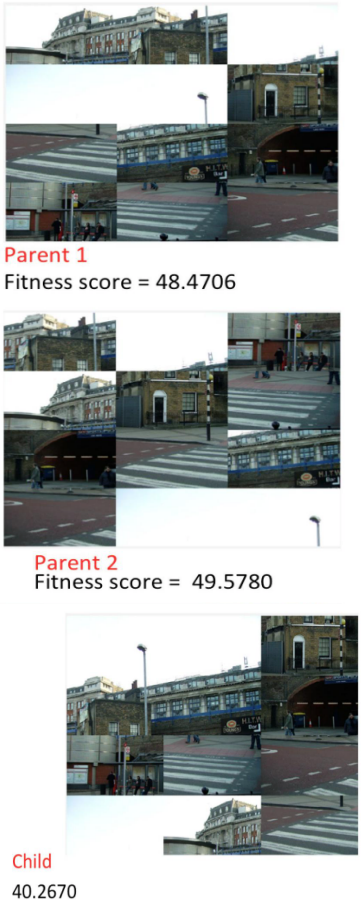
Assume (x_i,R) is available on the kernel. Check if one of the parents contains a piece x_j in spatial relation R of x_j, which is also a best-buddy of xi in that relation. Two pieces x_i and x_j are considered best-buddies if D(x_i,x_j,R) is the lowest fitness score they can achieve.I.e. there is no better piece x_k, that will give a lower fitness score D(x_k,x_j,R) as well as no x_k available, that can give D(x_i,x_k, R) lower than D(x_i,x_j,R). The piece considered x_j must be adjacent to x_i in one of the parents.

If such a piece is found, go back to phase one, if not, proceed to three

Phase Three

Pick random (xi,R) from the kernel and assign it x_j from available pieces such that D(x_i,x_j,R)

is lowest. Go to Phase One. The three phases keep on going until all the pieces are used up. Mutation is introduced in phases one and three. With a 5 percent probability, a random available piece is assigned as opposed to one that both parents agree on in phase one and the most compatible on e in phase three.



3.2 Convolutional Neural Network Augmentation

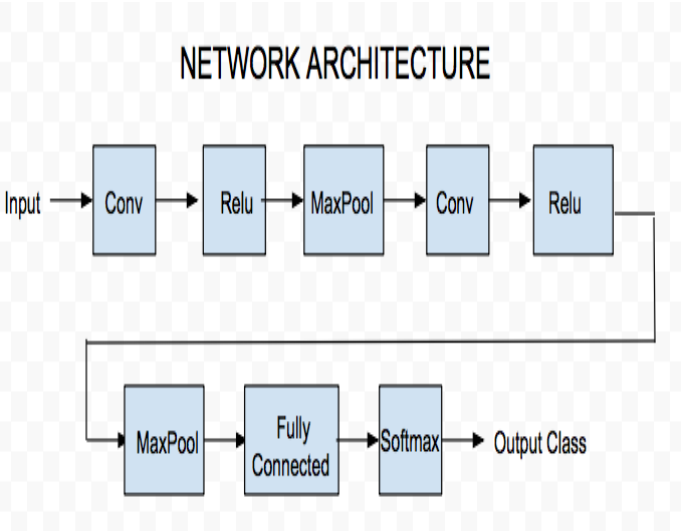
The Algorithm proposed as is, always requires 1000 randomly initiating chromosomes. As an extension to the Algorithm proposed by Sholomon et al[1], we decided to try to influence the starting state of the Genetic Algorithm. The rationale is that if the the genetic algorithm starts out with chromosomes that are already quite good, then convergence would be faster and the number of generations required for the algorithm could be reduced. We decided to train a convolutional Neural Network to solve the Jigsaw task and then use its output as input to the Genetic Algorithm.

