Multi-Robot Path Planning using Co-Evolutionary Genetic Programming

Rahul Kala
School of Cybernetics, University of Reading, Reading, Berkshire, UK

rkala001@gmail.com, Ph: +44 (0) 7466830600, http://rkala.99k.org/

Citation: R. Kala (2012) Multi-Robot Path Planning using Co-Evolutionary Genetic Programming, *Expert Systems With Applications*, 39(3): 3817-3831.

Final Version Available At: http://www.sciencedirect.com/science/article/pii/S0957417411014138

Abstract: Motion planning for multiple mobile robots must ensure the optimality of the path of each and every robot, as well as overall path optimality, which requires cooperation amongst robots. The paper proposes a solution to the problem, considering different source and goal of each robot. Each robot uses a Grammar based Genetic Programming for figuring the optimal path in a maze-like map, while a master evolutionary algorithm caters to the needs of overall path optimality. Co-operation amongst the individual robots' evolutionary algorithms ensures generation of overall optimal paths. The other feature of the algorithm includes local optimization using memory based lookup where optimal paths between various crosses in map are stored and regularly updated. Feature called wait for robot is used in place of conventionally used priority based techniques. Experiments are carried out with a number of maps, scenarios, and different robotic speeds. Experimental results confirm the usefulness of the algorithm in a variety of scenarios.

Keywords: Path Planning, Motion Planning, Mobile Robotics, Genetic Programming, Grammatical Evolution, Co-operative Evolution, Multi-Robot Systems.

1. Introduction

The problem of multi-robot motion planning deals with computation of paths of various robots such that each robot has an optimal or near optimal path, but the overall path of all the robots combined is optimal. This is a more complex task as compared to a single-robot motion planning, where the factor of coordination among the various robots is not applicable, and the single robot can use its own means to compute the path (Parker, Schneider, and Schultz, 2005). The problem of motion planning may be centralized or decentralized. In centralized planning all the robots are centrally planned by a planner, usually taking into account all the complex interactions that they may have. This results in the generation of a very complex configuration space, over which the search is to be performed. The decentralized planning, on the other hand, has an independent planner for every robot. Each robot is planned separately in its own configuration space, which makes the planning much simpler. Then efforts may be made to avoid the possibility of collision between the various robots. The centralized planning is more time consuming, but optimal as compared to decentralized planning (Arai and Ota, 1992; Sánchez-Ante and Latombe, 2002).

In this paper we assume a simple problem modeling scenario. We have a maze like map of M *x* N grids. Each grid may be black or white, denoting the presence or absence of obstacle, which are all assumed to be stationary. The grid is however not the unit position for robots which can occupy partial grid positions on the map. The complete map consists of horizontal and vertical lanes, which cris-cross each other at grids known as crosses. The task is to move n robots R_1, R_2, … R_n. Each robot R_i has a start grid S_i, which is its initial location; a goal grid G_i, which is the destination where it intends to arrive at the end of its journey; and a constant speed of motion V_i ($0 < V_i \leq 1$) grids/unit time step. All robots are assumed to be of the same size of 1 *x* 1 grids. The planner needs to plan the path of all the robots R_i. Let the path of robot R_i be given by $P_i(t)$, which denotes its position at time t. We know that $P_i(0)=S_i$. Let the path generated be such that the robot R_i reaches the goal G_i at time T_i i.e. $P_i(T_i)=G_i$. We assume that after reaching the goal, the robot

disappears from the goal, and does not obstruct other robots to pass by i.e. $P_i(T_i+1) = NIL$. In this assumption we differ from the works over robot priorities by Oliver et al. (); Bennewitz, Burgard, and Thrun (2001, 2002); and Carpin and Pagello (2009). The planner needs to work such that the average time of travel for all the robots is as small as possible i.e. $\min(\sum T_i)/n$. At the same time the planner needs to ensure that no collision occurs. Hence $P_i(t) \neq P_j(t) \vee 1 \leq i, j \leq n, i \neq j$.

Genetic Programming is extensively used evolutionary technique for problem solving. Here we represent the individual as a program and hence try to evolve it using the methodology of the evolutionary algorithms. The solution is obtained by executing the program so formed. Tree based representation of a program is a very common representation of individuals of genetic programming (J. R. Koza, 1992; Shukla, Tiwari, and Kala, 2010). The linear representation of the program of genetic programming gives rise to Grammatical Evolution. Here the individual is a sequence of integers. Every problem has its grammar, which represents the program syntax. The grammar is specified by Backus-Naur or BNF form. The generation of the phenotype solution from its genotype solution or sequence of integers takes place by selecting and firing rules specified in the grammar (O'Neil and Ryan, 2003, 2001). Our implementation of the Genetic Programming in this paper is similar to the general algorithm of grammatical evolution.

The evolutionary algorithms are used for solving various kinds of problems. These algorithms however face problems when the dimensionality of the problem becomes very large. In such a case the optimization provided by these algorithms becomes very slow, with fairly large chance of the algorithm getting struck at some local optima (Kala, Shukla, and Tiwari, 2010a). Co-operative evolutionary algorithms or co-evolutionary algorithms break up the problem into a number of smaller problems that together solve the main problems. The smaller problems have smaller dimensionality, and are hence very easy to be optimized. The different sub-problems or sub-modules evolve in parallel, which ultimately results in generation of very good modules that unite with each other to solve the problem well. Cooperation is encouraged between modules, and a module is given high fitness not only because it results in generation of higher fitness solutions, but also because it enables other modules to achieve high fitness value (Potter and DeJong, 1994, 2000; García-Pedrajas, Hervás-Martínez, and Muñoz Pérez, 2003).

In this paper we first present a co-evolutionary genetic programming based planning of multiple robots. Each of the individuals has its own optimization process that is based on the principles of Grammatical Evolution. These all try to generate and optimize the paths of the individual robots. However these need to consider the possibility of collision between the robots, for which the factor of co-operation comes into play and has been induced in the algorithm. A master genetic algorithm runs to select the best paths of the robots, out of all the paths available in the individual optimizations.

The major problem with the assumed scenario is the variable speed. This puts forward a number of scenarios, where sub-optimality may be possible without careful planning. Consider a big corridor with two robots A and B marching towards it. Consider maximum velocity of A being higher i.e. $V_A >> V_B$. Consider that robot B is closer to the corridor entry, and would enter it before A. But if robot B enters the corridor first, robot A would have to follow it, and both robots would be able to walk with a maximum velocity of V_B. It may be clearly seen that optimal strategy would be in case B waits for robot A to enter the corridor, and follows it inside the corridor. In such a case both robots would get a chance to travel by their maximum velocities. Hence we implement the concept of wait for robot, where a robot may be asked to wait so that some other robot may cross.

Optimization by evolutionary algorithms may get very complex in problems having too many dimensions and multiple modalities. In such a case it is often useful to use a local search strategy based on predefined heuristics that can aid the evolutionary algorithms. This saves the algorithm from likely convergence into local optima. The local search algorithm carries forward the task to converge the algorithm into some local or global optima, while the evolutionary algorithm does the task of locating the best possible regions, where the global optima may lie. In such a case the combined efforts of heuristics and evolutionary algorithms results in better optimization (Kala,Shukla, Tiwari, 2009). The same problem of complex optimization is prevalent in the present algorithm. Here the evolutionary algorithm tries to optimize the individual robotic path. In such a case also we use a local search strategy which tries to modify the path to

make it smaller in length. Longer paths are replaced by smaller paths. For carrying forward this replacement, an external memory is reserved that stores the smallest paths between all pairs of crosses. This table is regularly updated as more paths are found.

The novelty of the paper is threefold. We first propose a co-evolutionary genetic programming model for solving the problem. This model uses principles of cooperative evolution to generate paths that are optimal in time, by mixing centralized as well as decentralized planning. Secondly, we propose the model and issues with multi-speed, map based (single-lane) multi-robot motion planning. This includes the use of the wait for robot feature incorporated into the evolutionary approach. Thirdly we propose use of external memory for local optimization of the paths. This memory is initialized, updated, and queried for reduction in paths of individual robots.

The paper is organized as follows. In section 2 we present some of the related works into the field. In section 3 we present the co-evolutionary genetic programming model of the algorithm. Section 4 presents the concept of wait for robot. Section 5 presents the memory based local search used in the algorithm. The experimental results are given in section 6. Section 7 gives the conclusion remarks.

2. Related Works

The entire paradigm of path planning of multi-robots involve a large set of issues that range from dynamic changes in the robotic plan, to validity of non-holonomic constants, and execution and memory time complexity. Oliver et al. (1999) give a general architecture of path planning in these scalable scenarios. They followed a decentralized approach of planning where every robot does its own independent path planning. The map is built centrally at start and all changes are continuously monitored and any needed correction is made. Possible collisions are resolved by the coordination module, that uses a priority based coordination with a higher priority robot allowed to make the collision prone move, while the other robot compromises or waits. Possible collisions are predicted be extrapolation of the motion of robots, and any possibility is marked by a check-point. One of the robots is allowed to move, while the path of other may be re-planned.

Motion planning may be regarded as a part of the entire robotic architecture, for which again numerous models have been built. Asama, Matsumoto, and Ishida (1989) present a complete system for multi-robot planning, coordination, and control in their designed system ACTRESS (ACTor-based Robots and Equipments Synthetic System). In this system the authors demonstrate the mechanisms in which the different modules including sensors, processing, planning, manipulators, communication etc. come into play for intelligent and autonomous task accomplishment. The extension of their work can be found in (Asama et al. 1991) where the authors developed a multi-level path planning. The planning was different for static and dynamic cases. Special preventions were taken for deadlock avoidance that may take place many times in planning of multiple robots. The complete hierarchy of the system includes static path planning, use of local algorithm, use of mobile rules, task prioritization, deadlock avoidance by low-level deadlock solver, deadlock avoidance by high-level deadlock solver, and human operator expertise. Vendrell, Mellado, and Crespo (2001) presented a general framework for motion planning with multiple robots. They highlighted the need to plan, sense, and accordingly re-plan by every robot. The authors further present a hierarchical model of the various robot activities that range from high abstraction to low abstraction. The hierarchy includes mission, task, action, motions, trajectories, and robot order.

A number of models have been made into the earlier works of the authors on single robots that revolve around evolutionary algorithms. In (Kala, Shukla and Tiwari, 2010b) a hierarchical implementation of a customized evolutionary algorithm was proposed. A number of evolutionary operators were defined to result in better convergence. The hierarchical nature enabled the robot to work in dynamic environment. The coarser evolutionary algorithm mainly looked at the static obstacles, and their optimality, the finer evolutionary algorithm tried to escape from all dynamic obstacles using a space-time graph map. In another work Kala, Shukla, and Tiwari (2010c) used evolutionary algorithm for generation of path in an incremental manner. Here the complexity of path or the number of turns that the path could have increased

with time. The concept of momentum was floated which controlled the resolution with which the robot checked for the path feasibility and other metrics.

The general approach used in our algorithm is of Evolutionary Algorithm, which is a member of the class of probabilistic algorithms, for which a bulk of research has been done. Kavaraki and Latombe (1997), Kavaraki et al. (2002) presented a method for multi-robot path planning called as the Probabilistic Road Map (PRM), which has since then been extensively modified and used for research. The algorithm consists of an offline phase and an online phase. In the offline phase the algorithm learns the map, and stores the summary about the paths between various points in a special data structure called as the road map. This step involves identification of a number of nodes that might represent difficult points in the configuration space, and are chosen by heuristics. The number of points is further increased by selection of points around these points. This selection may go on till required points are obtained, with the selection being weighted by a node's difficulty or other selected heuristics. A local planning technique is used to swiftly compute the connectivity of these points to the other points. The collection of these vertices and edges makes the road map to use used for online planning. The online planner is query based which gets a query for a path and reacts to it by returning the shortest path based on the present map as well as the summarizations stored in the offline phase. This path may be further optimized by some local optimization technique, to increase the path optimality.

