# The Application of Multiobjective Evolutionary Algorithms on Robot Path Planning System

Wen Li

November 30, 2013

| | |
|---|---|
| Partners: | Wen Li |
| | Tu Yati |
| Instructor: | Professor Ying |

**Abstract**

This is a research report about *Robot Path Planning* using *Multiobjective Evolutionary Algorithm*(MOEAs) and testing its feasibility and efficiency. In this article, we assigned three objectives to Robot the Planning problem. Based on different preferences, decision makers require a set of solutions instead of one single result for their evaluation. In this case, we adopt MOEAs, which is the core of whole system. Evolutionary algorithm has the characteristics to solve *the Multiobjective Optimization Problem*(MOP[1]): Firstly, it can produce more than one solution in each execution; Second, by adding problem-specific heuristic methods, Pareto optimal solutions can be concluded. So multiobjective evolutionary algorithms have been used to solve multiobjective optimization problems in recent years, and have formed the different kinds of multiobjective evolutionary algorithms.

This article adopts *Non-dominated Sorting Genetic Algorithm II*(NSGA-II[2, 3]), *Multiobjective Evolutionary Algorithm Based on Decomposition*(MOEA/D[4]) and *Conical Area Evolutionary Algorithm*(CAEA[5]), which are the core of the Robot Path Planning System. We proposed three constraint conditions of path planning, which are Length, S-moothness and Security, and established the mathematical modeling of MOP of the Robot Path Planning System. For the specification of path planning, we adapted NSGA-II, MOEA/D and CAEA for they are suitable to solve this problem.

**Keywords:** Path Planning, Multiobjective Optimization Problem, Evolutionary Algorithm, Pareto Optimal.

# Contents

**I Introduction** **3**
- i Multiobjective Path Planning Overview . . . 3
  - A. Background . . . 3
  - B. The Multiobjective Optimization Problem . . . 3
- ii Evolutionary Algorithm Overview . . . 4
- iii Section Conclusion . . . 5

**II Basics** **6**
- i NSGA-II Overview . . . 6
  - A. Fast Nondominated Sorting Approach . . . 6
  - B. Diversity Preservation . . . 8
  - C. Crowded-Comparison Operator . . . 8
  - D. Main Loop . . . 8
- ii MOEA/D Overview . . . 9
  - A. Decomposition of Multiobjective Optimization . . . 9
  - B. The Framework of MOEA/D . . . 9
- iii CAEA Overview . . . 12
  - A. Conical Partition Strategy . . . 12
  - B. Conical Area Indicator . . . 12
  - C. Main Process of CAEA . . . 13
- iv Section Conclusion . . . 16

**III Algorithm Design** **17**
- i Experimental Design . . . 17
- ii Mathematical Model . . . 18
  - A. Code Chromosome . . . 18
  - B. Objective Function[6] . . . 18
- iii Evolution Operator . . . 20
  - A. Crossover Operator . . . 20
  - B. Mutation Operator . . . 21
  - C. Delete Operator . . . 22
- iv Section Conclusion . . . 22

**IV Results and Analysis** **23**
- i Introduction to Experiment . . . 23
- ii Data and Analysis . . . 24
  - A. Test 1: Generations . . . 24
  - B. Test 2: Population . . . 25
  - C. Test 3: Crossover Rate . . . 26
- iii Section Conclusion . . . 28

**V Conclusion** **29**

# I Introduction

In this section, we will introduce the background and the value of the research project.

## i Multiobjective Path Planning Overview

### A. Background

Path Planning is the act of finding a path to go from location A to B. The purpose of Robot Path planning is to move the robot through an open field with obstacles in it. In reality, path planning cares not only about Length, but also about some other criteria like Smoothness, Security and Energy Saving. As a consequence, the path planning problem is actually a MOP and because of this, the solution of this problem would have considerably realistic significance.

In this article, we focus on three constraint conditions:

a. to find a path for robot from the start point to the destination ;

b. to make the path be secure for the robot;

c. to find the optimal path after first two conditions achieved;

### B. The Multiobjective Optimization Problem

A multiobjective optimization problem can be stated as follows:

$$\begin{aligned} &\mathit{maximize}\ F(x) = \big(f_1(x), \ldots, f_m(x)\big)^T \\ &\mathit{subject\ to}\ x \in \Omega \end{aligned} \tag{1}$$

where $\Omega$ is the decision (variable) space, $F : \Omega \rightarrow R^m$ consists of $m$ real-valued objective functions and $R^m$ is called the *objective space*. The *attainable objective* set is defined as the set $\{F(x)|x \in \Omega\}$.

If $x \in R^n$, all the objectives are continuous and $\Omega$ is described by

$$\Omega = \{x \in R^n | h_j(x) \leq 0, j = 1, \ldots, m\}$$

where $h_j$ are continuous functions, we call (1) a *continuous MOP*.

As the objectives in (1) contradict each other, there is no point in $\Omega$ that maximizes all the objectives simultaneously. The best tradeoffs among them can be defined as Pareto optimality.

Let $u, v \in R^m$, $u$ is said to dominate $v$ if and only if $u_i \geq v_i$ for every $i \in 1, \dots, m$ and $u_j > v_j$ for at least one index $j \in 1, \dots, m$. A point $x^* \in \Omega$ is Pareto optimal to (1) if there is no point $x \in \Omega$ such that $F(x)$ dominates $F(x^*)$. $F(x^*)$ is then called a *Pareto optimal (objective) vector*. In other words, any Pareto optimal point that improves in one objective must lead deterioration in at least on other objective. The set of all the Pareto optimal points is called the *Pareto Set* (PS) and the set of all the Pareto optimal objective vectors is the *Pareto front*(PF[7]).

## ii Evolutionary Algorithm Overview

In artificial intelligence, an *evolutionary algorithm*(EA) is a generic population-based metaheuristic optimization algorithm. An EA uses mechanisms inspired by *biological evolution*, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the evaluation function(or the fitness function) determines the quality of the solutions. Evolution operations will be performed after the repeated execution of the above operators. *Genetic algorithms*(GA) belong to evolutionary algorithms.

Some important terms of EAs have been as follows:

**Chromosome** also called *a genome*, is a set of parameters which define a proposed solution to the problem that EA is trying to solve;

**Individual** is an individual of chromosome;

**Population** is the set of all individuals;

**Fitness** is used as the indicator of the objective of an individual. The individual that has the better fitness has a higher rate to be selected for reproduction.

**Selection** is the stage of a EA in which individual are chosen from a population for later breeding (recombination or crossover).

**Crossover** is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next.

**Mutation** is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next.

**Fitness function** , also called *Evaluation function*, is a particular type of objective function that is used to summarise how close a given design solution is to achieving the set aims.

The pseudocode of EAs is as follow:

```
EVOLUTIONARY ALGORITHM
1  PARAMETERINITIALIZATION()
2  ▷ Parameters like cross rate, mutation rate and generation number.
3  POPULATIONINITIALIZATION(population)
4  for $i \leftarrow 0$ to gen_num
5  ▷ or use other stopping criterion
6        do
7            while $j <$ population_size
8                  do $FitnessFunction(j)$
9                     $j \leftarrow j + 1$
10           end
11           parent_idx ← SELECTOPERATOR()
12           child_idx ← CROSSOVEROPERATOR(parent_idx)
13           offspring ∪MUTATIONOPERATOR(child_idx)
14           population ∪ offspring
15           $i \leftarrow i + 1$
16 end
```

Brief Summarize:

1) In *Select Operator*, only the individuals with relatively high fitness would be selected and reproduce offspring.