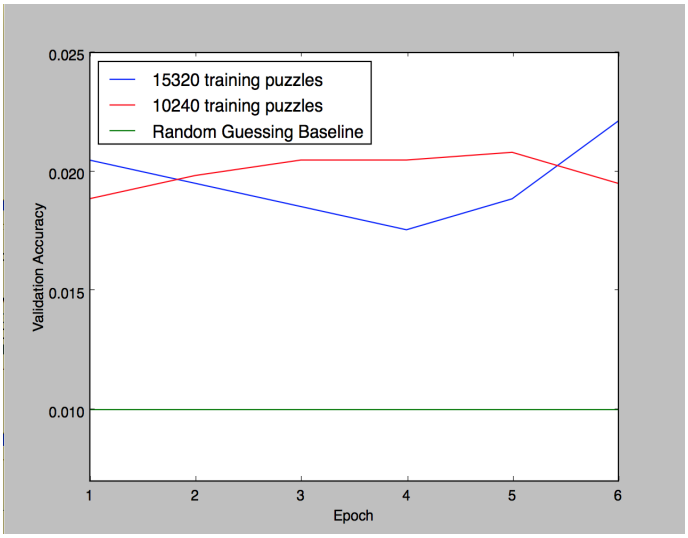
Problem Formulation

The input to the neural network is an image whose color channel are made up the jigsaw pieces stacked side by side. The task of the network was to predict the order in which the pieces were stacked together, thus assigning each piece its right position in the original image. To clarify, say we have a 3x3 puzzle. The pieces are numbered 1 to 9 according to their position in the original image. They are then stacked in a random configuration along the color channel. It is then the task of the network to predict the configuration in which they were stacked. We cast the problem as a classification problem. Since the configurations space is really large, 9 pieces produce $9! = 362990$ possibilities, it would be near impossible, given the computing resources at our disposal, to have $9!$ classes. As such, we decided to reduce the problem to the following. We keep track of 100 classes representing 100 randomly generated configurations. The objective of the network is now to predict the configuration (1 - 100) that has the closest Hamming Distance to the actual configuration of the given image. This work around made the solution space of the problem tractable.

Network Architecture and Implementation Details

The diagram below shows the network structure. Our implementation of the network was done in TensorFlow. We used a softmax cross entropy loss function. A hyper parameter search lead us to Adam Optimizer with a learning rate of 10^{-3} and a batch size of 128. We had to normalize the image channels to 1 by dividing by 255.





3.3 Evaluation Metric

There are two major metrics used to evaluate a jigsaw reconstruction. There is the direct comparison which measures the fraction of pieces located in their correct absolute location, and there is neighbor comparison, which measures the fraction of correct neighbors. The direct method has been shown [1] to be less accurate and less meaningful since it cannot handle slightly shifted cases. We thus used the neighbor comparison as our evaluation metric.

4 Experiments

4.1 Convolutional Neural Network

Given that we cast our solution space for the network into 100 classes, we set our baseline to be a validation accuracy of 0.01 corresponding to random guessing out of the 100 classes. Our convolutional neural network, was able to achieve validation accuracy of 0.022 which is more than twice our baseline. This performance is quite impressive considering that what the network is actually trying to achieve is predicting a configuration space of 9! that is being represented by 100 classes. The more puzzles provided to the network to solve, the better it got as the plot below suggests.

4.2 Solving Jigsaws via Genetic Algorithm

The table below shows the run time and accuracy results for difference puzzle piece sizes averaged over 10 runs. As can be seen from the table, we were able to achieve results comparable to those of the original paper in terms of the accuracy of rearrangement of jigsaw pieces and the fitness score of the result returned. Our reconstructions, though not always as accurate as the original by the neighbor accuracy metric described, produced a reconstructions whose fitness score were equal to the fitness score of the original image.This suggests that the algorithm sometimes gets stuck in a local minimum whose fitness score is the same as the un-jumbled image's score.