Bohlin and Kavraki (2000) further extend the basic PRM to build a lazy PRM. The PRM does a large number of collision checks that usually results in a large wastage of time. In Lazy PRM the concept is to generate random nodes in the offline mode and try to connect each node to its neighbors for connectivity assessment. The feasibility of the paths is checked lazily in a coarser to finer manner. An initial lazy checking of the path feasibility reduces number of checks, at the same time giving an idea of the feasibility. If path is found to be feasible, it may be checked at a finer level as well. The in-feasible paths are enhanced by placement of the nodes. The PRM methods can be adapted to present a strong variance between central and de-central path planning with multiple robots. A comparison of centralized planning, decoupled planning with global coordination, and decoupled planning with pair wise coordination was done by Sachez-Ante and Latombe (2002). A large number of experiments of different number of robots with large degrees of freedom and scenarios were done. Experimental results confirmed that the requirement of robot dependency was a major factor behind the better performance of the coordination level.

Clark (2005) used a probabilistic roadmap technique for solving the problem of motion planning of multiple robots. They used a single query based probabilistic roadmap planner. Here the author made a complete architecture comprising of software, communication, planning, and control where a robot could get the positional updates from all the other robots to get the complete roadmap and make decision. Here the author considered possibility of breaking of connectivity between two robots, resulting in generation of two unconnected sub-networks. The algorithm could track such changes and still make decisions of motion. Another probabilistic implementation can be seen in the work of Clark, Rock and Latombe (2003) who solved the problem of path planning for multiple-robot space system. Their planning algorithm generated random points or milestones and attempted to reach one milestone from the other, till the location was near to the goal. Heuristics were used for the selection of the milestone, whose neighborhood became the possible location of next milestone. Re-planning was an integral version of the algorithm, as the map could change in real time as per availability of information. Combined part results of a single decentralized planning system are used to serve as milestones of the centralized planner that checks for feasibility in combined motion.

Svestka and Overmars (1998) present a probabilistic solution for solving the problem of multi-robot path planning. In this approach they define the paradigm of coordinated graphs, which allows only a collision free motion of the mobile robot. The robot is not allowed to collide with any of the moving robots. Their approach is partially central in nature, where the major task of planning lies with a central authority, on whose decisions the individual robots are made to move. The entire configuration space is made discrete with respect to the movement of the multiple robots. This the authors do by the concept of flat graphs, which takes into account the prospective motion of multiple robots. Observing the highly time consuming nature of the algorithm and the low scalability to the number of robots (large attributed to the central

planning mechanism), the authors further extend their work by using multi-level graph clustering. Here the original graph is decomposed into a number of independent sub-graphs. This layered solution to the problem limits the number of nodes of the graph, which ultimately affects the time complexity of the algorithm in an exponential manner. The path hence generated by the algorithm is given some characteristics of non-central nature by introduction of mechanisms of smoothness of the path, for greater optimality. This is done by the application of a local planning algorithm for every robot in motion.

Dongyong et al. (2006) used an evolutionary approach to solve the problem. Here the paths kept improving with time. The authors used a chaos based evolution, where the individuals were disturbed by some factors depending upon user set parameters. If the disturbed individual resulted in a better fitness, it was retained else the parent was retained. The crossover and mutation rates were adapted with time as per the fitness value. The authors optimized the navigation time, path length, and path smoothness. Kuffner and LaValle (2000) present an interesting application of another probabilistic algorithm called RRT. Here the RRT Trees start exploding both from the source as well as from the goal. The intersection of the two trees hence provides the path early. Another deviation from the conventional RRT Trees that the authors present is that the exploration of any node continues to take place in the same direction till the goal, an obstacle, or map wall is met. This results in heavy exploration at every step of the algorithm, and the general ideal goes a long way in enabling the RRT escape from a local minima.

Wang and Wu (2005) make use of cooperative co-evolution in solving the problem of multi-robot path planning. This is primarily an evolutionary approach, where every robot has its own evolution. The evolution of all robots are shared, coordinated and communicated by a central mechanism. Every robot, in its evolution, enjoys a fitness that is a function of the length of path travelled and the constraints violated. The constraints refer to the individual robot constraints as well as the constraints to avoid collision with other robot. Slusny, Neruda and Vidnerova (2010) used evolutionary radial basis function networks and reinforcement learning for the task of localization and motion planning of a robot. The models were later analyzed by rule extraction technique, which converted the model to a rule based model for easy analysis and understanding. Kala, Shukla, and Tiwari (2010d) also used genetic algorithms for the optimization of a fuzzy inference system that was used for motion planning. This approach was performed at two levels. At the first level a probabilistic A* algorithm was used for making a rough sketch of the path, which was later on made detailed by the fuzzy based planner. In another work Kala, Shukla and Tiwari (2011) used evolutionary algorithms to make the rough sketch of the path, which was later traversed by the fuzzy planner. In this work again the concepts of incremental evolution were used.

Another novel part of the algorithm was of variable speeds, which again has its inspiration from literature, with numerous related models. Kolushev and Bogdanov (1999) take into account the speeds of the moving robots while deciding their path in a graphical representation of the map or the configuration space. The robots may speed down or speed up below the maximum allowable velocity to avoid collisions. At any edge, the speed is also taken account, and correspondingly both the lowest path length plan and the shortest duration plan may be found out. A simple graph search algorithm may be used for planning. Rules are made to alter the velocity such that no collision occurs at the nodes as well as the edges of the graph.

Priority based schemes have been extensively used for motion planning. Bennewitz, Burgard, and Thrun (2002) proposed a prioritized A* algorithm to coordinate the motion of multiple robots. Here the planning was done using a simple A* approach. Every robot had a priority, based on which the motion of the robots were carried out in case of possibilities of collision. The authors used a simple hill climbing approach to compute the best combination of priorities for the overall movement of robots. Priority is a major method for solving collisions in a multi-robot motion planning system. Bennewitz, Burgard, and Thrun (2001) consider the problem of finding the optimal priority scheme as the given problem. Their approach uses a combination of hill climbing and random search to set the correct priority scheme of the robots. Every priority scheme is judged by the optimality of paths generated in terms of length. At any step random exchange of priorities takes place. If these swaps result in better optimality, the change is retained, else rejected. The entire process is started multiple times from different configurations to escape from being trapped at local minima. We differ from these approaches by the fact that in our case the robots vanish after

reaching their goals, and hence do not restrict other robot movements. The concept of priority here is implemented by the concept of wait for robot.

3. Cooperative Genetic Programming

The basic working of the algorithm is a cooperative genetic programming planner. This planner operates in two levels: master and slave, similar to the architecture in (Moriarty, 1997; Moriarty and Miikkulainen, 1997; García-Pedrajas, Hervás-Martínez, and Muñoz Pérez, 2003). The slave genetic programming is basically a decentralized path planning for all the various robots in the system. Using this level the system tries to generate better and better paths for the individual motion of the various robots. The second level is the master level. Each robot in the slave has numerous genetic programming individual over which it tries to carry forward the optimization. The master is simply a genetic algorithm that tries to select the best combination of paths for the various robots, such that the overall path of all robots combined is optimal. In simple terms it may be regarded as the slaves optimize the individual robot paths, and the master optimizes the net work plan of all robots. However it needs to be noted that the slaves are conscious of cooperation amongst each other to generate collision free paths, and to help each other to optimize. Section 3.1 discusses the slave genetic programming. Section 3.2 presents the master genetic algorithm. The complete algorithm along with the interaction between the master and slave levels is done in section 3.3

3.1 Slave Genetic Programming

The slave genetic programming operates at the level of individual robots for the generation of their paths, which when combined with the other paths of the other robots result in an overall optimal movement of the entire pool of robots. Each robot has its own instance of genetic programming which comprises of multiple individuals representing the potential path of movement of the robot from source and possibly finishing at the goal.

Based on the principles of Grammatical Evolution (O'Neill and Ryan, 2001, 2003), we assume that the individual representation is in form of sequence of integers. In this manner we adopt a linear representation of the individual representing the path of the robot. Each integer denotes the movement of the robot. We know that once the robot is traveling on a straight path, it would continue to follow the same path, until it has got some point where it is possible to make a turn. The turns can only me made at some crossing, where multiple lanes meet each other. Whenever the robot meets a crossing, it needs to decide which way it has to follow. The number of ways possible may vary depending upon the type of crossing. The decision to which of the path is to be followed is decided by the individual, which uniquely determines the path of the robot. Before we discuss this conversion of the genotype or the sequence of integers to the phenotype or the robotic path, let us formally define some terms.

The direction of motion of any robot R_i may be given by d={left, right, up, down} representing the four possible directions of motion. We assume the various paths to criss-cross each other only in multiples of right angles. A robot R_i traveling in direction d at a point (x,y) is said to be at a crossing if any path criss-crosses the path of traversal of the robot in direction d at point (x,y). It may be verified that the definition of cross does not depend upon direction d. Hence

$$Cross(x,y) = \begin{cases} true & (\text{x, y}) \text{is a crossing, i.e. a vertical and horizontal line intersects} \\ false & \text{otherwise} \end{cases} \quad (1)$$

Each of the marked positions is a crossing as shown in figure 1. The robot is said to be struck at the location (x, y) while moving in direction d if it is not possible to move in the same direction. This may happen because of reaching of map limits, end of the road, or having some static obstacle ahead. Hence

$$Struck(x,y,d) = \begin{cases} true & \text{Robot can no longer move in direction d while at (x, y)} \\ false & \text{otherwise} \end{cases} \quad (2)$$

We are further interested in the number of possible moves (along with the actual moves) at every crossing, which would be used by robot for decision making. A robot would normally not like to traverse back into the same direction in which it was traveling since the traveled path would be wasteful. This however may be needed if the robot was struck. Let paths possible at any crossing (Cross(x, y)=true) at location (x,y) and direction d may hence be given by D(x,y,d), where D(x, y, d) $\subset$ {left, right, up, down}. Let size(D) denote the total number of elements in the set or the total number of moves possible. Hence

$$D(x,y,d) = \begin{cases} pt(x,y) - \sim d & struck(x,y,d) = false \\ pt(x,y) & struck(x,y,d) = true \end{cases} \quad (3)$$

Here pt(x,y) is a function that looks into the possibility of all the moves {left, right, up, down} and returns the set of possible moves. ~d is the direction of motion opposite to d.

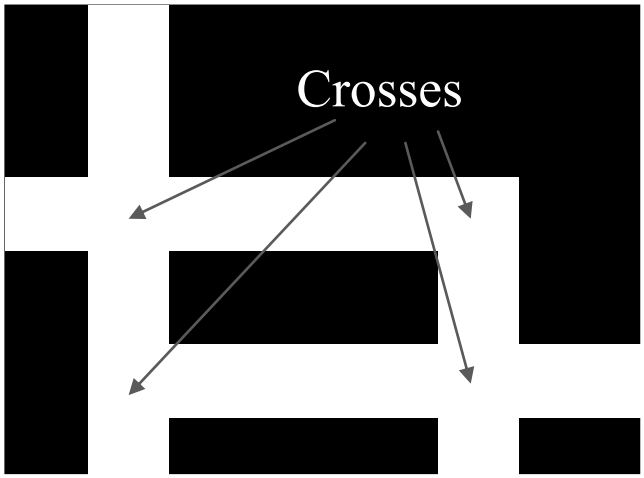

Figure 1: Various types of crosses. The figure depicts the numerous possible crosses that may be possible in a map where vertical and horizontal roads criss cross. The figure shows crosses with 4, 3, and 2 paths.

Let the genetic individual for robot R_i (that is the i^{th} instance of genetic programming) be given by $I^i<I^i_1, I^i_2, I^i_3, \ldots I^i_o>$. We need to convert the genotype I^i into the phenotype or the robotic path. Let ci be the pointer which points towards the integers in genotype and returns them whenever queried for decision making. We further define $move_A(x, y, c_i)$ as a function that returns the initial direction of motion of the robot while robot is at position (x, y) with pointer c_i. Here the robot is allowed to take any possible way. Further let $move_B(x, y, d, c_i)$ be a function that returns the next direction of motion of the robot while at position (x, y) moving with direction d with pointer c_i. Here the robot is not allowed to move back in direction it is coming from, unless it is struck. We need to first compute the initial direction of motion of the robot. We take a pointer c_i initially set to the first integer ($c_i = I^i_1$). The robot is initially located at its source $P(0)=S_i$. For the source we select the possible directions of movement of the robot. This is given by D=pt(x, y). Here (x, y) $=S_i$. The number of possible moves are given by size(D). We select the k^{th} move ($d=D_k$) in D where k is given by c_i mod size(D) as the first move of the robot. Hence

$move_A(x, y, c_i) = D_k$, (4)
D=pt(x, y),
$k = c_i$ mod size(D)

The pointer is updated to point to the next location ($c_i=I^i_2$). Now the robot is made to move in the same direction (d) till it reaches a crossing (Cross(x, y)=true) or it gets struck (Struck(x, y, d)=true). If either of these conditions is met, the robot has to make a decision regarding the next move. For this it first queries the total number of moves possible (D(x, y, d)) and then uses the integer pointed out by the pointer c_i to

decide which one of these moves is to be executed. Let the move be $move_B(x, y, d, c_i)$ which is given by k^{th} move in D ($d=D_k$) where k is given by c_i mod size(D). Hence

$$move_B(x, y, c_i) = D_k, \quad (5)$$
$$D=D(x, y, d),$$
$$k = c_i \bmod size(D)$$

The pointer is further incremented to point to next integer. In this mechanism the algorithm keeps iterating with decision being made at every crossing and when the robot is struck.