2) *Crossover Operator* is priority to *Mutation Operator*.

3) *Cross rate* and *Mutation rate* are the values in $(0.0, 1.0)$.

4) In this article, we assign fitness equal to the objective.

## iii Section Conclusion

In this section, we introduce the background of Robot Path Planning System, and the overview of MOPs and EAs. Some of these are based on previous research and on the next, we will concentrate on our new work about this problem.

# II Basics

In this section, we will introduce the basics about NSGA-II, MOEA/D and CAEA, and specify the mathematical model of the Robot Path Planning Problem.

## i NSGA-II Overview

The NSGA-II is one of the most popular multi-objective EAs based on dominance, which is an improved version of NSGA[]. It sorts the individuals in the population by their dominance depths to keep the closeness to the PF. In addition to this, NSGA-II prefers the individuals with larger crowding distances to preserve the diversity of distribution along the PF. NSGA-II has two important features as follows:

1) using *a fast nondominated sorting procedure* and *an elitist-preserving approach*;

2) preserving diversity among solutions of the nondominated solutions;

Because of these two improvements, NSGA-II reduces the time-consuming from $O(MN^3)$ of NSGA to $O(MN^2)$.

### A. Fast Nondominated Sorting Approach

Fast Nondominated sort, called FNDS for short, is the highlight part of w-hole algorithm.

For each solution we calculate two entities: 1) domination count$n_p$, the number of solutions which dominate the solution $p$, and 2)$S_p$, a set of solutions that the solution $p$ dominates. This requires $O(MN^2)$comparisons.

Every solution in the first nondominated front will initialize their domination counter as zero. Now, for each solution $p$ with $n_p = 0$, we visit each member ($q$) of its set $S_p$ and reduce its domination count by one. In doing so, if for any member $q$ the domination count becomes zero, we put it in a separate list $Q$. These members belong to the second nondominated front. Now, the above procedure is continued with each member of $Q$ and the third front is identified. This process continues until all fronts are identified.

Following pseudocode is the procedure of the FNDS:

```
FastNondominatedSort(population)
 1  for $p \in population$
 2      do
 3          $S_p \leftarrow \emptyset$
 4          $n_p \leftarrow 0$
 5          for $q \in population$
 6              do
 7                  if $(p \prec q)$
 8                    then
 9                          $S_p \leftarrow S_p \cup \{q\}$
10                    else  if $(q \prec p)$
11                            then
12                  end
13          end
14          if $(n_p = 0)$
15            then
16                  $p_{rank} \leftarrow 1$
17                  $F_1 \leftarrow F_1 \cup \{p\}$
18          end
19  end
20  $i \leftarrow 1$
21  while $F_i \neq \emptyset$
22      do
23          $Q \leftarrow \emptyset$
24          for $p \in F_i$
25              do
26                  for $q \in S_p$
27                      do
28                          $n_q \leftarrow n_q - 1$
29                          if $(n_q = 0)$
30                            then
31                                  $q_{rank} = i + 1$
32                  end
33          end
34          $i \leftarrow i + 1$
35          $F_i = Q$
36  end
```

In the line 7, we can see that if $p$ dominates $q$, then $q$ will be added into the set of solutions dominated by $p(S_p)$. In the line 14, $p$ belongs to the first front if its counter is zero, and $p$ will have $p_{rank} = 1$.

## B. Diversity Preservation

NSGA-II uses the *Crowding Distance Assignment*(CDA) to preserve diversity. $I[i].m$ stands for the objective $m$ of the idividual $i$. The algorithm as shown at the below outlines the crowding-distance computation procedure of all solutions in an nondominated set $I$.

```
CrowdingDistanceAssignment
1   $l = |I|$ $\triangleright$ number of solutions in $I$
2   for each i
3       do
4           $I[i]_{dist} \leftarrow 0$
5   end
6   for each objective m
7       do
8           $I = $ Sort(I, m)
9           $I[1]_{dist} \leftarrow I[l] \leftarrow \infty$
10          for $i \leftarrow 2$ to $(l-1)$
11              do
12                  $diff \leftarrow f_m^{max} - f_m^{min}$
13                  $I[i]_{dist} \leftarrow I[i]_{dist} + (I[i+1].m - I[i-1].m)/diff$
14          end
```

## C. Crowded-Comparison Operator

The crowded-comparison operator ($\prec_n$) guides the selection process at the various stages of the algorithm. Assume that every individual $i$ in the population has two attributes:

1) nondomination rank($i_{rank}$);

2) crowding distance($i_{dist}$);

A partial order $\prec_n$ can be defined as:

$$i \prec_n j \quad \text{if}(i_{rank} < j_{rank})$$

$$or \ ((i_{rank} = j_{rank}) \ and \ (i_{dist} > i_{dist}))$$

## D. Main Loop

Now, this is the main process of the NSGA-II: First, initialize the $P_0$ with a random function or a problem-specific function, and use $FNDS$ on every individual of the population. After that, do the evolution operators and then generate the offspring $Q_0$.

*NSGA − II*

```
1   $R_t \leftarrow P_t \cup Q_t$ $\triangleright$ combine parent and offspring population.
2   $F = $ FastNondominatedSort($R_t$)
3   $P_{t+1} = \emptyset$and$i = 1$
4   repeat  $\triangleright$ repeat until the parent population is filled.
5           CrodingDistanceAssignment($F_i$)
6           $P_{t+1} = P_{t+1} \cup F_i$
7           $\triangleright$ include $i$th nondominated front in the parent pop
8           $i = i + 1$
9     until $|P_{t+1}| + |F_i| \leq N$
10  Sort($F_i, \prec_n$) $\triangleright$ sort in descending order using $\prec_n$
11  $P_{t+1} = P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$
12  $Q_{t+1} = $ MakeNewPop($P_{t+1}$)
13  $\triangleright$ use selection, crossover and mutation to create a new population
14  $t = t + 1$
```

## ii MOEA/D Overview

Compared to NSGA-II, MOEA/D is a more efficient EA, which uses the decomposition strategy. *Qingfu ZHANG* proposed this EA in 2007.

### A. Decomposition of Multiobjective Optimization

There are several approaches for converting the problem of approximation of the PF into a number of scalar optimization problems. In this article, we concentrate on one of them, which is Tchebycheff Approach[7].

In Tchebycheff Approach, the scalar optimization problem is in the form

$$minimize g^{te}(x|\lambda, z^*) = \max_{1\leq i\leq m} \{\lambda_i |f_i(x) - z^*\}$$

$$subject\ to\ x \in \Omega \tag{2}$$

where $z^* = (z_1^*, \ldots, z_m^*)^T$ is the reference point, i.e., if the goal is minimization, $z^* = \min\{f_i(x)|x \in \Omega\}$ for each $i = 1, \ldots, m$. For each Pareto optimal point $x^*$ there exists a weight vector $\lambda$ such that $x^*$ is the optimal solution of (2) and each optimal solution of (2) is a Pareto optimal solution of (1). Thus, it is able to obtain different Pareto optimal solutions by altering the weight vector.