Figure 1. Algorithm Performances on Different Piece Numbers

Algorithm Performance Averaged Over 10 runs. Values given are mean over 10 runs

	4 Pieces	16 Pieces	32 pieces	64 Pieces	96 Pieces
Runtime	14.6s	49.1s	165.2s	452.8s	884.7s
Initial Fitness	11.65	168.14	356.33	679.38	971.04
Fitness of Unjumbled	10.95	36.51	62.33	106.03	147.46
Fitness of Reconstruction	10.95	36.51	62.33	106.03	156.45
Reconstruction Accuracy	70%	50%	100%	70%	77.9070%

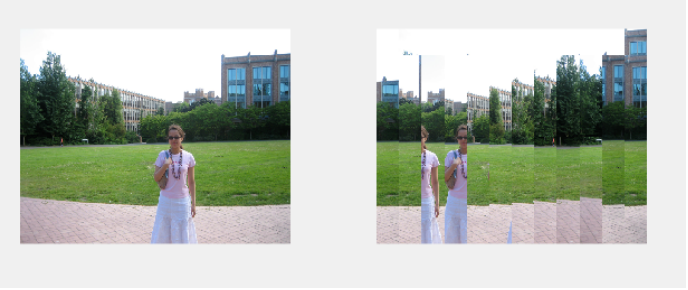
*Initial Fitness is the best fitness score of the starting chromosomes.

In the domain of run time however, we were unable to match the original paper's results. The paper describes solving a 432 piece puzzle in 43.63 seconds, however our implementation takes around 2 hours to run on a puzzle of the same size. We believe this huge difference is due to differences in the specifics of the implementation of the crossover function.

Figure 2. 96 PIECES:GENERATION 1

Left: Best Reconstruction so far. Right: Second Best

Figure 3. 96 PIECES:GENERATION 100



Left: Actual Image. Right: Reconstructed Image.

4.3 Genetic Algorithm + Convolutional Neural Network (CNN)

As an augmentation to the original algorithm, we fed the reconstruction output of the CNN as the starting population of the Genetic Algorithm. We were mainly interested in two effects
1.Did the run time of the algorithm improve?
2.How did the accuracy of the reconstruction change?

Algorithm Performance Averaged Over 10 runs. Values given are mean over 10 runs

	CNN Augmentation 5120 Training samples	CNN Augmentation 10240 Training samples	CNN Augmentation 15360 Training Samples	No CNN Augmentation
Runtime	31.7650s	31.5004s	31.6363	32.4472s
Initial Fitness	33.5658	31.8653	31.5795	29.8991
Reconstruction Fitness	19.2903	19.2903	19.2903	19.2903
Reconstruction Accuracy	70%	60%	60%	70%

*Initial Fitness is the best fitness score of the starting chromosomes.

The table above contrasts the performance of the CNN augmentation in different regimes of training data size with the pure Genetic Algorithm performance on the 3x3 Jigsaw. In general, the augmented algorithm has a better run time. Though the difference is 1 second for this regime of 3x3 puzzles, it is easy to see how this gain could be more significant as the puzzle size increases. In general, the reconstructed puzzle from the augmented model are not as accurate as the pure Genetic Algorithm. However, the reconstructions always had the same minimal fitness score as the original, meaning that it is finding a good minimum, even if this is not the original reconstruction.

5 Conclusion

The Jigsaw puzzle problem is an interesting problem with applications in many domains. Looking forward, one extension we plan to explore is to solve the jigsaw problem using only a neural network. We envision embedding convolutional layers in a Long Short Term Memory or Recurrent Neural Network which would directly predict the right configurations instead of using our current trick of having 100 representative configurations. We would also like to investigate more avenues for improving the run time of our current model.

References

[1] H. Freeman and L. Garder. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. IEEE Transactions on Electronic Computers, EC-13(2):118127, 196

[2] T. Cho, S. Avidan, and W. Freeman. A probabilistic image jigsaw puzzle solver. In IEEE Conference on Computer Vision and Pattern Recognition, pages 183190, 2010.
[3] D. Sholomon, O. E. David, and N. S. Netanyahu. A genetic algorithm-based solver for very large jigsaw puzzles. In IEEE Conference on Computer Vision and Pattern Recognition, pages 17671774, 2013.
[4] D. Pomeranz, M. Shemesh, and O. Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. In IEEE Con- ference on Computer Vision and Pattern Recognition, pages 916, 2011