The motion of the robot may stop in either of two possible ways, either the robot reaches its goal, or the genotype is short of integers that is crossing after $c_i=I^i_n$ is to be processed. In case the stop is due to the termination of integer sequence, we reset the pointer ($c_i=I^i_1$) and allow the algorithm to proceed. This reset may however take place only once in the entire motion of the robot. In this manner the genotype may be finally converted into the phenotype. The complete algorithm is summarized in figure 2.

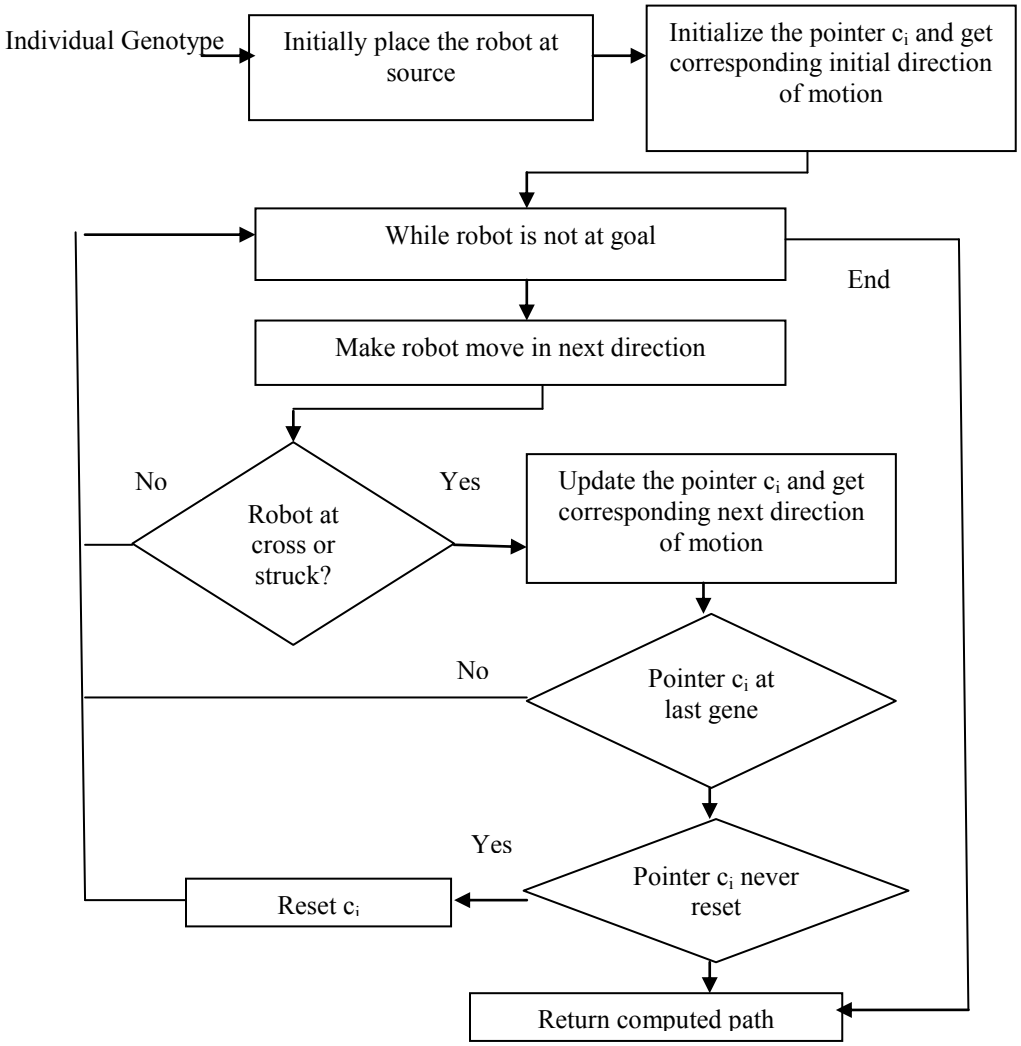

Figure 2: Conversion of a genetic programming genotype to robotic path phenotype. The algorithm takes an individual which is a collection or integers. The robot is moved in its computed direction. Whenever any cross is encountered, the pointed integer is consulted to get the next direction of motion, and the pointer is updated. The algorithm terminates if goal is found or when the individual terminates twice.

We study the conversion by an example. Let the map be as given in figure 3. The start position and goal of the robot is marked as shown in figure. Let the genotype be given by <2, 5, 3, 5, 2, 1, 3, 3, 3, 4, 5, 2, 2>. Initially the pointer is at 2. The possible moves are {down}. The algorithm selects the 2 mod1 i.e. the 0^{th} move which is down (Note that ordering starts from 0 and not 1). The robot marches down, until it reaches cross A. Here possible moves are {right, down}. The pointer is at 5. The robot selects 5 mod 2 i.e. 1^{st} move, which comes out to be down. The robot marches down till cross B. Here the possible move is {right}. Pointer is at 3. The move selected is 3 mod 1, i.e. right (0^{th} move). The robot comes to point C. Now possible moves are {right, down}. Pointer is at 2. The move selected is 2 mod 2, i.e. right (0^{th} move). Robot moves till cross D. Here possible moves are {up}, which is selected by pointer at 1.The robot travels up and reached cross E for which the possible move is only right. This is selected by pointer at 3. This makes the robot reach cross F with possible moves {right, down}. Pointer is at 2. The selected move is 3 mod 2 i.e. the first move or down. Robot moves in the same direction until it reaches cross G. The rest of the motion may be verified accordingly. At the end the robot will terminate its journey at goal, with some genes to spare.

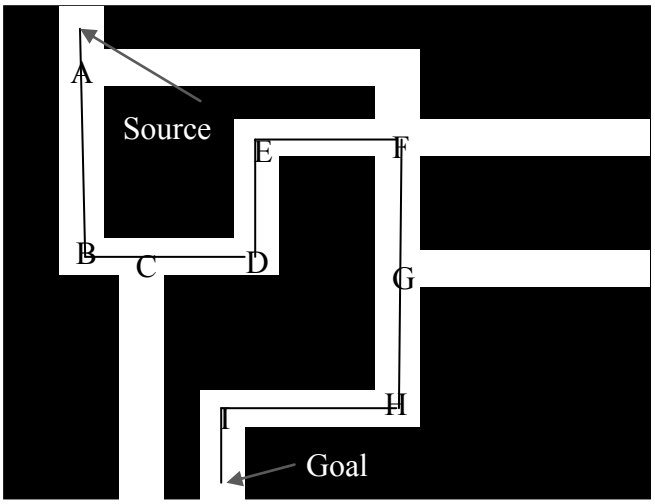

Figure 3: Sample map for conversion of genotype to phenotype. Figure is a sample map that shows the path used by a randomly generated robot for motion from source. The path traversed comes out to be Source → A → B→ C→ D→ E→ F→ G→ H→ I→ Goal

The next important task to be carried out in the implementation of the slave genetic programming is the genetic operators. We use two genetic operators for the algorithm, crossover and mutation. Crossover is applied to the top elite individuals. The number of individuals that undergo crossover is given by cross x popSize, where cross is the crossover rate and popSize is the total number of individuals in the population pool. These individuals are paired in groups of 2 and crossover is applied to each paid. Scattered crossover technique is used for the generation of two children from every pair of parents. The generated children replace the two poorest fitness individuals in the population pool. The mutation operator is applied to rest of the individuals in the population pool. All these left individuals undergo change in their genes, where the change is given by a uniform mutation. Here the gene value is changed to a random value with a probability given by the mutation rate.

The last part left in the implementation of the slave genetic programming is to have a fitness evaluation technique. We have already seen the mechanism of conversion of a genotype or an integer sequence into its phenotype or the robotic path. The fitness for any individual is measured by two factors denoting the individual and cooperative aspects of the overall path planning. The first factor for robot R_i (F^i_1) measures the fitness of the individual path. Using this measure, an individual tries to generate shorter and shorter paths which may ultimately contribute towards the overall optimality of all the robots. This fitness is measured by the total length of the path. In case the robot did not reach the goal, a penalty is added proportional to the distance between the last point touched by the robot and the actual goal location. This may be given by

$F^i_1 = l_i + \alpha \| P_i - G_i \|$ (6)

Here l_i is the total length of path, G_i is the goal and P_i is the last point touched by the robot. α is the penalty constant canned as the penalty constant for non-reach ability of goal.

By optimizing based on this objective it is highly likely that individual robots disregard the cooperation factor which is a major factor in co-evolution and enables different modules to develop characteristics to contribute well to the overall problem. This factor is served by the other factor of the fitness function (F^i_2). In this factor we look into the possible collisions between the robot with other robot, or the factors by which this path results in sub-optimal paths of other robots. The path of this robot is a part of multiple paths of the other robots. These paths are generated by the master as we shall see later. We select the best e individuals of the master. Each of these individuals specifies the paths of each of the robots. We simply replace the path corresponding to robot R_i in all these e individuals by the current individual. We note the difference in the fitness value, which is a measure of F^i_2. This may be hence given by

$$F_2^i = \sum_{j=1}^{e} \left(Fit(X_{j,R_i \to I^i}) - Fit(X_j) \right) \quad (7)$$

Here X_j is the j^{th} best individual of the master. $X_{j,R_i \to I^i}$ denotes best j^{th} individual of master with the path of robot R_i replaced by the current genotype I^i. The net fitness may be given by

$F^i = F^i_1 + F^i_2$ (8)

3.2 Master Genetic Algorithm

While the slave genetic programming does the task of generation of good paths for the individual robot, which have already taken the factor of cooperation into them, the task left is the evolution of the overall working strategy of the entire algorithm. This task is done by the master genetic algorithm, which decides how each and every robot is to move.

The first task is the individual representation of the master genetic algorithm. The individual is a set of n pointers, each pointing to some individual of the slave genetic programming. Let these pointers or the genetic algorithm individual be given by $<J_1, J_2, J_3, \ldots J_n>$. Here any pointer J_i points to one of the individuals of the i^{th} slave genetic programming (I^i) that represents the path of the i^{th} robot. This may be easily understood from figure 4.

The next task is the application of the genetic operators for the individual. Similar to the slave genetic programming, we use the genetic operators of crossover and mutation in this technique. The crossover mixes two parent individuals and generates two children. These two children replace the two weakest individuals of the genetic algorithm. A scattered crossover technique is followed in which every child randomly takes half of the pointers from first parent and the other half from the other parent. The mutation operator simply changes the genes of the individual. The gene is simply a pointer to some individual of the slave genetic programming. The mutation deletes an existing pointer and makes it point to some other individual in the genetic programming. Preference may be given to the newly created individuals of the genetic programming.

The last task to be carried out is the fitness evaluation. The fitness evaluation of the master genetic algorithm simply returns the average traveling time of each of the robots. The traveling time of the robot is measured by the time that the robot takes to reach its goal. However the master genetic algorithm needs to ensure that all the paths are traversed without any kind of collision between the robots. For this a simultaneous simulation of all robots is done, knowing that the various robots have different speeds. It is further possible that some of the robots do not reach their goal. For both these aspects a penalty is added to the fitness function. The resultant fitness function of the master genetic algorithm hence becomes.

$$F = \left(\sum_{i=1}^{n} T_i + \alpha \| P_i - G_i \| \right) + \beta.C \quad (9)$$

Here T_i is the time the robot R_i reaches its goal G_i, P_i is the last point touched by the robot in its journey, α and β are penalty constants (β > α), C is the time first collision occurs (0 is no collision occurs). α is called as the penalty constant for non-reach ability of goal. β is called as the penalty constant for collision.