### B. The Framework of MOEA/D

In the following description, the Tchebycheff approach is used as the decomposition method.

Let $\lambda^1, \ldots, \lambda^N$ be a set of even spread weight vectors and $z^*$ be the reference point. By using the Tchebycheff approach the problem can be decomposed into $N$ scalar optimization subproblems, and the objective of the $j$th subproblem is

$$g^{te}(x|\lambda^j, z^*) = \max_{1 \leq i \leq m} \{\lambda_i^j |f_i(x) - z_i^*|\} \qquad (3)$$

where $\lambda^j = (\lambda_i^j, \ldots, \lambda_m^j)^T$. MOEA/D minimizes all these N objective functions simultaneously in a single run.

In MOEA/D, a neighborhood of weight vector $\lambda^i$ is defined as a set of its several closest weight vectors in $\{\lambda_1, \ldots, \lambda_N\}$. The neighborhood of the $i$th subproblem consists of all the subproblems with the weight vectors from the neighborhood of $\lambda^i$. The population is composed of the best solution found so far for each subproblem.

At each generation $t$, MOEA/D with the Tchebycheff approach maintains:

- a population of $N$ points $x^1, \ldots, x^N \in \Omega$, where $x^i$ is the current solution to the $i$th subproblem;
- $FV^1, \ldots, FV^N$, where $FV^i$ is the F-value of $x^i$, i.e., $FV^i = F(x^i)$ for each $i = 1, \ldots, N$;
- $z = (z_1, \ldots, z_m)^T$, where $z_i$ is the best value found so far for objective $f_i$;
- an external population(EP), which is used to store nondominated solutions found during the search.

The algorithm works as follows:
**Input**:

- MOP(1);
- a stopping criterion;
- N: the number of the subproblems considered in MOEA/D;
- a uniform spread of N weight vectors: $\lambda^1, \ldots, \lambda^N$;
- T: the number of the weight vectors in the neighborhood of each weight vector.

**Output**: EP. This is the pseudocode:

*MOEA/D*

```
▷ **Initialization**
EP ← $\emptyset$
**ComputeDistance**:
      **do** Compute the Euclidean distances
          between any two weight vectors
**end**

**InitializeNeighborhood**:
      **do** For each $i = 1, \ldots, N$, $setB(i) = i_1, \ldots, i_T$,
          where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the $T$ closest weight vectors to $\lambda^i$.
**end**

**InitializePopulation**:
      **do** by random or a problem-specific method.
**end**
**InitializeReference**:
      **do** by a problem-specific method.
**end**

▷ **Update**:
**repeat**
        **for** $i \leftarrow 1$ *to* $N$
            **do**
                **Reproduction**()
                **Improvement**()
                **UpdateReference**:
                    **do** For each $j = 1, \ldots, m$, if $z_j < f_j(y')$,
                        Then $z_j \leftarrow f_j(y')$
                **end**

                **UpdateNeighborhood**:
                    **do** For each index $j \in B(i)$,
                        if $g^{te}(y'|\lambda^j, z) \leq g^{te}(y^j|\lambda^j, z)$,
                        then set $x^i = y'$ and $F - value^j = F(y')$
                **end**
                **UpdateEP**:
                    **do** Remove from EP all the vectors
                        which are dominated by $F(y')$.
                        If no vectors in EP dominate $F(y')$,
                        add $F(y')$ to EP.
                **end**
        **end**
   **until** ($stop\_criteria = true$)
```

We can see from the algorithm, the majority of time-consuming is updating $T$ neighborhood(line 31). Therefore, the time-consuming of MOEA/D is $O(NT)$. Because $T < N$, MOEA/D is more time-efficient than NSGA-II.

## iii CAEA Overview

A *Conical Area Evolutionary Algorithm* is proposed by *Weiqin YING* to further improve computational efficiencies of evolutionary algorithms for bi-objective optimization. CAEA partitions the objective space into a number of conical subregions and then the MOP becomes a set of scalar optimization subproblem.

### A. Conical Partition Strategy

It requires an *utopia point* and a *nadir point* in the objective space to partition the decision space. Let $A$ be a set of solutions, $A \in \Omega$. The utopia point and the nadir point of A are respectively: $F^{\triangle}(A) = (f_1^{\triangle}(A), f_2^{\triangle}(A))$ and $F^{\nabla}(A) = (f_1^{\nabla}(A), f_2^{\nabla}(A))$ where $f_i^{\triangle}(A) = \min\limits_{x \in A} f_i(x)$ and $f_i^{\nabla}(A) = \max\limits_{x \in A} f_i(x), i = 1, 2$
In addition, the transformed objective vectors are required: $\bar{y} = y - F^{\triangle}(A)$. After this transformation, the utopia point is at the origin$(0, 0)$.

**Definition 1** (Observation vector). *The observation vector for any transformed point $\bar{y} = (\bar{y_1}, \bar{y_2})$ is $\bar{V}(\bar{y}) = (\bar{v_1}(\bar{(y)}), \bar{v_2}(\bar{y}))$ where $\bar{v_i}(\bar{y}) = \frac{\bar{y_i}}{\bar{y_1} + \bar{y_2}}, i = 1, 2$.*

The observation vector has these features: $\bar{v_i}(\bar{y}) \leq 0$, and $\bar{v_1}(\bar{y}) + \bar{v_2}(\bar{y}) = 1$. Given a prescribed number of divisions N, a series of uniformly distributed observation vectors $\bar{V^{(k)}}$ can be defined as $\bar{V^{(k)}} = (\frac{k}{N-1}, 1 - \frac{k}{N-1}), k = 0 \ldots N - 1$.

**Definition 2** (Conical subregion). *For the central observation vector $\bar{V^{(k)}} = (\frac{k}{N-1}, 1 - \frac{k}{N-1}), k = 0 \ldots N - 1$, the region $\bar{C} = \bar{y} = (\bar{y_1}, \bar{y_2}) | \bar{y_1} \geq 0 \wedge \bar{y_2} \geq 0$ can be devided into N conical subregions: $\bar{C^{(k)}} = \bar{y} \in \bar{C} | \frac{k-0.5}{N-1} \leq \bar{v_1}(\bar{y}) < \frac{k+0.5}{N-1}$, $k = 0 \ldots N - 1$ where k is called the index of the sub region $\bar{C^{(k)}}$.*

It can be inferred by Def.2 that the index of the subregion in which a solution $x \in A$ lies is $k = \llcorner \frac{(N-1)(f_1(x) - f_1^{\triangle}(A))}{f_1(x) - f_1^{\triangle}(A) + f_2(x) - f_2^{\triangle}(A)} + \frac{1}{2} \lrcorner$ where $\llcorner \cdot \lrcorner$ denotes the bottom integral function.

### B. Conical Area Indicator

CAEA borrows the feature of *hypervolume* to present an efficient conical area indicator for bi-objective area.

