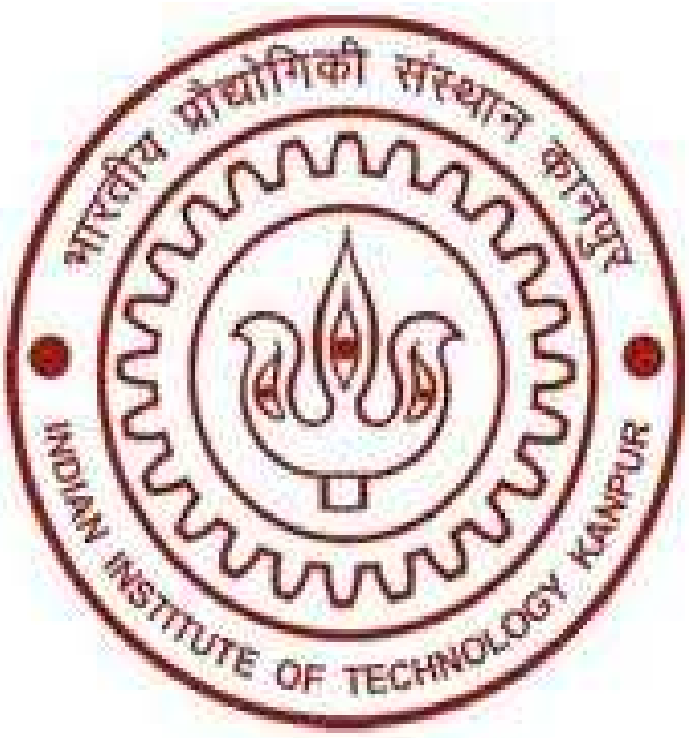
PROJECT - REPORT



CS365
Artificial Intelligence

Genetic Algorithm for Edge matching puzzles
INSTRUCTOR: Amitabh Mukherjee

Group S4

Aniruddha Kumar Sahu
Gangaprasad Koturwar

Abstract

Edge Matching puzzles are ancient puzzles and were put forth to the world as challenge in 2007 by Christopher Monckton. The puzzle was modified and was published under the name Eternity. The puzzle attracted programmers and mathematicians owing to its huge search space and the NP completeness of the problem. No complete solution has been provided to the puzzle yet and thus the puzzle still continues to be the challenge.

Further details regarding this challenge can be found at: `[Eternity-Puzzle.com]` [1]

Through this project under Artificial Intelligence we are trying to apply Genetic Algorithm to solve the puzzle. The results when compared to other successful algorithms are poor but they can be improved to a greater extent.

PUZZLE OVERVIEW: The puzzle consists of 256 tiles, each tile(fig 1) colored with different colors out of color patterns provided.

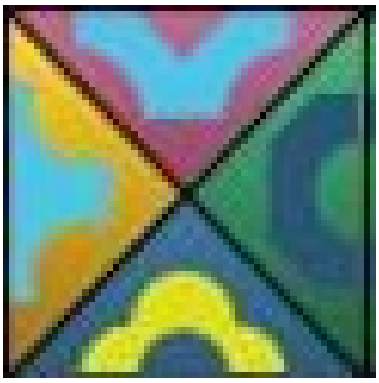


fig 1. A tile in the puzzle [2]

Each tile has its edges colored with different colors from the color patterns(fig 2) provided(consists of 23 colors). The puzzle has been developed by taking care that no two tiles will have all their edge patterns matching.

[1] `http://www.eternity-puzzle.com`
[2] Courtesy: Papa Ousmane Niang's paper

fig 2. Patterns used in puzzle [3]

Tiles are to be placed on a 16X16 grid. The constraint to be followed in placing tiles is to make sure the matching edges of the adjecent tiles should match w.r.t the color they represent. Thus in case of 16X16 grid there can be maximum of the 480 edges to be matched. Apart from the color pattern discussed above a particular color(grey in this case) is reserved for the boundry tiles. Each entering tile can adapt any of the 4 orientation owing to its rotation. i.e once the tile is chosen there are further 4 choices regarding the placement.