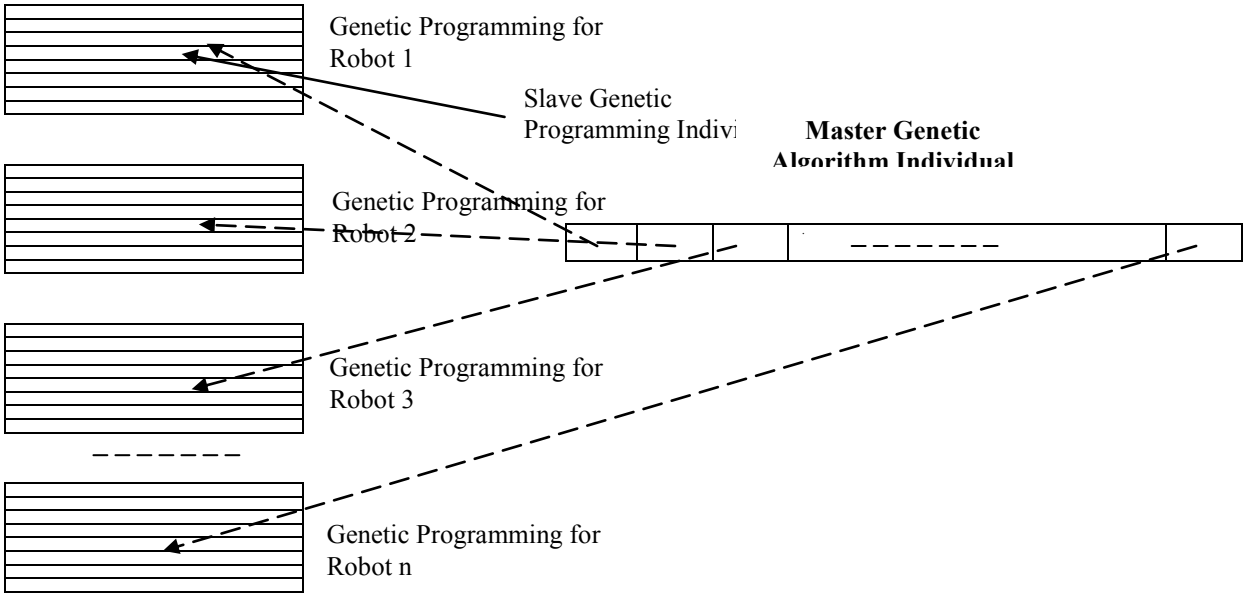

Figure 4: Individual representation of master genetic algorithm. Every individual of the master genetic algorithm points to an individual of the slave genetic programming. Figure represents one individual of the master genetic algorithm

The penalty constant for collision (β) is usually kept higher than the penalty constant for non-reach ability of goal (α). This is to encourage collision free movements. It would be better not to have collisions that not to have a robot reach its goal. The path may be feasible only when both these factors are zero, but characteristic positions of robots in the map may result in scenarios where it is not possible to have all the robots have a collision free journey to their goals.

3.3 Algorithm Outline

We have already discussed the individual master and slave evolutionary algorithms. In this section we briefly give a combined picture of these two algorithms. The complete algorithm is presented in figure 5. It may be easily seen that the algorithm is iterative in nature, where both the evolutionary algorithms run in parallel. They hence share a common stopping criterion. A unit iteration of all the evolutionary algorithms is executed in parallel. This gives all of these a chance to evolve their population as per the changing scenarios and findings. It may be noted that all the various evolutionary algorithms do not run in isolation, but are rather dependent on each other. A boost in the fitness value of the master genetic algorithm by selection of some individual from some slave genetic programming may mean the slave genetic programming being re-directed to produce different kinds of individuals. Similarly if some new individual produced by one of the slave genetic programming results in improvement of the overall path being controlled by the master genetic programming, it may redirect the master to produce more individuals of the same kind. We have already discussed about the factor of cooperation between the various slave genetic algorithms.

4. Waiting for Robot

The algorithm framework presented in section 3 is self-sufficient to carry planning of multiple robots, but it fails on a number of scenarios for which we introduce the concept of waiting for robot. Consider the case

where a slower robot is following a higher speed robot. This scenario was discussed earlier and was presented in figure 6 to ease the rest of the discussion. In this scenario the path of the two robots are shown. It would be better if the slower robot would have waited at point A for the faster robot to move. In such a case the slower robot would have followed the faster robot which naturally means a better moving strategy minimizing the collective motion time of all the robots. In reality with multiple criss-crosses and multiple robots there can be complex scenarios where one robot waiting for another robot may be fruitful.

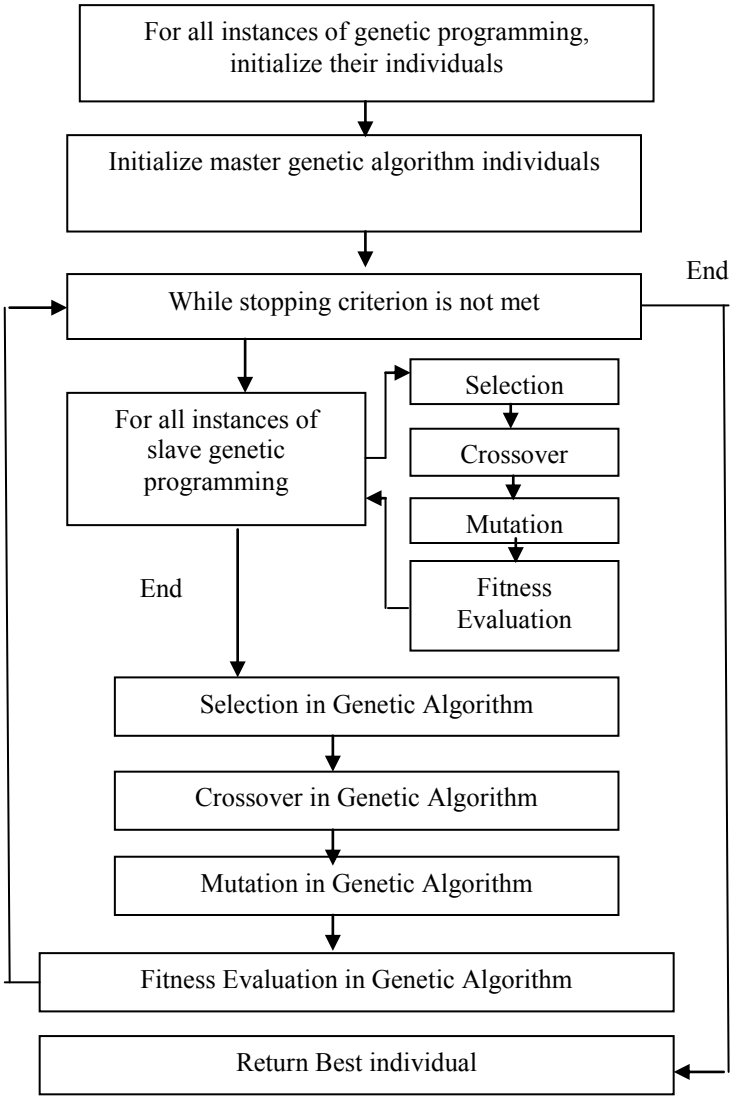

Figure 5: The basic evolutionary algorithm

We informally state that wait may only be performed at the crossing (cross(x, y)=true); more precisely at a unit step from crossing, since we are to leave the crossing free for the moving robot. Waiting at any other location other than crossing may be equivalent to waiting at some crossing. Further the wait must always end with some robot crossing the point of crossing for which the robot is waiting. It is natural that there is no use waiting more than that, unless the robot is actually waiting for another robot after which it plans to move. This would be equivalent to the robot waiting for the second crossing robot with no necessary relation with the first crossing robot. It may be hence noted that the concept of waiting for robot at crossing is important and may further boost the performance of the model presented in section 3. We hence formally define the concept and modify the algorithm presented in section 3 as per its requirements.

We defined paths possible at any crossing (Cross(x, y)=true) at location (x,y) and direction d be given by D(x,y,d), where D(x, y, d) $\subset$ {left, right, up, down}. Now the robot may be additionally asked to wait at some crossing and hence the option is added to D(x,y,d). The modified value is given by $D_{mod}(x, y, d) = D(x, y, d) \cup \{wait\}$. Here wait signifies that the robot is being asked to wait for a robot. size(D_{mod}) denotes the total number of elements in the set or the total number of moves possible. Consider the genotype pointer is pointing towards any location c_i at any instance of time. Now the move made by the robot $move_B(x, y, d, c_i)$ is given by

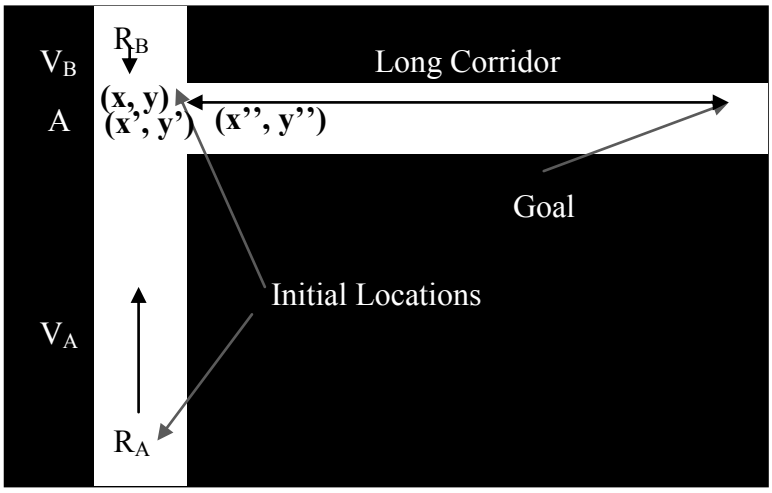

Figure 6: The problems with planning with multiple speeds. The figure shows two robots R_A and R_B moving with velocities V_A and V_B ($V_A >> V_B$). If R_B enters the corridor first, RA will have to follow for the rest part of the journey with reduced speeds. Optimal path has R_A entering before R_B.

$$move_B(x, y, c_i) = D_k, \quad (10)$$
$$D=D_{mod}(x, y, d),$$
$$k = c_i \bmod size(D_{mod})$$

Similarly the first move of the robot is modified to work over D_{mod} in place of D. The restriction however here is that the first move cannot be a wait. This is because it is not necessary that the robot is initially located at a crossing, which is a requirement for the wait move. The first move $move_A(x, y, c_j)$ is hence given by

$$move_A(x, y, c_i) = D_k, \quad (11)$$
$$D=pt(x, y),$$
$$k = c_i \bmod size(D_{mod})$$
$$move_A(x, y, c_j)=\{wait\} \ \forall \ j<i.$$

More precisely the robot has to wait just before the crossing and not on the crossing. In case it waits on the crossing, it would prohibit any robot to pass by including the robot for which it is waiting to pass by. Hence the decision towards whether the robot has to wait or not needs to be made before the robot enters into crossing. This may not be the immediately preceding step of crossing (as robots have fractional velocities) but the step after which the robot enters anywhere within the crossing region. We define the penultimate position of the robot (x,y) such that any step further from this in direction d would result in the robot entering into the crossing area. Let this be given by pen(x,y,d). Now as soon as any robot enters into pen(x, y, d), it queries the next crossing cross(x', y', d) and gets the move which it would be expected to make $move_B(x', y', d)$. Here (x', y') is the location of the next crossing (in this query the pointer c_i is not altered). In case this move comes to be wait (or $move_B(x', y', d)=\{wait\}$), the robot does not move and waits at (x, y) and the pointer c_i is incremented (or the wait move is already executed). Note that after the robot is allowed to move, it would reach the crossing and the next move would be made as per the new integer pointed by c_i. In case the move does not come to be wait, the robot continues its motion.