**Definition 3** (Conical area). *Let $\bar{y'} \in \bar{C}^{(k)}$, $0 \leq k \leq N-1$, $\bar{y'}$ denote the reference point which should be an approximate infinity point dominated by all feasible solutions, then the conical area for $\bar{y'}$ is the area of the portion $\tilde{C}(\bar{y'}) = \{\bar{y} \in \bar{C}^{(k)} | \neg(\bar{y'} \prec \bar{y}) \wedge \bar{y} \wedge \bar{y'}\}$ , written as $S(\bar{y'})$.*

The conical area $S(\bar{y'})$ is the sum of the area of two triangles conical area $\triangle OL^{(1)}U^{(1)}$ and $\triangle \bar{y'} U^{(2)} U^{(1)}$ where $U^{(1)} = (a\bar{y'_2}, \bar{y'_2})$ is the intersection between $\bar{y_2} = \bar{y'_2}$ and the upper boundary of $\bar{C}^{(k)}$:

$$
\begin{aligned}
S(\bar{y'}) \; &= \; 0.5(b-a)(\bar{y'_2})^2 + 0.5\frac{1}{a}(\bar{y'_1} - a\bar{y'_2})^2 \\
&= 0.5\frac{1}{a}(\bar{y'^2_1})^2 + 0.5b(\bar{y'_1})^2 - \bar{y'_1}\bar{y'_2}). \qquad (4)
\end{aligned}
$$

However, it still requires some work to prove the connection between the conical area indicator and local Pareto optimality. Because of the length of this report, the process of the proof will not be involved. In conclusion, the indicator possesses the feature of local Pareto optimality, so that the solution with the minimal conical area must be nondominated in the decision subset.

## C. Main Process of CAEA

In CAEA, the $k$th scalar subproblem $g^{caea,k}, 0 \leq k \leq N-1$, uses the conical area as its scalar objective in the form:

$$
\begin{aligned}
&\textit{minimize } g^{caea,k}(x) = S(F(x) - F^{\triangle}(\Omega)) \\
&\textit{subject to } x \in \Omega^{(k)}. \qquad (5)
\end{aligned}
$$

The $k$th individual in the population of CAEA is just the best solution $x^k$ found so far for the subproblem $g^{caea,k}$. Each generation of CAEA only produce one child $x'$. CAEA uses an area-based tournament selection *Area-Tour()* for the solutions with smaller conical areas should be preferred in the neighboring subregions. The pseudocode is as shown below:

*AreaTour()*

```
1) Randomly select two indexes $a$, $b$ from $0 \dots N-1$.
2) Set $a_1 \leftarrow a-1$ and $a_2 \leftarrow a+1$. If $a = 0$ then $a_1 = 2$ else if $a = N-1$
   then $a_2 = N-3$. Then $b_1$ and $b_2$ are set similarly in terms of $b$.
3) If $(\frac{S(F(x^{a_1}))+S(F(x^{a_2}-F^{\triangle}))}{2} - S(F(x^a) - F^{\triangle}) >$
   $\frac{S(F(x^{b_1}))+S(F(x^{b_2}-F^{\triangle}))}{2} - S(F(x^b) - F^{\triangle}))$ then $k \leftarrow a$ else $k \leftarrow b$
4) **Output**: $k$th index of the selected solution in $P$.
```

CAEA only needs to update individual associated with the subproblem in terms of conical area in which the only offspring $x'$ lies. The whole algorithm works as follows:

**Step 1 Initialization**:

```
 1      do
 2          Generate $N$ initial solutions $A \leftarrow \{z^0, \ldots, z^{N-1}\}$
 3          by randomly sampling from $\Omega$.
 4
 5          Initialize $F^{\triangle} \leftarrow (f_1^{\triangle}, f_2^{\triangle})$ and $F^{\nabla} \leftarrow (f_1^{\nabla}, f_2^{\nabla})$
 6          by setting $f_i^{\triangle} \leftarrow \min_{0 \le j \le N-1} f_i(z^j)$, $f_i^{\nabla} \leftarrow \max\limits_{0 \le j \le N-1} f_i(z^j)$.
 7
 8          InitializePopulation:
 9          Create a population $P \leftarrow \{x^0, \ldots, x^{N-1}\}$
10          where $x^0 = \ldots = x^{N-1} = null$.
11          Set $A' \leftarrow A$.
12
13          For each$k = 0 \ldots N-1$: if $A^{(k)} \neq \emptyset$,
14          then $x^k \leftarrow \arg \min\limits_{z \in A^{(k)}} S(F(z) - F^{\triangle})$ and $A' \leftarrow A' - x^k$.
15
16          For each $k = 0 \ldots N-1$: if $x^k = null$,
17          then $x^k \leftarrow \arg \min\limits_{z \in A'} d(\bar{V}(F(z) - F^{\triangle}), \bar{V^{(k)}})$
18          and $A' \leftarrow A' - x^k$.
19
20          Set $nE \leftarrow N$ and $L \leftarrow \ulcorner \frac{N}{10} \urcorner$.
21  end
```

**Step 2 Evolution**:

```
1          do
2              Selection:
3              Set $iL \leftarrow nE \mod L$.
4              If $iL < 2$, then $p_1 \leftarrow iL \times (N-1)$ else $p_1 \leftarrow AreaTour()$.
5              Set $p_2 \leftarrow Areatour()$
6
7              Reproduction:
8              Create an offspring $x'$ from $x^{p_1}$ and $x^{p_2}$
9              using genetic operators.
10             Set $nE \leftarrow nE + 1$
11
12             Update $F^{\triangle}$ and $F^{\nabla}$
13             For each $i = 1, 2$, if $f_i(x') < f_i^{\triangle}$,
14             then set $f_i^{\triangle} \leftarrow f_i(x')$ and
15             if $f_i(x') > f_i^{\nabla}$,
16             then set $f_i^{\nabla} \leftarrow f_i(x')$.
17
18             Update $x^k$:
19             Set $k_1 \leftarrow \llcorner \frac{(N-1)(f_1(x')-f_1^{\triangle})}{f_1(x')-f_1^{\triangle}+f_2(x')-f_2^{\triangle}} + \frac{1}{2} \lrcorner$
20             and $k_2 \leftarrow \llcorner \frac{(N-1)(f_1(x^{k_1})-f_1^{\triangle})}{f_1(x^{k_1})-f_1^{\triangle}+f_2(x^{k_1})-f_2^{\triangle}} + \frac{1}{2} \lrcorner$
21
22             If $k_2 \neq k_1 \vee S(F(x') - F^{\triangle}) < S(F(x^{k_1} - F^{\triangle}))$
23             then set $x^{k_1} \leftarrow x'$
24     end
```

**Step 3 Stoping Criterion**:

```
1   if the stopping criterion is not satisfied
2       then go back to Step 2
3   end
```

**Step 4 Post-processing**:

```
1         do
2             Remove the dominated solutions from
3             the final population $P$ to obtain the approximate $PS$ $P'$.
4
5             Output $A'$ and the approximate $PF$ $F(P')$.
6   end
```

where $\ulcorner \cdot \urcorner$ denotes the top integral function, $d(\cdot, \cdot)$ represents the Euclidean distance between two vectors in **Step 1 line 17** and mod in **Step 2 line 3** denotes the modulus operator.