The puzzle can be better understood if smaller versions of the same are considered. This puzzle can be viewed as 2X2 and also in 4X4 versions. In case of 2X2 there can be maximum of 4 edge matches and 24 in case of 4X4. A general formula for maximum edge matches of nXn puzzle is:

$$(\text{E.M})_{max} = 2n(n-1)$$

The main challenge of the problem is its huge search space. If all the possibilities are considered while placing the tiles the total no of possibilities in case of 2X2 turns out to be 4! in chosing the tile and further 4 choices for each chosen tile i.e. $4!X4^4$ or 6144. The similar calculation for 8X8 gives $64!X4^{64}$ possibilities!!! And the value increases rapidly in case of 16X16 which is $(256!X4^{256})$ turns out to be of the order 10^{665}. Which makes all the search algorithms less effective in solving the puzzle.

RECURSIVE BACKTRACKING ALGORITHM: This is the most primary attempt in searching the solution. It builds the grid through piece by piece construction of the grid. The algorithm follows the following steps:

[3]Courtesy: Papa Ousmane Niang's paper (Figure 1)

---

1. *Initialize by placing a random tile on grid*
   at any point of buiding solution,

2. *Search for the suitable tile to be placed*

   - If found place then,

   place the tile;

   - If not found then,

   remove last placement and search for another alternative;

3. *Continue the procedure untill whole grid is filled*

Recursive Backtracking Algorithm

The algorithm is very promising to provide the solution and meets the requirements in smaller versions of the puzzle. The drawback of the program is its recursive backtracking. As the size of search space increases program takes lots of time. Results from the previous attempts shows the time taken to solve the puzzle:

Code	**6X6**	**8X8**	**10X10**
Doc Smith	891msec	91mins	10+hrs
Joel	790msec	107mins	10+hrs

Table 1 Results comparison

Doc smiths code is written in C++ and Joels code uses the same algorithm but implemented using JAVA. The same code when tried for 16X16 takes several days to show reasonable progress.

Further details regarding these implementations can be found at: `www.grokcode.com` [4]

GENETIC ALGORITHM: The evolutionary algorithm which primarily works similar to the evolution process on earth. As a very rapid overview to the algorithm, the algorithm works as follows:

[4] `http://www.grokcode.com/10/e2-the-np-complete-kids-game-with-the-2-million-prize/`

A population is created by creating a group of individuals randomly. The individuals in the population are then evaluated based on their fitness which is defined by the user. The fitness function gives the individuals a score based on how well they can perform at the given task. Two individuals are then selected based on their fitness, the higher the fitness, the higher the chance of being selected. These individuals then "reproduce" to create one or more offspring, after which the offspring are mutated randomly. This continues until a suitable solution has been found or a certain number of generations have passed, depending on the needs of the programmer.

Detailed discussion for the same can be found at: `Genetic Algorithm Overview` [5]

- **Terminologies:**

  - **Individual**
    Any possible solution to the problem is represented as Individuals in the evolutionary system. The Individual which meets the requirements of the solution is called the best individual and is returned as solution.
    Individuals in case of Eternity II can be represented as all possible grid fillings.
    For ex:

33:4	202:3	113:3	52:3	23:3	6:3	13:1	98:2	59:3	38:3	226:3	234:1	120:4	118:2	159:1	138:4
26:2	97:2	136:4	105:2	84:2	144:2	99:4	178:2	254:1	212:3	81:1	64:1	74:4	96:2	141:3	174:2
72:2	183:1	65:4	91:2	123:4	124:2	221:2	205:1	230:2	87:4	62:2	41:3	39:1	53:2	155:3	251:1
68:2	195:3	188:1	132:1	198:2	168:4	222:1	133:1	129:2	252:2	152:2	210:4	58:2	48:4	37:2	54:2
170:3	246:2	225:4	67:4	49:4	112:4	114:2	111:1	63:4	116:1	197:4	71:1	30:4	137:2	24:4	55:3
90:1	208:4	237:2	238:1	164:3	176:4	175:2	104:3	103:2	167:1	217:2	88:3	32:4	211:2	171:3	160:4
243:3	163:4	92:4	157:4	156:2	115:4	102:2	143:3	145:1	165:3	191:2	79:3	9:3	5:3	161:4	16:4
15:3	199:2	244:4	235:4	216:2	186:2	34:4	255:1	154:4	82:3	166:4	250:2	35:4	76:3	190:1	239:2
108:3	131:4	28:4	10:2	153:1	247:2	80:3	229:4	224:2	162:3	119:4	121:2	4:1	192:3	44:4	29:3
134:2	181:4	180:2	47:4	142:3	117:1	127:1	61:4	232:4	172:3	241:4	8:3	25:3	20:1	31:3	60:3
135:1	182:2	253:1	42:2	248:3	21:4	109:3	93:3	86:1	100:4	50:2	107:2	18:4	220:3	196:4	193:2
101:3	83:1	218:4	11:2	233:4	51:4	140:2	169:2	215:4	207:1	179:3	213:4	214:4	200:3	125:1	57:1
249:2	85:3	149:3	70:3	189:4	56:2	95:1	204:2	227:3	209:3	231:2	185:3	223:2	110:3	130:3	201:1
94:4	173:3	73:1	219:3	187:4	148:3	75:3	228:2	203:1	256:3	245:4	128:1	206:1	14:4	2:2	45:3
40:2	77:4	236:2	158:4	150:2	7:4	126:2	194:4	66:4	89:2	151:1	122:3	240:1	12:2	27:2	69:2
17:2	106:3	46:1	78:1	22:3	1:1	19:4	146:4	147:1	184:1	139:3	177:3	242:3	36:4	3:4	43:4

fig. 3 Individual representation for Eternity II.

[5] `http://geneticalgorithms.ai-depot.com/Tutorial/Overview.html`

Here the entries shows the tile no and orientation. for ex 33:4 represents tile no 33 placed with 270^o clockwise rotation.

- **Population**
  The group of all the Individuals add up to create a population. Population is total no of all the Individuals in each generation.

- **Chromosomes**
  The blue-print of each Individuals stores the various properties. These serves the similar purpose served by chromosomes.

- **Genes**
  The various properties of Individuals are stored as genes.

- **Fitness**
  The fitness function indirectly gives the surviving probability of the individual, and is totally dependent on the intended solution.
  Fitness for the individuals can be related with the no. of edge matches. Exact function for the same is

$$\textbf{Fitness} = \textbf{1-}\frac{\textbf{no of edge matches}}{\textbf{max no of edge matches}}$$

- **The Operations:**
  The various evolutionary operations are used to create a new generation with better surviving abilities(fitness) to reach the perfect individual faster. These operations are discussed below.

  - **Selection**
    The individuals are selected from the population to apply different operations to create new generations. The selection is mostly done on the basis of fitness values. i.e. the individuels with higher fitness has higher probability of getting selected.
  - **Crossover**
    Two parents are selected through selection and then the child is created by passing some properties of one parent and remaining from the other. Refer fig 3.

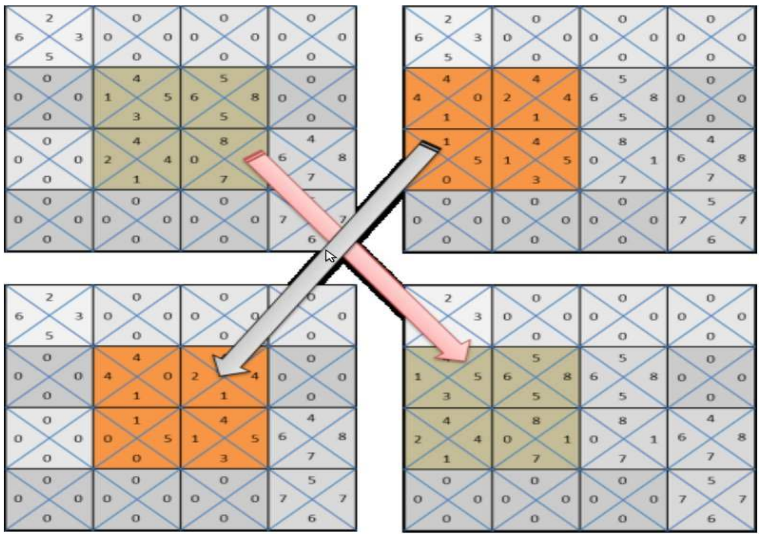


fig. 3 The crossover operation [6]

The same has been implemented for the puzzle as follows:
Two parents are selected (the ones with best fitness) then randomly two regions are chosen in two randomly selected points among the parents. Here region is defined as some area mXn tiles, (where m and n are parameters of selected rectangular area). Then by exchanging region parents generate two offsprings(children) with new genetic material, but preserve the overall composition of parents.