Further when a robot is waiting, it must always wait for some robot that is guaranteed to cross from the crossing of wait. This ensures that we know exactly about when the robot may be allowed to resume its journey. Let (x, y) be the location of a robot R_i penultimate to crossing. Let (x',y') be the location of the crossing. Further let (x'', y'') be the position next to crossing (outside the crossing area) where R_i would be moving after wait is over. Since we know the complete genotype of motion of R_i, we can easily compute its entire path, excluding the waits which are dependent on the movement of the other robot and cannot be computed in isolation. We search for robots R_j ($j \neq i$) which are presently moving and would be found at location (x'', y'') at some instance of time ahead but not after R_i reaches its goal if allowed to move without waits. Again it be noted that complete path of R_j is already known excluding the waits. If we get some R_j such that R_j is moving and R_j crosses (x'', y'') at some time ahead but not after R_i reaches its goal if allowed to move without waits, we state that the robot R_i will wait for R_j ($R_i \rightarrow R_j$). The current state of R_i is labeled as waiting, and all it is not allowed to move further. Only after R_j crosses location (x'', y''), the robot R_i is allowed to move. In case no R_j ($j \neq i$) satisfies the criterion, the wait of R_i is cancelled and its normal motion continues.

It is important to put the check that R_j should be moving to escape from deadlock. If this condition is not checked it would be possible to have a situation $R_i \rightarrow R_j$ and $R_j \rightarrow R_i$, in which case both R_i and R_j are not moving and waiting for each other. More generally it would be possible to have a scenario $R_a \rightarrow R_b \rightarrow R_c$, $\rightarrow R_d, \ldots \rightarrow R_a$, which is a deadlock.

In this manner we may be able to model multiple deadlock free scenarios with multiple robots waiting in any complex or simple manner. Hence by this introduction of wait in the problem modeling, we are able to generate more flexible movement strategy of the multiple robots.

5. Memory based Local Search

The model of the algorithm discussed in section 4 may take a reasonably long time to carry out the task of optimization. It is a common technique to assist the evolutionary process by some heuristic means. In this algorithm we do this by storing a lookup table at a dedicated memory and use the same for optimization. The purpose of the lookup table is to store information about shortest paths between crosses. In case the path of any of the robots happens to be larger than the path stored in this lookup table, we may easily replace it by the smallest path already known. In such a case the robot would reach the goal in a much smaller path and if the modified path has no collisions, the overall optimality of the solution may be enhanced. The lookup table enables the robot to quickly attain the optimal path, while it may be reasonably away from the same.

Another use of the lookup table is to enable cooperation amongst the robots. If one robot finds an optimal path between any pair of crosses, it must be able to share the same with the other robots, to enable them benefit from the finding. By making a lookup table if a robot finds the smallest path between any pair of crosses (as per the current exploration), it stores it into the lookup table which may be accessed by all the other robots if they need to go between the same pair of crosses in their journey.

The lookup table or memory is supposed to store smallest paths between crosses. Suppose that there are v crosses (cross(x, y)=true)in the system. We select a total of η ($\eta \leq v$) crosses randomly from the available v crosses that participate in the lookup table. η decides the size of the lookup table. The lookup table is a data structure that stores the smallest path between any of these crosses, where each cross is defined by the robot position (x, y) and direction d . Let the selected crosses be (V_1, V_2, …. V_η). The lookup table stores the minimal path length Val(V_i, V_j) between from cross V_i to cross V_j. It further stores the actual set of genotype integers that result in the generation of smallest path P(V_i, V_j) from cross V_i to cross V_j.

One of the tasks associated is the initialization and update of the lookup table. Initially we set

$$Val(V_i, V_j) = \begin{cases} \text{infinity} & V_i \neq V_j \\ 0 & V_i = V_j \end{cases} \quad (12)$$

$P(V_i, V_j)$=NIL (13)

As we proceed with the algorithm, we get multiple paths on converting the genetic individual genotypes to corresponding phenotype paths. For the converted paths we check all pairs of crosses for possibility of shorter paths. Hence

$Val(V_i, V_j) = \min\{ length(P'_{V_i \to V_j}), Val_{old}(V_i, V_j)\}$ $\forall$ V_i and V_j in generated path P, V_j comes after V_i (14)

Here $Val_{old}(V_i, V_j)$ is the previous value and $Val(V_i, V_j)$ is the updated value of the data structure. $length(P'_{V_i \to V_j})$ is the length of path between V_i and V_j in P'. Similarly the path needs to be modified. Hence we get

$$P(V_i, V_j) = \begin{cases} P'_{V_i \to V_j} & length(P'_{V_i \to V_j}) < Val_{old}(V_i, V_j) \\ \text{no change} & \text{otherwise} \end{cases} \quad (14)$$

In this manner the lookup table is ready and keeps getting updated as newer paths are found. The last part left is to use the table as the local search of the individuals. For any path of the slave genetic programming we apply an additional operator reduce that attempts to shorten the path of the robot with a probability red. If the probability red is very low, there is almost no replacement and the factor of local search is almost lost. In case the factor is very high, it is likely that the individual slave genetic programming algorithms keep developing characteristics as per the characteristics in the lookup table, and the overall algorithm might get converged at some local minima.

For any path in its genotype form which needs to be reduced, we randomly try to select pairs (V_i, V_j) in the path P' of the robot, such that $P'_{V_i \to V_j} > Val(V_i, V_j)$. For this path we replace the genotype integers from V_i to V_j by $P(V_i, V_j)$. Suitable genes may be added at the two ends of the path. The previous path P' ($S \to V_{i-1} \to V_i \to V_j \to V_{j+1} \to G$) hence becomes ($S \to V_{i-1} \to P(V_i, V_j) \to V_{j+1} \to G$) where S is the source and G is the last pointed touched by the robot. If no V_i, V_j satisfy the criterion, that is the path is optimized as per the lookup table, then no actions take place by this operator. The complete operation may be repeated over a number of times, as it is likely that the generated path may further be reduced by the lookup table.

In this manner the application of this additional operator to the genetic programming individual may result in an enhanced optimization to the problem, considering the reasonably complex nature of the problem. This plays a major role in giving good results early.

6. Results

The algorithm was tested on a simulation tool built in JAVA. The complete algorithm was coded in multiple modules which included the master genetic algorithm, slave genetic programming, and the operations over the individuals of these two evolutionary techniques. A simulation tool used the genotype of the evolutionary technique to simulate the complete system and generate the necessary phenotype paths. This was done separately for both the master and the slave evolutionary techniques. A JAVA Applet based GUI client was built that used multi-threading to show the optimal motion strategy of all the robots, as calculated by the algorithm. The maze-like map was given to the algorithm in a form of a BMP image. The paint utility was used for the generation of the map.

Maze-like maps were used for all the testing experiments. The robots could only move in four directions. Hence in these maps it is not necessary that the optimal path lies close to the straight line joining the source and the goal, which would have been the case for robots which can move in multiple directions. If the motion is to be made from one corner of a square to its diagonally opposite corner, in these maps it makes no difference whether the journey is made close to the diagonal joining the corners, or using the vertical and horizontal edges. The distance of travel (which would come out to be equivalent to the Manhattan distance) would be the same. For the same reasons the map used for testing purposes had a number of turns such that the final optimal path between any source and goal is large and complex enough.

The algorithm was first simulated for 2 robots. The intention was to check the path optimality of both these robots, as well as their cooperation to figure out a collision-free path. In order to ensure a complex overall path, the two robots were given diagonally opposite source and goals. Further to make collisions move likely, the source of the first robot was made the goal of the other, and vice versa. The first robot was initially located at (0, 2) and had to reach a goal (24, 23). The other robot was initially placed at (24, 23) and had to reach a goal of (0, 2). The speed of the two robots was 0.5 and 0.3. The simulation was carried out for a total of 20 iterations. The master genetic algorithm had a population size of 400. The mutation rate was fixed to 0.1. The top 40% individuals were used for the crossover operations. The rest individuals were generated by mutation operation. The genetic programming had a population size of 250 individuals. The mutation rate was 0.1. Here the top 20% individuals were used for crossover, the rest being generated by mutation. The individual had a size of 25 real numbers or genes. The penalty constant for not reaching goal was 100, and the penalty constant for collision was 1000. The algorithm, upon simulation generated the paths, which were displayed. The parameters and results are summarized in table 1. The motion of the robots along with time may be seen in video 1. The general motion of the robots is also given in figure 7 for ease of discussion.

Table 1: Summary of situation and results for simulation with 2 robots

S. No.	Factor	Robot 1	Robot 2
1.	Source	(0, 2)	(24, 23)
2.	Goal	(24, 23)	(0, 2)
3.	Speed	0.5	0.3
4.	Reached Goal	Yes	Yes
5.	Collision	No	No
6.	Time	142	383
7.	Path Length	66	115

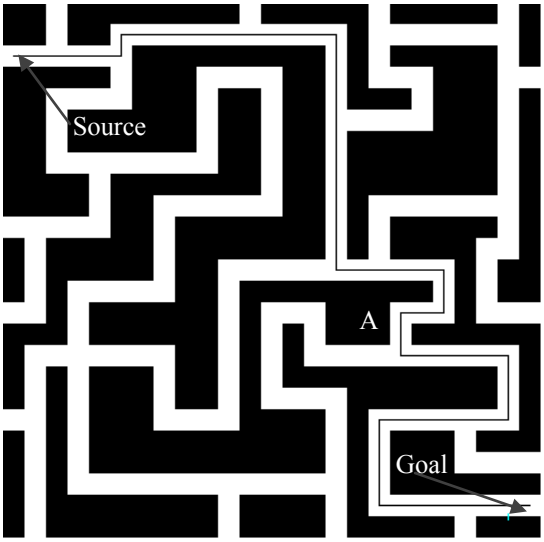

Figure 7(a): Path traced by robot 1 for simulation with 2 robots

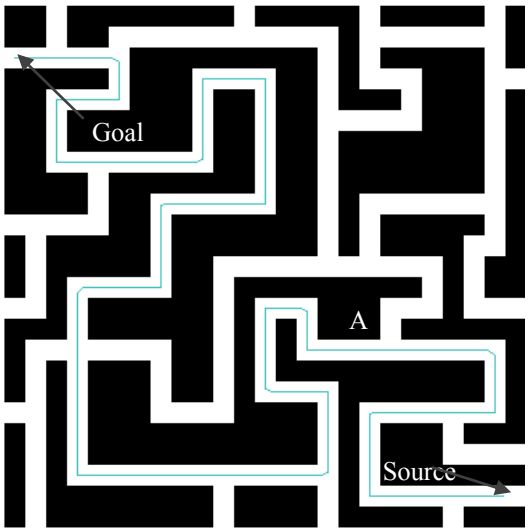

Figure 7(b): Path traced by robot 2 for simulation with 2 robots

It can be easily seen that as per the set locations, the collisions between the robots was natural. In this case the robots made use of the wait feature and hence first robot waited for the second robot at point A (shown in figure 7). This enabled the two robots to move to their own goals. It may again be easily seen that wait can only be applied when the robots do not go in opposite directions. Suppose a robot is waiting at a cross, on the way where the other robot wants to go to. It is natural that there would be a collision, since the moving robot would collide with the waiting robot. Further we observe that the speeds of the robots play a big role in deciding the path. The paths for same set of parameters would turn out to be different if the

velocities are changed. The velocities decide the possible points of collision, to which the colliding robots cooperate to find a collision free strategy.