The major computational costs of CAEA are the update operations in **Step 2**. Since each child needs to update the individual of only one subproblems in **Step 2**, the complexity of CAEA is $O(1)$ for an offspring and $O(N)$ for $N$ offsprings. Hence CAEA has an extraordinary efficiency compared to NSGA-II($O(N^2)$) and MOEA/D($O(TN)$).

## iv Section Conclusion

In this section, we made some introductions about three EAs which are used in this article. In the next section, the details of the algorithms will be presented.

## III Algorithm Design

This section details the design of our experiment and algorithms. To transfer the *Robot Path Planning* problem to a MOP, a specific math model is used and it will be presented on the next.

### i Experimental Design

As shown in the following figure, we can see that the start point and the destination point are fixed on the two corners on of the map. The size of this map is $20 \times 20$. In the experiment, all the individuals are initialized randomly and optimized with EAs. This experiment is based on a random

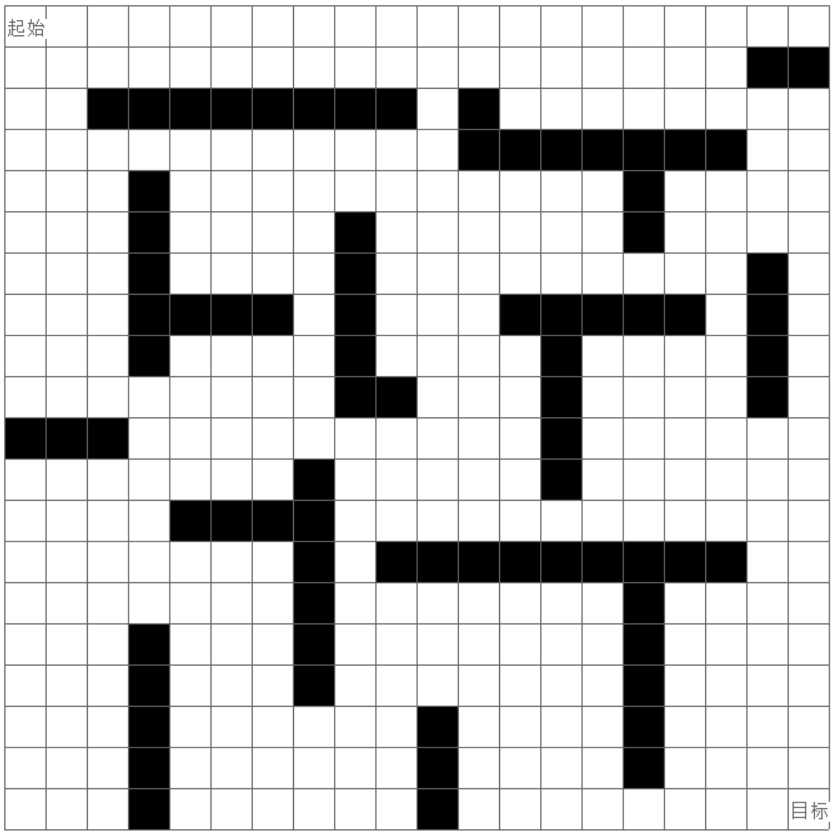


Figure 1: a sample path map

static environment, which means the blocks on the map are certain when the test begins and all the initializations are random. This environment has some features:

1) Before the test starts, the position and the number of blocks can be modified. Other parameters like *crossover rate*, *mutation rate* and number of *population* and *generation* also are changeable.

2) During the test is running, all these parameters above can not be changed or modified.

## ii Mathematical Model

### A. Code Chromosome

To code *chromosome*, this article employed *vector* in *C++ STL*. Each *vector* stores one *path* of an individual. Its structure:

$$path \; = \; \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

where $n$ is changeable. $(x_0, y_0)$ and $(x_{n-1}, y_{n-1})$ stand for the start point and the destination point. $(x_i, y_i)$, $i = 1, 2, \ldots, n-2$ stand for the coordinates of the points on the map.

An individual has the structure as follows:

$$indiv \; = \; \{path, obj\}$$

where *indiv* is the initial of the individual and *obj* stands for the vector of the objective values which acquire from *objective function.* So we can conclude that the population is as shown below:

$$population \; = \; \{indiv_1, indiv_2, \ldots, indiv_k\}$$

$$k = populationsize$$

### B. Objective Function[6]

First of all, task of path planning is to find a way from the start point to the destination point without blocked. Besides, this article assigned another three objectives: *Length*, *Smoothness* and *Security.*

The objective functions evaluate every path in every individual. These are the main process:

**a. Length**

$$F_1(P) = 1 - shortest\_len / \sum_{i=1}^{n-1} |p_i p_{i+1}|$$

where *shortest_len* is the straight-line distance from the start point to the destination point. $|p_i p_{i+1}|$ is the distance of $p_i p_{i+1}$. This $F_1(P)$ is the objective function of *Length* and the smaller $F_1(P)$ means the shorter *Length.*

**b. Smoothness**

The *Smoothness* uses the angle the path travelling as an indicator.

$$\textbf{if } n > 2, \quad F_2(P) = \big((\sum_{i=2}^{n-1} \theta_i + k\frac{\pi}{2})/(n-2)\big)/\pi$$

$$\theta \;=\; \arccos\big(\frac{p_{i-1}p_i \cdot p_ip_{i+1}}{|p_{i-1}p_i||p_ip_{i+1}|}\big)$$

$$\textbf{if } n = 2, \quad F_2(P) = 1.$$

where, $\theta_i$ $(i = 2, \ldots, n-1)$ is the angle between $p_{i-1}p_i$ and $p_ip_{i+1}$. $k$ is the number of $\theta_i > \pi/2$, which means punishment. $F_2(P)$ is proportional to the average angle which the path $P$ travelled. As we can see from the above, the smaller $F_2(P)$ means the better smoothness.

**c. Security**

Since our robot is not a particle, *Security* becomes a very important indicator. *Security* is related to the distance between our moving object and blocks. The moving object should keep a secure distance to the block in case of the crashing.

$$F_3(P) \;=\; 1 - \big(average\_dist(p_n) - M \cdot imp\big) \;/\; (shortest\_len/2)$$

$$average\_dist(p_n) \;=\; \sum_{x=1}^{n-1}\big(\sum_{y=1}^{m} dis(x,y)\big) \;/\; (m \cdot n)$$

where $average\_dist(p_n)$ is the function to calculate the average distance of every point in the $p_n$ to all blocks. $shortest\_len/2$ means the longest distance from the point to a block. $n$ is the number of points in the $n$th path $p_n$, while $m$ is the number of the blocks in the field. $M$ is a counter of the point of intersection of the path and the block. $M \cdot imp$ means the punishment for the intersection and in this article, $imp$ equals the width of the map.

$F_3(P)$ is the objective function to evaluate the *Security*. The same as *Length* and *Smoothness*, the lower the *Security* is, the better.

All these three objective functions have the domain of [0, 1]. Usually, the ideal value is 0 which can not obtain. In addition, it still requires an objective function to check whether the path passed the blocks or not. We assigned a counter $D$ to record the number of blocks on the path, which is used as an indicator for punishment. If $D$ of an individual is very big, it will be less possible to be selected to reproduce offspring. This process is very simple, so that it will not be presented in this article.