The steps involved in Crossover operations can be found in 4.6 section of [Niangs paper] [7]

– **Mutation**
A parent is selected and is allowed to undergo certain property changes internally to create fitter child. Refer fig 4.

In case of the puzzle this operation has been implemented as follows:
Two type of mutation are performed, (a) Rotate region Mutation and (b) Swap Mutation.
(a) Rotate region Mutation operator, rotates any randomly selected region containing NxN tiles(where N could be one or more than one) in any possible angles(90, 180, 270 degree)

[6] Courtesy: Papa Ousmane Niang's paper (Figure 22)
[7] Paper can be found: `http://home.iitk.ac.in/~anirkus/cs365/projects/p4.pdf`

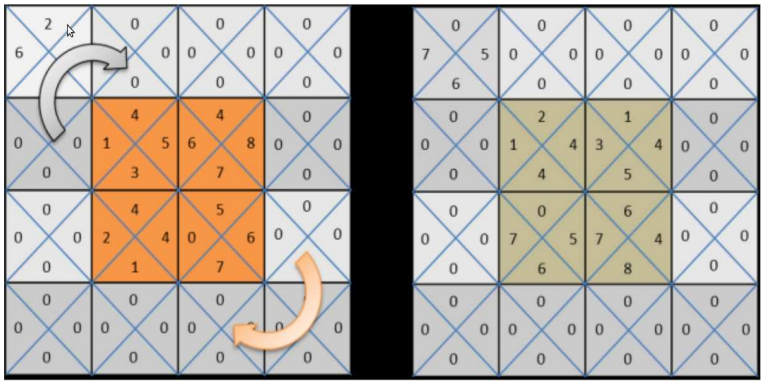


fig. 4 The Rotate-Mutation Operation [8]

(b) Swap Mutaion operator, exchanges two randomly regions(again, can be of random size) from same board(parent)

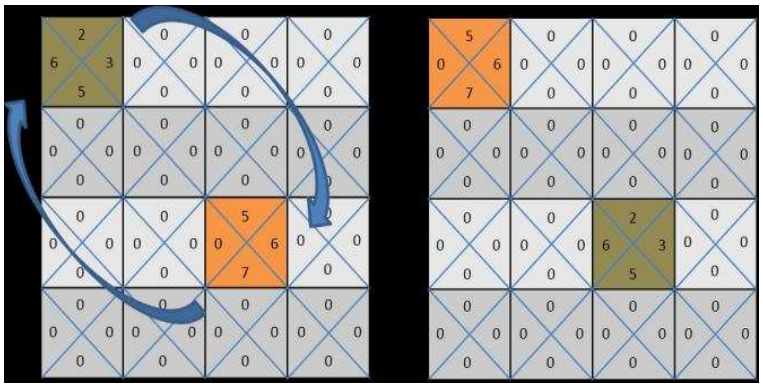


fig. 5 The Swap-Mutation Operation [9]

The steps involved in Mutation operations can be found in 4.5 section of [Niangs paper]

– **Elitism**
During each generation few of the top best fit individuals are selected to retain their properties as it is for the next generation.

– **Evaluations**
The total no of individuals formed is basically the evaluations. It can be used as stopping criterion in case best individual is not found.

[8] Courtesy: Papa Ousmane Niang's paper (Figure 17)
[9] Courtesy: Papa Ousmane Niang's paper (Figure 15)

- **Algorithm**

---

1. *Initialize by creating a random group of Individuals*
   at any point of creating new generation,
2. *Search for the best individual*
   - *If found then,*
     return;
   - *If not found then,*
     continue;
3. *Evaluate the fitness of the each Individual*
4. *Do operations*
   Crossover, Mutation, Elitism etc
5. *Generate the new generation*
6. *Repeat*

Genetic Algorithm

- **Implementation**

The code has been implemented in C++. We are using the code by Jorge Munoz with above mentioned implementations.
The code requires little more efforts to run, as the readme file is somewhat misleading. But code can be used to implement all the necessary parameters.