Table 2: Summary of situation and results for simulation with 3 robots

S. No.	Factor	Robot 1	Robot 2	Robot 3
1.	Source	(0, 2)	(24, 23)	(23, 0)
2.	Goal	(24, 23)	(0, 2)	(1, 24)
3.	Speed	0.4	0.7	0.8
4.	Reached Goal	Yes	Yes	Yes
5.	Collision	No	No	No
6.	Time	166	167	71
7.	Path Length	66	117	57

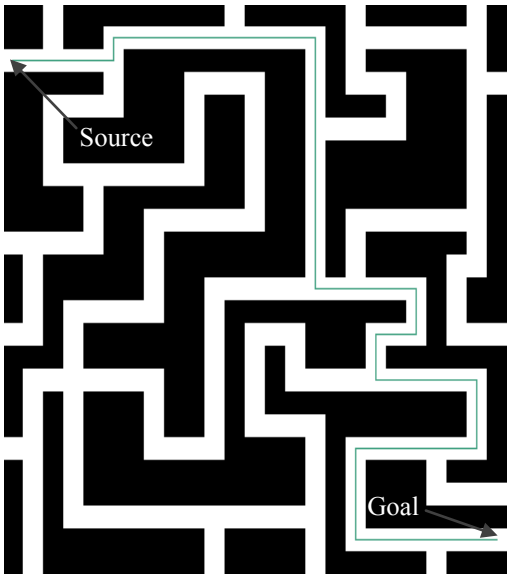

Figure 8(a): Path traced by robot 1 for simulation with 3 robots

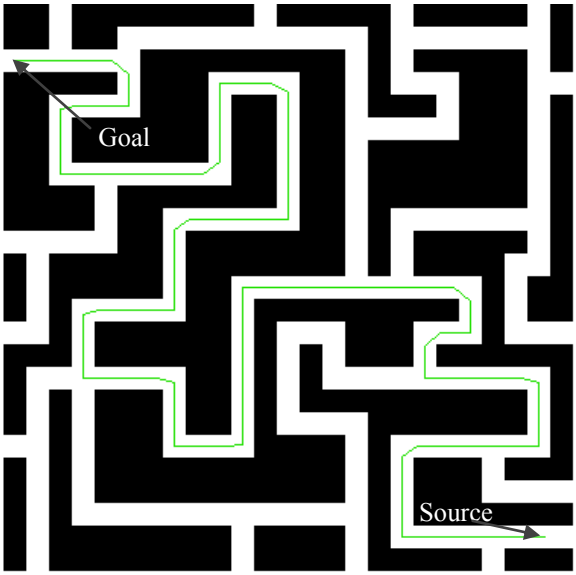

Figure 8(b): Path traced by robot 2 for simulation with 3 robots

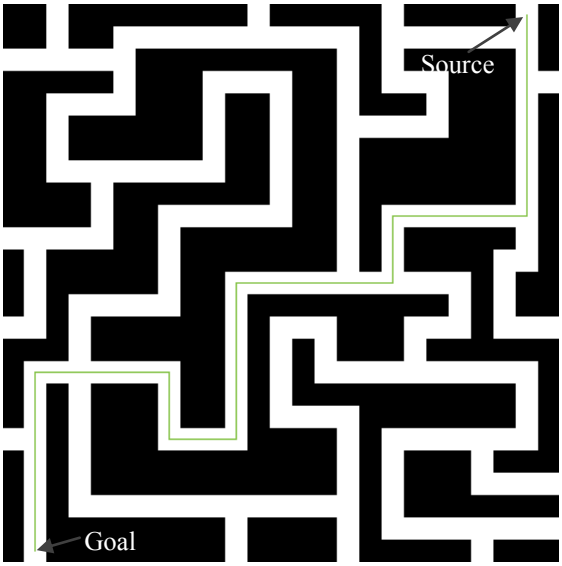

Figure 8(c): Path traced by robot 3 for simulation with 3 robots

The other execution was carried over the same map. To test the algorithm performance in higher number of robots, we added another robot to the simulation. So there were a total of 3 robots that tried to find their goal from the starting initial positions. The evolutionary parameters were kept same as the previous run. The source of the first robot was (0, 2), while its goal was (24, 23) and its speed was 0.4. The second robot had the source as (24, 23), goal as (0, 2), and speed 0.7. The third robot had source (23,0), goal (1, 24) and a speed 0.8. The simulation was carried out and the results were displayed using the GUI tool. The parameters and results are summarized in table 2. The motion of all the robots is given in video 2. The results are further drawn in summarized form in figure 8.

Table 3: Summary of situation and results for simulation with 4 robots

S. No.	Factor	Robot 1	Robot 2	Robot 3	Robot 4
1.	Source	(0, 2)	(24, 21)	(23, 0)	(1, 24)
2.	Goal	(10, 24)	(23, 0)	(1, 24)	(24, 21)
3.	Speed	0.6	0.8	0.7	0.8
4.	Reached Goal	Yes	Yes	Yes	Yes
5.	Collision	No	No	No	No
6.	Time	112	57	81	88
7.	Path Length	67	46	57	71

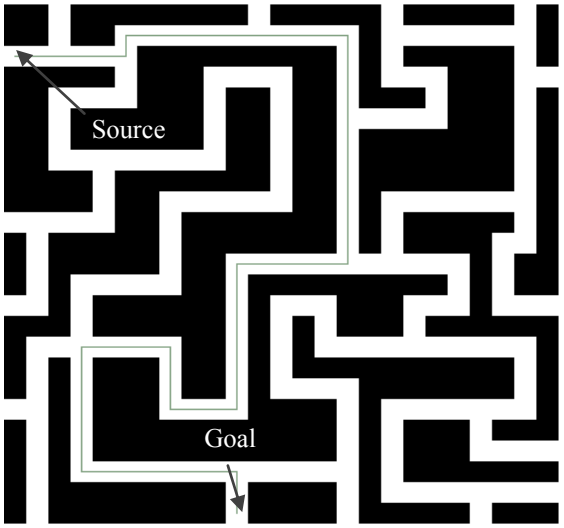

Figure 9(a): Path traced by robot 1 for simulation with 4 robots

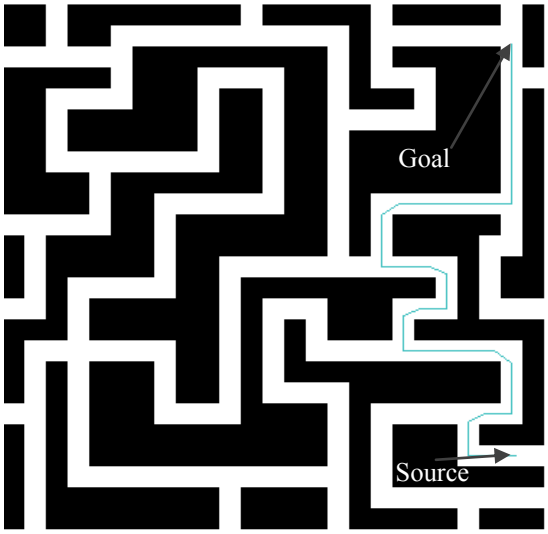

Figure 9(b): Path traced by robot 2 for simulation with 4 robots

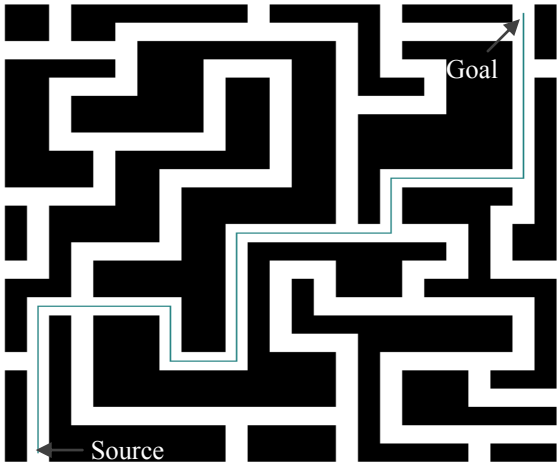

Figure 9(c): Path traced by robot 3 for simulation with 4 robots

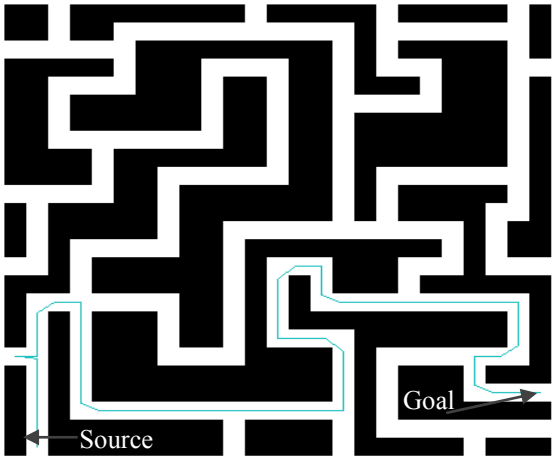

Figure 9(d): Path traced by robot 4 for simulation with 4 robots

Here as well we see that every robot tried to simultaneously cater to two needs, to find an optimal path, and to find a collision free path. It is evident that there are limited options or paths, and hence an optimal path may not be collision free and vice versa. An overall solution is hence difficult to obtain. It may further be seen that speed is a major factor. Consider the point A as shown in figure 9. If robot 1 traveling from top left corner was traveling faster, it would have resulted in some collision with robot 2 traveling from bottom right corner. This would have resulted in either of the robot waiting, or both of them force to recalculate their paths.

We further test the scalability of the algorithm with more robots. The next simulation involved 4 robots. The initial locations of the four robots were (0, 2), (24, 21), (23, 0), and (1, 24). The goals were specified as (10, 24), (23, 0), (1, 24), and (24, 21). The speeds were set to be 0.6, 0.8, 0.7, and 0.8. The evolutionary parameters were kept same as the previous run. The result in this case is given in video 3, and the same is summarized in figure 9. The parameters and result statistics is given in table 3.

It may be seen that though the modeling scenario was reasonably complex with multiple robots that were supposed to find their way out to the goal, the algorithm was able to generate these paths for all robots. All the robots could reach their goals in a reasonably optimal path. It may be possible for better paths to exist or the individual robots, but this may lead to collisions and hence needs to be avoided. The factor of optimization is the average running time of the robots, which is optimal in this case. It may be emphasized that every addition in robot means an immense increase in possibilities at the coordination level. Two robots may only need to ensure that they somehow do not collide, but multiple robots can interact in plentiful ways. We may avoid collision between two robots by some change in their paths, but the changed paths might result in more collisions with any of the other robots. Hence a very complex interaction amongst robots makes the problem difficult.

The last experiment was done using 5 robots. The locations and the speeds of the robots were changed. The source of first robot was (2,0) and goal was (10, 24). The speed of this robot was 0.6. The second robot had a source of (24, 21) and goal of (23, 0). The speed of the robot was 0.8. The third robot was at (24, 3) and was supposed to move to (0, 19). The speed was kept as 0.7. The fourth robot had a source (0, 19) and goal (24, 21). The speed was 0.8. The fifth and the last robot was had a source (11, 0), goal (0, 14), and speed 0.2. The simulation was carried out and the final results were recorded. The movement of all the robots is shown in video 4. Figure 10 is the equivalent figure for the same simulation. The parameters are results are given in table 4.

Table 4: Summary of situation and results for simulation with 5 robots

S. No.	Factor	Robot 1	Robot 2	Robot 3	Robot 4	Robot 5
1.	Source	(2, 0)	(24, 21)	(24, 3)	(0, 19)	(11, 0)
2.	Goal	(10, 24)	(23, 0)	(0, 19)	(24, 21)	(0, 14)
3.	Speed	0.6	0.8	0.7	0.8	0.2
4.	Reached Goal	Yes	Yes	Yes	Yes	Yes
5.	Collision	No	No	No	No	No
6.	Time	115	69	81	83	149
7.	Path Length	69	56	57	67	30

It may again be verified that despite heavy complexity, the algorithm could figure out a collision-free motion strategy for the motion of the robots. It needs to be again noted that motion of a robot by a path completely blocks the path. For the entire duration that the robot occupies that path, no robot would be able to move in any direction without collision. Hence the number of alternate paths plays a major role in the system. Too many robots with less number of alternative path might mean no path may be possible for some or the other robot that reaches the goal without collision. Either the robot may not reach goal at all, or it may reach the goal with collision. In some cases the robot might have to take too many unnecessary (and in some cases redundant) paths before moving on a path that takes it to goal. Hence there is a limitation of the map to the maximum number of robots that can move in. For the same reasons we do not experiment with larger number of robots. Hence we see that multiple robots may be optimally moved into the map.