## iii Evolution Operator

Evolution operators play the important roles in reproduction. In this article, we employed three evolution operators: *Selection*, *Crossover* and *Mutation*. Operation of *Selection* is related to the implementation of each EA. As a consequence, this section will not introduce this again. In the following, we will concentrate on *Crossover* and *Mutation* operators.

### A. Crossover Operator

If two paths have at least one same point, they can be crossed through the crossover operator. This operation requires two paths $path_1$ and $path_2$ from the parents, and then generates two children with paths related to both parents. There are two kinds of crossover, one is *uniform crossover* and the other one is *single-point crossover*. Based on the situation of this path planning problem, we decided to use single-point crossover. As the figure shown below:

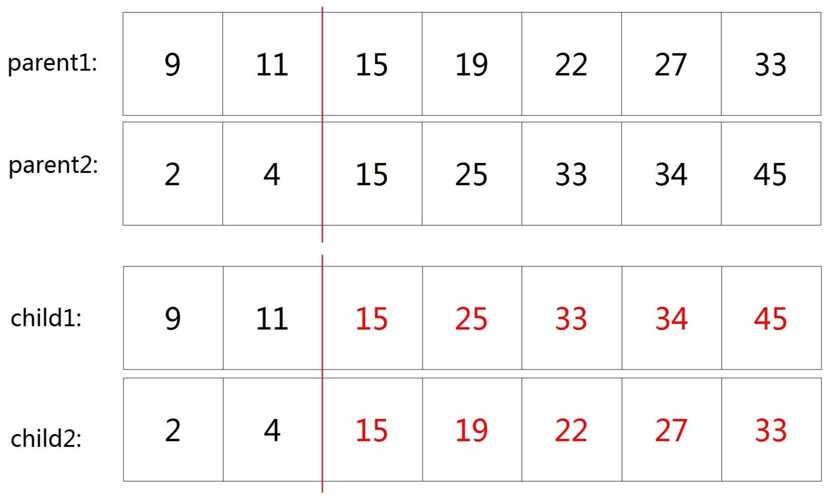

parent1: 9 11 15 19 22 27 33
parent2: 2 4 15 25 33 34 45
child1: 9 11 15 25 33 34 45
child2: 2 4 15 19 22 27 33

Figure 2: single-point crossover

Here is the pseudocode of crossover function:

*Crossover*()

```
 1  Input: $parent1$, $parent2$
 2  Initialize $child1$ and $child2$
 3  $r \leftarrow$ RANDOM() in $[0.0, 1.0]$
 4  if ($r < cross\_rate$)
 5     then random initialize $cross\_point$ in $parent1$ and $parent2$
 6           for $i \leftarrow 0$ to $cross\_point$
 7                do $child1_i \leftarrow parent1_i$
 8                   $child2_i \leftarrow parent2_i$
 9           end
10           for $j \leftarrow cross\_point$ to $n-1$
11                do $child1_j \leftarrow parent1_j$
12           end
13           for $k \leftarrow cross\_point$ to $n-1$
14                do $child2_k \leftarrow parent2_k$
15           end
16     else  $child1 \leftarrow parent1$
17           $child2 \leftarrow parent2$
18  end
19  Output: $child1$, $child2$
```

## B. Mutation Operator

Mutation operator plays the role of keeping the population diversity. The operation of mutation is not always heading the good way. on one hand, mutation operator can optimize the path in many situations, on the other hand, if this path is already feasible, mutation might damage this feasibility. This is the pseudocode:

Mutation()

```
 1  Input: $population$
 2  $indiv \in population$
 3  for $i \leftarrow 0$ to $population\_size$
 4       do $r \leftarrow$ RANDOM() in $[0.0, 1.0]$
 5          if ($r < mutation\_rate$ & $indiv_i\_size > 2$)
 6             then $point \leftarrow$ RANDOM()
 7                  if (ISSERIES($point$) & ISBLOCK($point$) $= false$)
 8                     then $point$ replace the point in $indiv_i$
 9                  end
10          end
11  end
12  Output: $population$
```

where at line 7, the mutation will be executed, only if *point* is series to the next and the previous point of it and *point* is not block point.

### C. Delete Operator

*Delete Operator* is a problem-specific operator. The function of this *Delete* is to reduce the length of the path by deleting repetitions and the useless. This operator can speed up the whole algorithm. The following is the process of delete operator:

```
$Delete()$
Input: $population$
$indiv \in population$
for $i \leftarrow 0$ to $population\_size - 1$
    do for $j \leftarrow 0$ to $indiv_i\_size$
        do for $k \leftarrow indiv_i\_size - 1$ to $j$
            do if $(k = j)$
                then $point \leftarrow k$
                    $flag \leftarrow true$
            end
        end
        if $flag = true$
            then $delete\ indiv_{j \rightarrow k}$
        end
    end
end
Output: $population$
```

## iv Section Conclusion

In this section, we introduced the math model of *Robot Path Planning* problem including, chromosome coding, parameter setting and objective functions. In addition, we presented three evolution operators employed in this article.

## IV Results and Analysis

### i Introduction to Experiment

The experiment of *Robot Path Planning* is based on the program we developed, so that we need to describe the features of this program.

1) Three indicators, *Length*, *Smoothness* and *Security*, have the same codomain [0.000, 1.000]. The lower the indicator becomes, the shorter, the smoother or the securer the path is.

2) The greater crossover rate leads to the faster convergence of Pareto solution set while the mutation rate is the opposite. The value of the mutation rate is dependent on other parameters. As an instance, the bigger the population is, the lower the mutation rate should be. In general, the crossover rate is in [0.4, 0.9] and the mutation rate should be in [0.001, 0.1].

3) This experiment employs a random static environment. The size of maps does not affect the value of parameters, like crossover rate, mutation rate, population size and total generation. Because of that, we decided to use a $20 \times 20$ map as shown below:

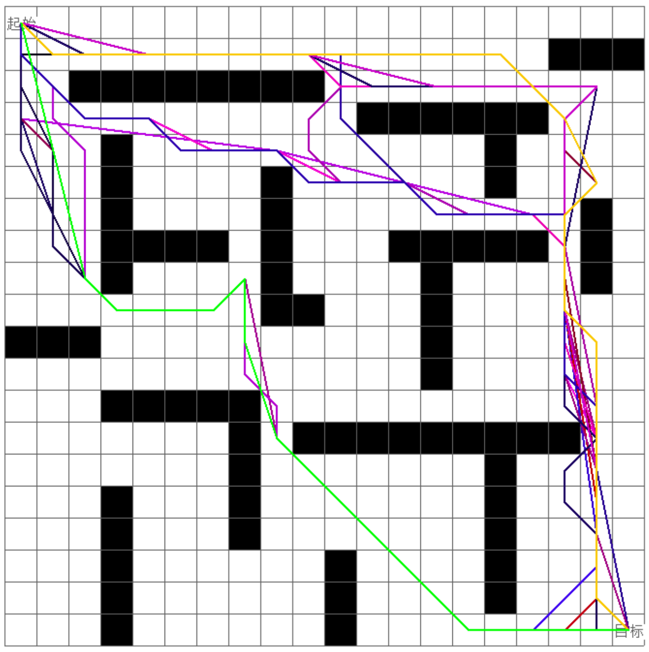


Figure 3: a running sample

## ii Data and Analysis

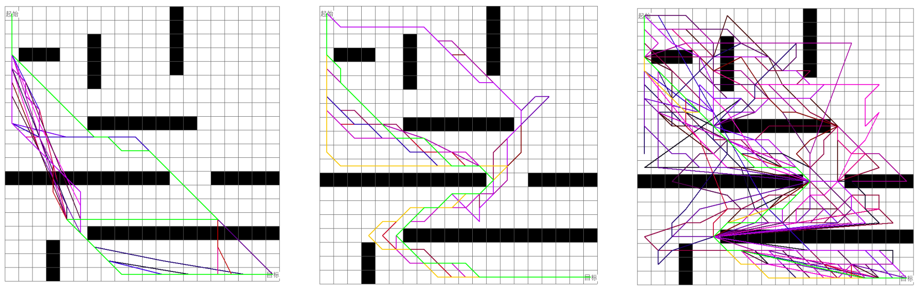
### A. Test 1: Generations

In this test, the parameters are as follows: *crossover rate* = 0.9; *mutation rate* = 0.1; *population size* = 100. The variable is *generation* from 100*th* to 500*th*. These are the data of test 1:

| Generation: | | 100th | 200th | 300th | 400th | 500th |
|---|---|---|---|---|---|---|
| NSGA-II | Length | 0.365 | 0.356 | 0.328 | 0.360 | 0.286 |
| | Smoothness | 0.708 | 0.100 | 0.143 | 0.143 | 0.143 |
| | Security | 0.384 | 0.402 | 0.361 | 0.354 | 0.399 |
| MOEA/D | Length | 0.374 | 0.378 | 0.373 | 0.344 | 0.370 |
| | Smoothness | 0.186 | 0.114 | 0.146 | 0.151 | 0.140 |
| | Security | 0.369 | 0.374 | 0.359 | 0.346 | 0.351 |
| CAEA | Length | 0.361 | 0.351 | 0.349 | 0.351 | 0.351 |
| | Smoothness | 0.219 | 0.172 | 0.175 | 0.161 | 0.151 |
| | Security | 0.382 | 0.341 | 0.343 | 0.344 | 0.344 |

Table 1: Data from 100th to 500th generation.

We can see from the table that the convergence of NSGA-II began at about 200*th* to 300*th* generation. About MOEA/D, which is very similar to NSGA-II, also began to converge since around 300*th*, while CAEA had not converged too early for we can see the optimizing at 300*th*, 400*th* and 500*th*. These are the running screenshots of three algorithms(green line is for the shortest path, yellow one is for the smoothest and the purple line is for the securest line):



(a) NSGA-II (b) MOEA/D (c) CAEA

Figure 4: Running screenshots at the 100*th*

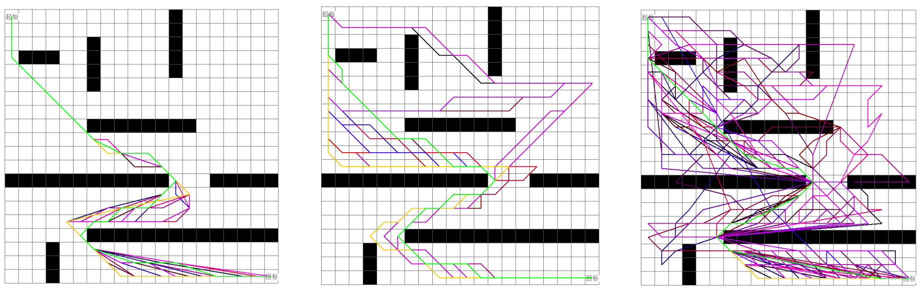

(a) NSGA-II (b) MOEA/D (c) CAEA

Figure 5: Running screenshots at the 200$th$

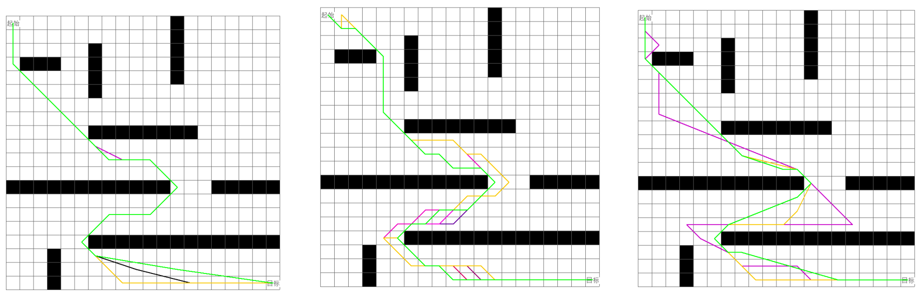

(a) NSGA-II (b) MOEA/D (c) CAEA

Figure 6: Running screenshots at the 500$th$

At 500$th$, regardless of some paths caused by mutation, most the paths are converged. From 100$th$ to 200$th$, we can clearly see that the diversity of three of CAEA is better than the other two, and MOEA/D is better than NSGA-II.

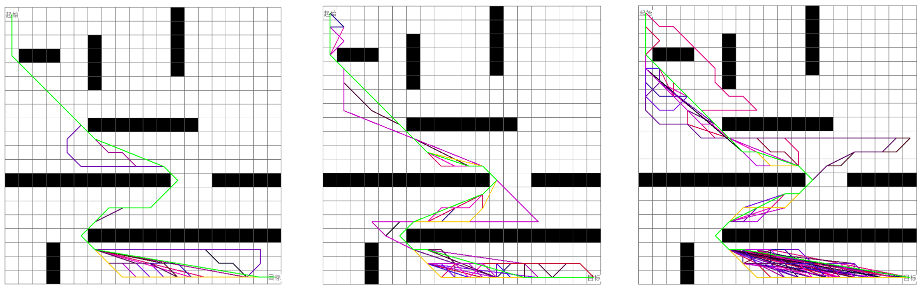
## B. Test 2: Population

Test 2 is testing the interaction of population size and results. Under the condition of crossover rate = 0.9, mutation rate = 0.1 and generation = 500, we collected three sets of results for each algorithm. The variable is the size of population: 40, 100, 200.

This is the data table:

| Populations size: | | 40 | 100 | 200 |
|---|---|---|---|---|
| NSGA-II | Length | 0.286 | 0.286 | 0.328 |
| | Smoothness | 0.105 | 0.143 | 0.139 |
| | Security | 0.430 | 0.399 | 0.361 |
| MOEA/D | Length | 0.378 | 0.332 | 0.298 |
| | Smoothness | 0.189 | 0.192 | 0.200 |
| | Security | 0.382 | 0.360 | 0.333 |
| CAEA | Length | 0.353 | 0.351 | 0.350 |
| | Smoothness | 0.156 | 0.151 | 0.171 |
| | Security | 0.388 | 0.344 | 0.370 |

Table 2: Data of different population size.



(a) NSGA-II (b) MOEA/D (c) CAEA

Figure 7: CAEA under different population size

Because of the large generation and the high crossover rate, the influence of population size is not very considerable. On the other hand, the larger the population is, the more diversity it has. The population of 40 is the lowest feasible size.