The details regarding code:[Jorge Munoz paper] [10]

[10]Code can be found on: `http://www.caos.inf.uc3m.es/~jorge/papers.php?lang=en#a2009`

– **Results:**
The evolutionary operations play a vital role in achieving the solution. These operations independently try to increase the fitness of the new generation. The code takes many parameters and runs accordingly. User is supposed to provide the rates of different evolutionary operations. The optimization of these rates can be carried out to improve the results.
The parameter values are read as relative to each other. For ex, {elitism rate:10,mutation rate:50,crossover rate:10} makes $population * \frac{10}{10+50+10}$ individuals to be retained for the next generation as elite individuals. Similarly the no of individuals undergoing crossover,mutation can also be calculated.
The following graph shows the no of successful edge matches for different parameters:

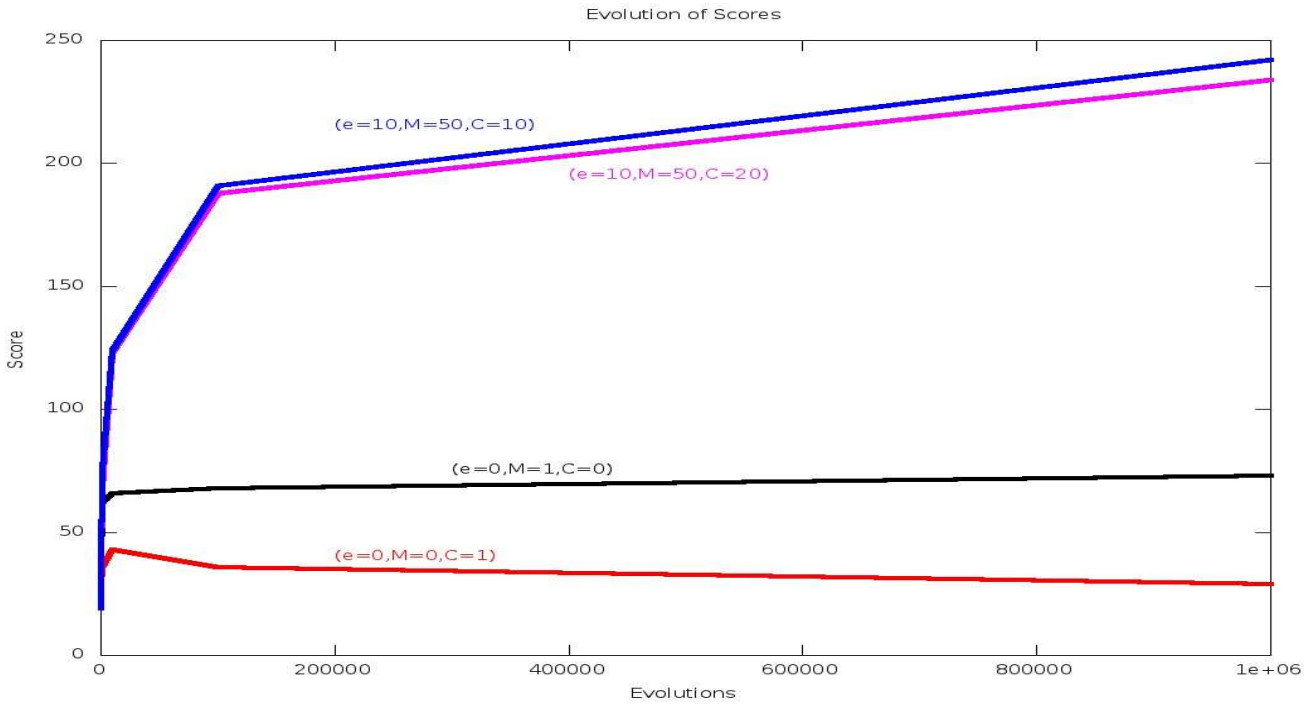


fig. 6 Score-Evaluations graph

The graph shows the various scores achieved by altering the evolutionary operations. There cant be a definite answer for best set of rates of these operations, but the trend can be learnt through sufficient number of experiments. The Experiments we carried showed the above results. (These are the best of the certain number of trials carried for each category)

From the graph it is clear that the operations apllied individually give poor results but the simulteneous operations improves the result to a larger extent.

code takes too much time to give the output in case of higher evaluation limits. The best result achieved through this code had matched 391 edges out of 480. i.e. the fitness value of 0.185416667. We were able to reach 292 (fitness 0.391666667) in much lesser number of evaluations, which we couldnt test for higher number of evaluations due to the time it takes and system gets hung in between.