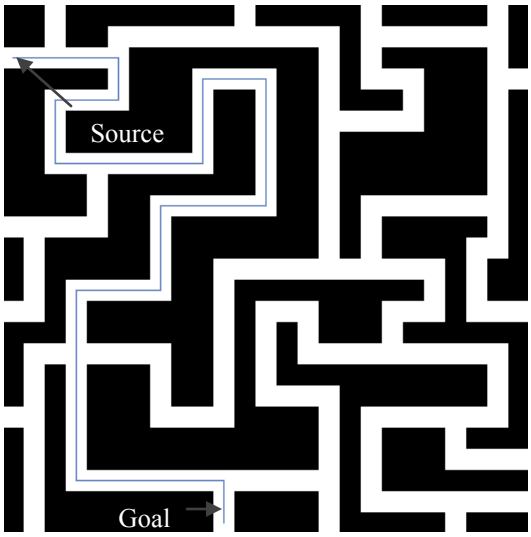

Figure 10(a): Path traced by robot 1 for simulation with 5 robots

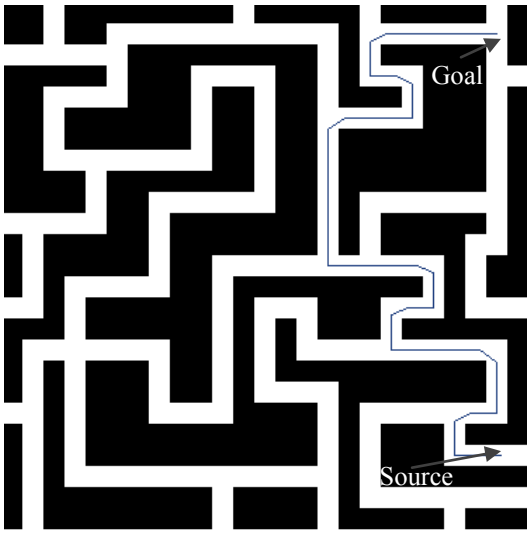

Figure 10(b): Path traced by robot 2 for simulation with 5 robots

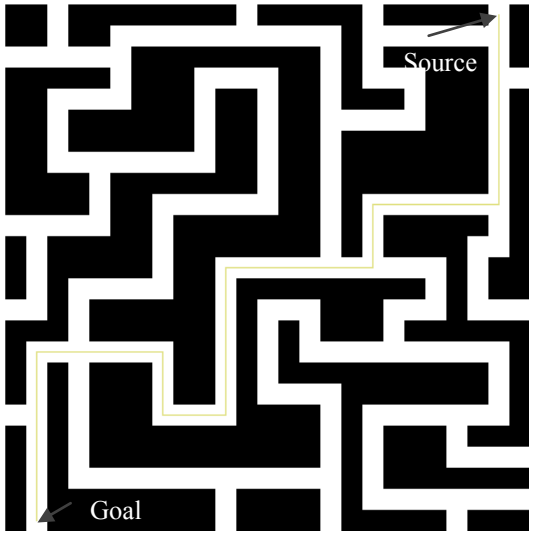

Figure 10(c): Path traced by robot 3 for simulation with 5 robots

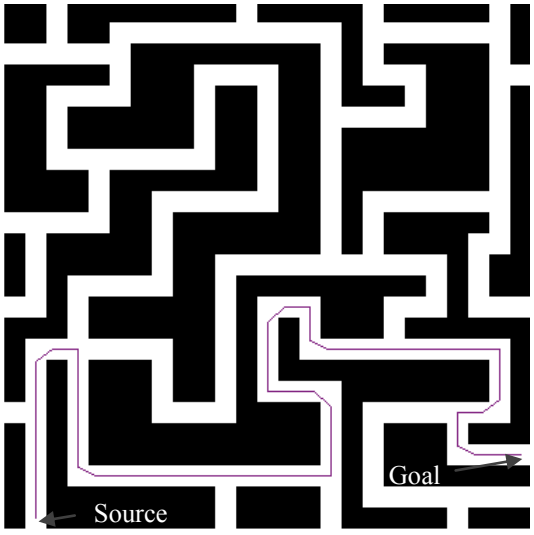

Figure 10(d): Path traced by robot 4 for simulation with 5 robots

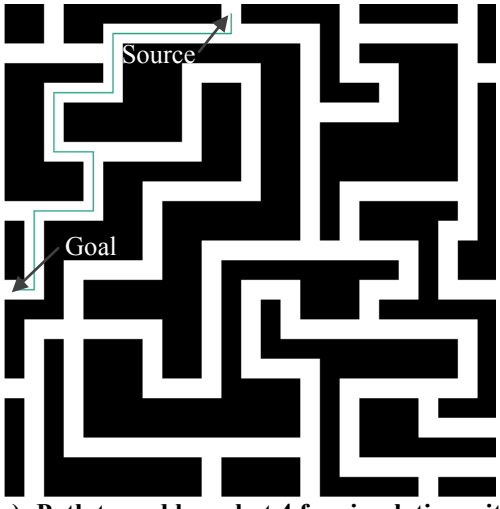

Figure 10(e): Path traced by robot 4 for simulation with 5 robots

The next scenario we generate is a very simple one. Here we make two robots move in opposite directions, so as to make them reach a common goal. In this scenario we aim to test the working of the wait feature of the algorithm. It is evident that it is not possible to move the robots in a manner that collision can be avoided without any robot waiting. The map has one crossing and one of the robots needs to wait so that the motion may terminate without collision. The simulation for this case is given in video 5 and figure 11. It may be seen that one of the robot waits for the other. This again proves the usefulness of the wait feature of the algorithm.

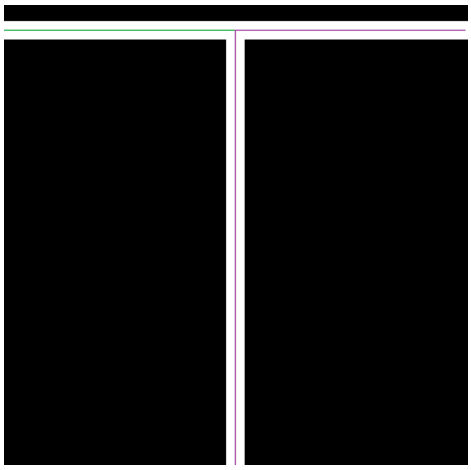

Figure 11: Path traced by two robots using wait for robot feature

Apart from the designed algorithm and the wait feature, the other module we designed was the local optimizations using a lookup table. We stated that this lookup table was important for a robot to use the findings of the previous generations, as well as the other robots. This enables swift motion towards the optima. We executed the same scenario with 4 robots without the lookup table. We observed that the results were highly sub-optimal. Many times robots were struck at some loops, where they returned back at a point they had visited earlier. The results with the execution of the program without lookup table are given in table 4. The paths of the various robots may be seen in figure 12. Video 6 shows the motion of the robots. This shows that the module of lookup table was effective and played a vital role in making the algorithm.

We further study the execution time of the algorithm for the increase in number of robots. Based on our understanding of the algorithm it may be easily seen that the execution time should increase with increasing number of robots. This is because of the fact that for every increase in robot, a Genetic Programming instance is created and executed. Also the simulation plays a keen role in deciding the execution time. If the path to goal for all the robots is very simple, they would all attain their goals and the complete simulation would stop. On the other hand if there is a collision in the path of the robot, or it does not reach the goal till the end, there is a lot of computation that is performed. This is because for a very long time (until the genomic length is completely iterated) some or the other robot keeps wandering at the map. The increase in number of robots increases the possibility of collisions to a great extent and hence multiple paths need to be generated and tried. All this consumes a lot of time and execution time further increases. For the same reasons simulations having slower moving robots would take more computation time. We take the same map as discussed in the above discussions. We plot the execution time for the various numbers of robots. This is shown in figure 13. It can be easily seen that there is more than linear increase in computation time for increase in robots. This is because of the cooperation factor as discussed.

We further extend the analysis to the stud y of the optimization of a single scenario. We study how the optimization time changes along with generations. Every generation has a different execution time. We may infer that the execution time depends upon the path of the robots. Every individual of both the genetic programming and genetic algorithm represents paths which need to be simulated multiple times. Longer paths would mean longer simulation time and hence the total time of execution of that generation would increase. As we move along with generations, paths start getting close to optimal solutions. The lengths of the paths reduce. More robots start to reach their goal, which means less wandering in the maps. Hence in

general the execution time for higher generations is smaller than the execution time for the smaller generations. However the evolutionary operators may sometimes result in too fit or too unfit individuals. In case of the former, there is a sharp decrease in execution time, while the latter results in a sharp increase of the same. The sharp increase or decrease may be averaged out to some degree by the other individuals in the population pool. Figure 14 shows the graph for the execution time for various generations. The general trend has been plotted as a dashed trend line. The simulation of 4 robots was used.

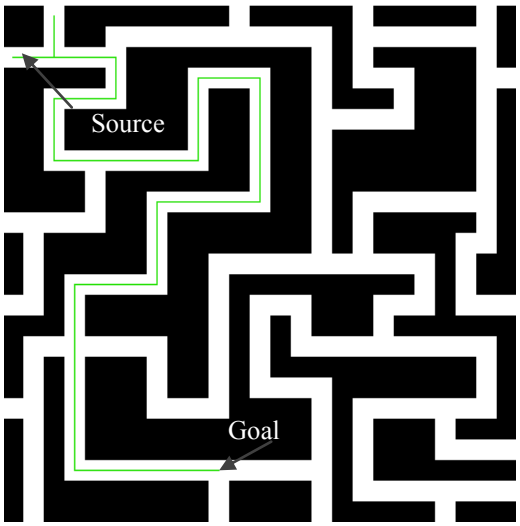

Figure 12(a): Path traced by robot 1 for simulation with 4 robots without memory based optimization

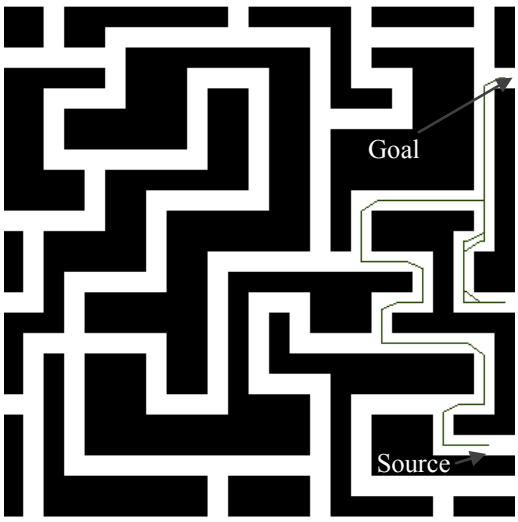

Figure 12(b): Path traced by robot 2 for simulation with 4 robots without memory based optimization

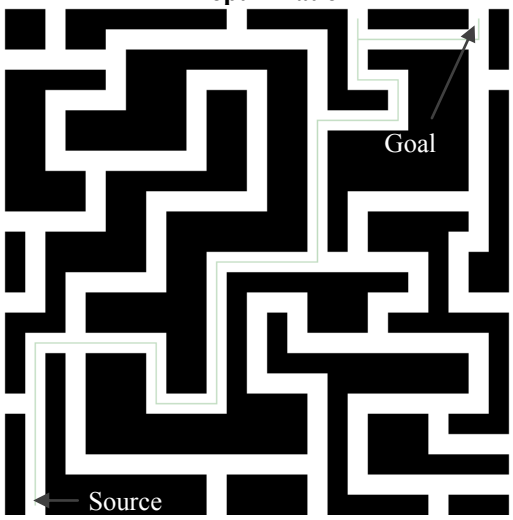

Figure 12(c): Path traced by robot 3 for simulation with 4 robots without memory based optimization

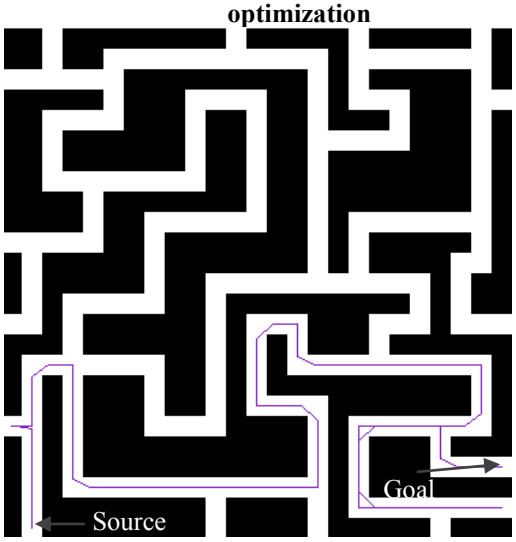

Figure 12(d): Path traced by robot 4 for simulation with 4 robots without memory based optimization

7 Conclusions

In this paper we attempted to solve the problem of multi-robot motion planning. The modeling scenario had a maze-like map where the different robots were initially located at distinct places and were given their own goals that they were supposed to reach. We further assumed that each robot moves with its own speed. The algorithm framework made use of co-evolutionary genetic programming. The task of planning was performed at two levels. At the first level a linear representation of Genetic Programming was used. The individual in this case consisted of instructions for movement whenever a cross is encountered. There was a

different instance for every robot. The other level consisted of a genetic algorithm instance. This algorithm selected the individuals from the genetic programming and tried to generate a combination such that the overall path of all the robots combined is optimal. An individual of this level had pointers pointing out to genetic programming instances. The resultant algorithm could solve the proposed problem, but there were a number of scenarios that it could not solve. For this we included the feature of wait for robot, where a robot may be asked to wait before some crossing till a robot passed the same. In this manner robots were capable of escaping from reasonably complex scenarios in a cooperative manner so as to reach their own goals. Many collision prone scenarios could hence be controlled, making collision free planning possible. We further saw that this feature resulted in generation of optimal paths. This algorithm also solved the purpose, but the resulting problem became highly complex. This necessitated the need of some local search strategy, which we included into the algorithm in form of a lookup table. Here, as the algorithm proceeds, optimal paths between any two crossings are monitored and stored. The path of any individual may hence be modified, and it may be made to follow by the pre-computed optimal path, rather than the path as per its genes. This feature hence made it possible to generate optimal paths reasonably early in highly complex scenarios.