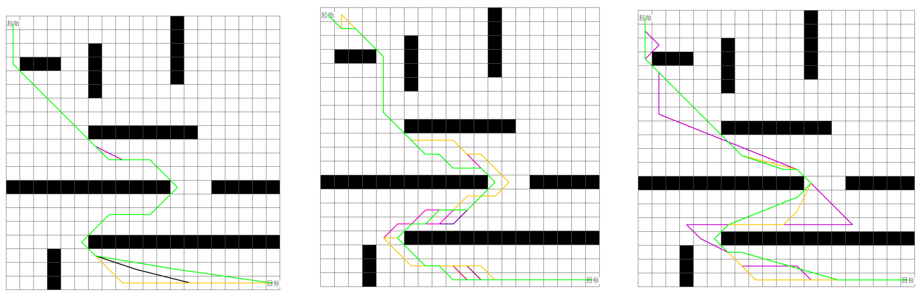
## C. Test 3: Crossover Rate

Test 3 is about the connection between crossover rate and results. Crossover is the major operation of reproduction, so the rate should not be too low. Other parameters: $mutation\ rate = 0.1$; $population\ size = 100$; $generation = 500$. In this test, we used the rate of 0.4, 0.65 and 0.9, and the data is as shown below:

| Crossover rate: | | 0.9 | 0.65 | 0.4 |
|---|---|---|---|---|
| NSGA-II | Length | 0.286 | 0.315 | 0.369 |
| | Smoothness | 0.143 | 0.762 | 0.706 |
| | Security | 0.399 | 0.424 | 0.372 |
| MOEA/D | Length | 0.370 | 0.373 | 0.367 |
| | Smoothness | 0.140 | 0.111 | 0.115 |
| | Security | 0.351 | 0.395 | 0.368 |
| CAEA | Length | 0.351 | 0.355 | 0.355 |
| | Smoothness | 0.151 | 0.194 | 0.192 |
| | Security | 0.344 | 0.364 | 0.378 |

Table 3: Data of different crossover rate.

This table shows us that under the condition of 0.4 and 0.65 crossover rate, NSGA-II generated results which are much worse than under 0.9 crossover rate.



(a) NSGA-II (b) MOEA/D (c) CAEA

Figure 8: Crossover rate: 0.9

In Figure 8 three of them can generate feasible results. However, if the rate is lower, we can see some changes as shown in Figure 9 and Figure 10. In NSGA-II, the individuals with higher diversity is selected by *crowdingdistance* method. But with lower crossover rate, these individuals are hard to preserve.

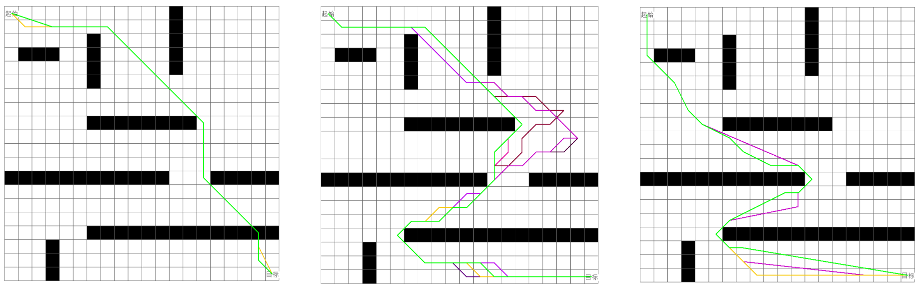

(a) NSGA-II (b) MOEA/D (c) CAEA

Figure 9: Crossover rate: 0.65

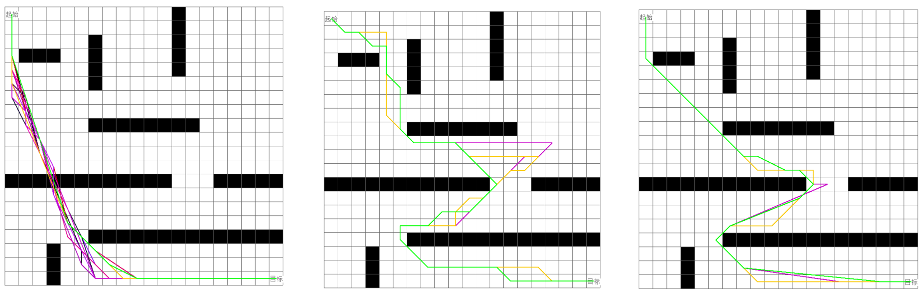

(a) NSGA-II (b) MOEA/D (c) CAEA

Figure 10: Crossover rate: 0.4

These running screenshots illustrate that the diversity preservation of NSGA-II has a closely link to the crossover rate. In some extreme condition, if the crossover rate is not high enough, the diversity preserving efficiency of NSGA-II will decrease, which might lead to some bad solutions. In contrast, MOEA/D and CAEA can still generate feasible results.

## iii Section Conclusion

In this section, we introduce the environment of this experiment and the three tests we did by data table and screenshots. It was shown in the before that comparison of three algorithm in data and figures. This is a direct way to show the difference of three algorithm we used.

## V Conclusion

This article introduced the renowned *Evolutionary Algorithms*: NSGA-II and MOEA/D, and a very new EA based on decomposition: CAEA. In addition, we analysed that how to use EAs to solve a very old and classic problem, Robot Path Planning, by changing it to a *Multiobjective Optimization Problem*. Finally, we did some tests about this problem by using these three EAs, and collected data and then made some conclusions.

NSGA-II, MOEA/D and CAEA, all these three EAs are very suitable for solving Robot Path Planning problem. NSGA-II is a very stable EA. As for this problem, it can generate a set of solutions with relatively high feasibility in a single run. Furthermore, it can optimize the whole population and make them close to the Pareto Front. Nevertheless, the efficiency of NSGA-II and its diversity of solution are not very high.

MOEA/D is an EA based on decomposition, which means it has a very fast performance speed. The theoretical basis and the test results of MOEA/D show that this is a revolutionary EA.

CAEA is proposed by my tutor, professor YING, which also is based on decomposition. It has been proved that CAEA has very low time-consuming compared to NSGA-II and CAEA. This means that CAEA can run faster than NSGA-II and MOEA/D. In this article, we can conclude that the diversity of the solutions of CAEA is higher than NSGA-II and MOEA/D. Diversity is an important indicator of an EA, and this shows the potential of CAEA.CAEA can only solve biobjective optimization problem by now. The research for tri- and multi-objective optimization problem is still ongoing.

Finally, my team and I truly appreciate professor YING and his help for this research project.

## References

[1] Kalyanmoy Deb. Multi-objective optimization. *Multi-objective optimization using evolutionary algorithms*, pages 13–46, 2001.

[2] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Lecture notes in computer science*, 1917:849–858, 2000.

[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[4] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *Evolutionary Computation, IEEE Transactions on*, 11(6):712–731, 2007.

[5] YING Weiqin, XU Xing, FENG Yuxiang, and WU Yu. An efficient conical area evolutionary algorithm for bi-objective optimization. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 95(8):1420–1425, 2012.

[6] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

[7] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer, 1999.