The following image shows the individual with 292 matched edges.

```
# Pieces:  256
# Score:   292
# Fitness: 0.391667
#
```

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	110:4	63:1	149:1	93:4	95:2	140:3	138:2	167:2	200:3	198:3	199:3	130:4	134:2	91:4	71:2	137:3
2	194:3	47:3	37:1	12:1	192:3	168:3	242:2	90:4	81:2	77:1	254:3	88:1	204:1	186:4	237:1	31:4
3	241:4	183:4	14:4	209:4	213:2	153:1	232:4	79:1	147:1	49:4	96:1	75:3	72:1	67:1	50:4	1:1
4	34:1	166:2	57:4	202:2	251:4	24:2	252:1	125:4	105:2	53:2	70:3	106:4	126:2	99:3	25:3	22:1
5	94:2	170:4	8:4	188:2	114:4	9:2	83:4	68:2	171:4	161:4	84:4	97:2	108:4	132:2	195:1	215:3
6	201:1	107:3	52:4	256:1	227:3	119:1	238:1	181:3	51:2	129:2	218:4	203:4	127:4	121:1	197:3	58:3
7	205:2	190:4	66:3	160:2	7:1	5:3	86:1	185:3	115:1	145:1	36:2	221:1	35:1	55:3	23:3	15:1
8	3:2	193:4	206:3	144:4	92:3	69:3	64:1	118:1	128:3	154:3	10:4	217:3	233:3	151:1	247:3	43:4
9	29:2	157:2	100:1	141:3	169:2	224:2	240:1	124:3	234:3	155:3	45:4	150:2	235:3	16:1	28:3	32:1
10	214:4	113:1	78:3	176:4	182:2	236:1	112:4	46:1	48:1	2:3	38:2	191:4	62:4	80:2	133:3	117:4
11	248:1	249:4	208:3	116:1	163:3	196:4	184:2	220:1	143:4	82:3	165:3	76:4	74:4	89:2	139:1	223:2
12	243:2	212:1	103:4	101:2	56:2	231:1	30:1	20:2	73:4	244:3	211:3	245:2	229:3	255:2	164:3	216:4
13	19:2	207:3	135:3	225:4	27:2	228:3	219:3	18:2	152:4	158:2	180:4	85:1	111:1	104:1	178:2	87:1
14	40:4	136:2	120:2	230:3	4:1	33:3	226:3	175:4	172:2	59:4	246:1	65:3	123:3	131:3	250:1	222:4
15	189:4	162:4	60:1	41:3	42:1	13:1	239:3	177:4	159:3	54:2	156:1	6:4	148:4	142:2	173:1	174:1
16	210:2	102:4	17:3	187:2	21:2	26:3	44:1	253:4	98:4	39:2	61:4	11:2	146:2	109:2	179:3	122:2

fig. 7 The individual with 292 edge matches. [11]

– **Drawbacks and future progress:**
The algorithm works fairly well for completely unorganized search space and can be applied in all the cases where other algorithms fail. But the implementation used in the code leads the process to get stuck in the local maxima. This is very serious problem in this code and can be realised by codes failure to solve lower order puzzles like 2X2(gets stuck at 0.5 fitness) and 4X4(0.333 fitness). This can be avoided by using adaptive operations. As the algorithm proceeds crossover starts having negative impact on the fitness of the individual, and focused mutation operation can be used to parse the local maxima. Code is too complex to understand and to carry out these modifications but the experiments supports this.