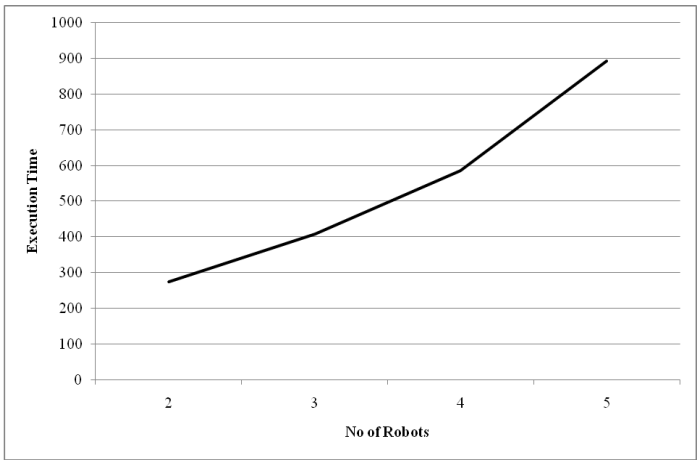

Figure 13: Execution time v/s number of robots

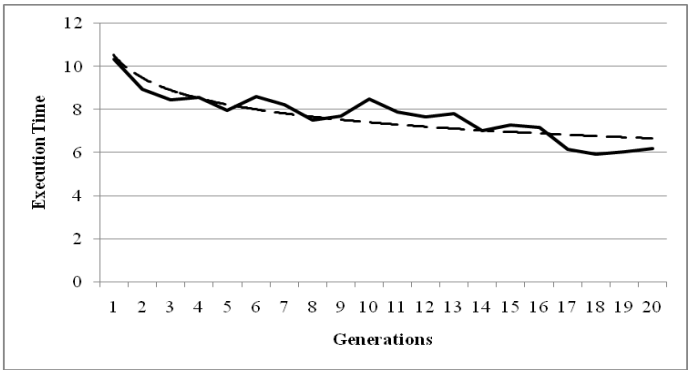

Figure 14: Execution time v/s generations

The algorithm was developed as a JAVA software, which could read BMP images as maps. The various parameters of the evolutionary algorithm as well as the problem specifications could be defined as well. A number of executions of the algorithm were carried out with different number of robots. In all scenarios we saw that the algorithm could figure out a collision free path for all robots from the source to the goal. The paths seemed optimal in length and travel time. The different runs were done with different positions of the robots. This ensures that the algorithm generates optimal results to different modeling scenarios. We further did experimentation of the algorithm with the same scenarios without the memory based optimization. It was seen that the algorithm could not perform well. The resultant paths were not optimal. Further the effectiveness of the wait for robot feature was experimented by a simple scenario, and the resultant path

had a robot to wait for the other robot. The experimental results prove that the algorithm could effectively solve the proposed problem as per the mentioned modeling scenarios in a mix of easy to complex scenarios.

In all the cases presented, it was a common observation that the speed of the various robots played a big role in deciding their paths. The different speeds meant different points of likely collision, for which some alternative strategy had to be built. Further it was observed that the later generations of the algorithm were a lot quicker. This was due to the optimized nature of the individuals at those times. The increase in the number of robots implied an increase in the complexity which increased the execution time.

The presented approach however has some limitations as well. The biggest limitation is the optimization time of the algorithm. This algorithm may hence not be usable for many real time scenarios. It may also be possible that the real world scenario may have dynamic maps, which the present approach does not consider. The other limitation of the algorithm is that the modeling scenario gets very complicated for the addition of large number of robots. In lust to have an optimal overall path with a central planning technique, we put a restriction on the maximum possible number of robots that may be computed in real time. In real world it may be possible to move any number of robots in some complex mechanism. This approach may not be able to evolve the same. The addition of some heuristics to the planning algorithm may help to further increase its scalability to such scenarios. These heuristics may further help to get the execution time of the algorithm down. All these problems may be worked over into the future. Further an exhaustive testing of the algorithm with different models, maps, and robot locations and velocities may be done. This would clearly speak out the positive and negative aspects of the algorithm.

References

T. Arai & J. Ota (1992) Motion Planning of multiple mobile robots, Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent robots and Systems, pp 1761-1768.

H. Asama, A. Matsumoto, and Y. Ishida (1989) Design of an Autonomous and Distributed Robot System: ACTRESS, IEEE/RSJ Intl. Workshop on Intelligent Robots and Systems, pp 283-290, Tsukuba, Japan

H. Asama, K. Ozaki, H. Itakura, A. Matsumoto, Y. Ishida, & I. Endo (1991) Collision avoidance among multiple mobile robots based on rules and communication, IEEE/RSJ Intl. Workshop on Intelligent Robots and Systems, pp 1215-1220, Osaka, Japan

M. Bennewitz, W. Burgard, & S. Thrun (2001) Optimizing schedules for prioritized path planning of multi-robot systems, Proceedings 2001 IEEE International Conference on Robotics and Automation, pp 271 – 276

M. Bennewitz, W. Burgard, & S. Thrun (2002) Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots, Robotics and Autonomous Systems, 41: 89–99

R. Bohlin & L. E. Kavralki (2000) Path Planning using Lazy PRM, Proc. IEEE Intl. Conf. On Robotics and Automation, pp 521-528, San Francisco, CA

S. Carpin & E. Pagello (2009) An experimental study of distributed robot coordination, Robotics and Autonomous Systems, 57: 129-133

C. M. Clark (2005) Probabilistic Road Map sampling strategies for multi-robot motion planning, Robotics and Autonomous Systems, 53: 244–264

C. M. Clark, S. M. Rock, J. C. Latombe (2003) Dynamic Networks for Motion Planning in Multi-Robot Space Systems, In: Intl. Symp. of Artificial Intelligence, Robotics and Automation in Space

Y. Dongyong, C. Jinyin, N. Matsumoto, Y. Yamane (2006) Multi-robot Path Planning Based on Cooperative Co-evolution and Adaptive CGA, Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp 547 – 550, Hong Kong

N. García-Pedrajas, C. Hervás-Martínez, J. Muñoz Pérez (2003) COVNET: a cooperative coevolutionary model for evolving artificial neural networks, IEEE Transactions on Neural Networks 14 (3): 575–596.

R. Kala, A. Shukla, & R. Tiwari (2009) Comparative analysis of intelligent hybrid systems for detection of PIMA indian diabetes, Proceedings of the IEEE 2009 World Congress on Nature & Biologically Inspired Computing, pp 947 - 952, Coimbatote, India

R. Kala, A. Shukla, & R. Tiwari (2010a) Clustering Based Hierarchical Genetic Algorithm for Complex Fitness Landscapes, International Journal of Intelligent Systems Technologies and Applications, 9(2): 185-205
R. Kala, A. Shukla, & R. Tiwari (2010b) Dynamic Environment Robot Path Planning using Hierarchical Evolutionary Algorithms, Cybernetics and Systems, 41(6): 435-454
R. Kala, A. Shukla, & R. Tiwari (2010c) Robotic Path Planning using Evolutionary Momentum based Exploration, Journal of Experimental and Theoretical Artificial Intelligence, Accepted In Press
R. Kala, A. Shukla, & R. Tiwari (2010) Fusion of probabilistic A* algorithm and fuzzy inference system for robotic path planning, Artificial Intelligence Review, 33(4): 275-306
R. Kala, A. Shukla, & R. Tiwari (2011) Evolving Robotic Path with Genetically Optimized Fuzzy Planner. International Journal of Computational Vision and Robotics, Inderscience Publishers, Accepted In Press
L. E. Kavraki, & J. C. Latombe (1997) Probabilistic roadmaps for robot path planning, In Practical Motion Planning in Robotics: Current Approaches and Future Directions, pp 33-53, John Wiley
L. E. Kavraki, P. Svestka, J. C. Latombe, & M. E. Overmars (2002) Probabalistic roadmaps for path planning in high dimensional spaces, IEEE Trans. on Robotics and Automation, 12(4): 566 - 580
F. A. Kolushev & A. A. Bogdanov (2000) Multi-agent Optimal Path Planning for Mobile Robots in Environment with Obstacles, Proc. PSI'99, Lecture Notes in Computer Science 1755, pp. 503-510, Springer-Verlag Berlin, Heidelberg.
J. R. Koza (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, Mass.
J. J. Kuffner & S. M. LaValle (2000) RRT-Connect: An Efficient Approach to Single-Query Path Planning, In Proc. 2000 IEEE Int'l Conf. on Robotics and Automation, Vol. 2, pp 995 – 1001, San Francisco, CA , USA
D. E. Moriarty (1997) Symbiotic Evolution of Neural Networks in Sequential Decision Tasks. Ph.D. thesis, Department of Computer Science, University of Texas, Austin, Texas, 1997.
D. E. Moriarty & R. Miikkulainen (1997) Forming neural networks through efficient and adaptive coevolution. Evolutionary Computation, 5(4):373–399.
M. O'Neill, & C. Ryan (2001). Grammatical evolution. IEEE Transactions on Evolutionary Computation, 5(4): 349–358
M. O'Neill & C. Ryan (2003) Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, volume 4 of Genetic programming. Kluwer Academic Publishers
C. S. Oliver, M. Saptharishi, J. M. Dolan, A. Trebi-Ollennu, & P. K. Khosla (1999) Multi-Robot Path Planning by Predicting Structure in a Dynamic Environment, In Proceedings of the First IFAC Conference on Mechatronic Systems, Vol. II, pp 593–598
L. P. Parker, F. E. Schneider, & A. C. Schultz (2005) Multi-Robot Systems: From Swarms to Intelligent Automata Vol. 3, Springer-Verlag, New York.
M. A. Potter, K. A. De Jong (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (1): 1–29.
M. A. Potter, & K. A. DeJong (1994). A cooperative coevolutionary approach to function optimization. In Davidor, Y. and Schwefel, H.-P., editors, Proceedings of the Third Conference on Parallel Problem Solving from Nature, pp 249–257, Springer-Verlag, Berlin, Germany.
M. A. Potter, K. A. De Jong (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (1): 1–29.
G. Sánchez-Ante & J.C. Latombe (2002) Using a PRM Planner to Compare Centralized and Decoupled Planning for Multi-Robot Systems, Proc. IEEE Int. Conf. on Robotics and Autom., pp 2112 – 2119
A. Shukla, R. Tiwari, and R. Kala (2010) Towards Hybrid and Adaptive Computing, Springer-Verlag Berlin, Heidelbrg
S. Slusny, R. Neruda, P. Vidnerová (2010) Comparison of behavior-based and planning techniques on the small robot maze exploration problem, Neural Networks, 23: 560-567
P. Svestka & M. H. Overmars (1998) Coordinated path planning for multiple robots, Robotics and Autonomous Systems, 23: 125-152
E. Vendrell, M. Mellado, and A. Crespo (2001) Robot planning and re-planning using decomposition, abstraction, deduction, and prediction, Engineering Applications of Artificial Intelligence, 14: 505–518

M. Wang & T. Wu (2005) Cooperative co-evolution based distributed path planning of multiple mobile robots, Journal of Zhejiang University Science, 6A(7): 697-706