[11] Its original problem Tile is on: `http://home.iitk.ac.in/~anirkus/cs365/projects/Tiles.txt`

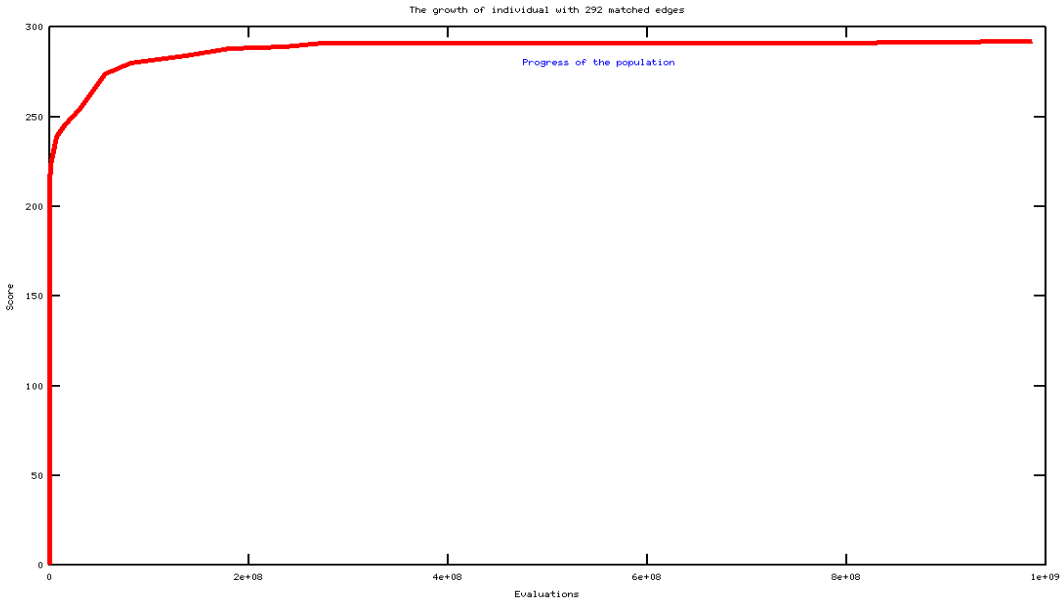


fig. 8 The growth of best individual and problem of local maxima

Apart from this the fitness criteria used primarily focuses on just the no of matched edges. If a fitness function which depends upon some useful patterns along with the existing criteria(i.e. Multi objective approach would be more favourable). Also the tournament selection criteria has been deliberately adapted to avoid extra complexity. A better selection operation which not only focuses on certain of the best but the whole population can be introduced.

- **The best score:**
  The best score till date was achieved by using Tabu search algorithm. Roughly the working of the Tabu Search is as follows:
  The algorithm initializes by creating a random solution. During each iteration, different solutions are formed by some standard procedure. All the solution leading to lowering in the previous result are discarded and others are made to undergo different solution formings. The ruling out of the weak solution provides this method the required upper edge in problem of local maxima. This method could match as much as 418 edges in 478,196,791 evaluations. The overall best solution had 467 matched edges which also basically used this algorithm.

  Details can be found in [Wei-Sin Wang and Tsung-Che Chiang paper] [12]

[12]Paper can be found: `http://home.iitk.ac.in/~anirkus/cs365/projects/p5.pdf`

- **References:**

1. Evolutionary Genetic Algorithms in a Constraint Satisfaction Problem: Puzzle Eternity-II

   by "Jorge Munoz, German Gutierrez, and Araceli Sanchis", University Carlos III of Madrid Avda. de la Universidad 30, 28911 Legan's, Spain (2009)

2. Solving the Eternity-II Puzzle Using Evolutionary Computing Techniques

   (A thesis) by "Papa Ousmane Niang", Concordia University, Montreal, Quebec, Canada (2010)

3. Solving Eternity-II puzzles with a tabu search algorithm

   by Wei-Sin Wang and Tsung-Che Chiang, National Taiwan Normal University,Taiwan, R.O.C.(2010)

4. Eternity II puzzle - Wikipedia, the free encyclopedia

   `http://en.wikipedia.org/wiki/Eternity_II_puzzle`

5. Codes of Jorge Munoz

   Codes that used for paper "Evolutionary Genetic Algorithms in a Constraint Satisfaction Problem: Puzzle Eternity II"