Recent Advances on Practical Evolutionary Optimization

Ke Tang and Xin Yao

USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications (UBRI)
School of Computer Science and Technology
University of Science and Technology of China

August 5, 2013

USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications

Outline

- **Introduction**
- **Algorithm Configuration**
- **Constraints Handling**
- **Evolutionary Multi-Objective Optimization**
- **Scaling Up EAs**
- **Summary**

Outline

- **Introduction**
- Algorithm Configuration
- Constraints Handling
- Evolutionary Multi-Objective Optimization
- Scaling Up EAs
- Summary

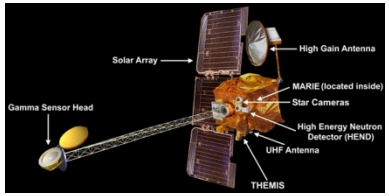
Introduction

- Evolutionary Algorithms (EAs): Algorithms that mimic natural evolution?

- There are several famous EAs with different historical backgrounds.
  - Genetic Algorithms
  - Evolution Strategies
  - Particle Swarm Optimizer
  - Ant Colony Optimization
  - etc.

Introduction

Successful applications of EAs:



Aerospace

G. S. Hornby et al., Automated antenna design with evolutionary algorithms. American Institute of Aeronautics and Astronautics, 2006.

Logistic

Thomas Weise, Alexander Podlich, Kai Reinhard, Christian Gorldt, and Kurt Geihs (2009): "Evolutionary Freight Transportation Planning," in Applications of Evolutionary Computing

Architecture

Ludger Hovestadt. Beyond the Grid - Architecture and Information Technology. Applications of a Digital Architectonic. Birkhäuser Basel / Boston 2009.

Robotics

Zykov V., Mytilinaios E., Adams B., Lipson H. (2005) "Self-reproducing machines", Nature Vol. 435 No. 7038, pp. 163-164

Introduction

- Framework of EAs

1. Generate the initial **population** $P(0)$ at random, and set $i \leftarrow 0$;
2. REPEAT
   (a) Evaluate the fitness of each individual in $P(i)$;
   (b) **Select** parents from $P(i)$ based on their fitness in $P(i)$;
   (c) **Generate** offspring from the parents using *crossover* and *mutation* to form $P(i+1)$;
   (d) $i \leftarrow i+1$;
3. UNTIL halting criteria are satisfied

- Most EAs are Stochastic Population-Based Generate-and-Test algorithms.

Introduction

- Different EAs: variants of the same Generate-and-Test Algorithm, depending on
  - representations,
  - variation operators,
  - selection schemes.

- Talking in classical AI: EAs face the same fundamental issues, i.e., representation and search.

Introduction

- EAs do not require rich domain knowledge to use, although domain knowledge can be incorporated into EAs.

- That means, EAs are particularly suitable for optimization problems whose objective functions are
  - Not differentiable
  - Not continuous
  - Even without explicit mathematical formulations

Introduction

This tutorial concerns the following general questions regarding using EAs in practice.

1. **Algorithm Configuration**: How to configure an EA for my real-world problem?

2. **Constraints Handling**: How to tackle problems with constraints?

3. **Evolutionary Multi-Objective Optimization**: How to simultaneously optimize multiple objective functions?

4. **Scaling Up EAs**: How to address large scale problems with EAs?

Introduction

- Numerical optimization problems are used in most illustrative examples, but many techniques/ideas are readily applicable (or can be easily adapted) to other types of optimization problems.

- Some interchangeably used terminologies
  - Individual ⇔ a solution to an optimization problem ⇔ a point in the solution space (search space)
  - Population ⇔ a set of solutions

Outline

- Introduction
- **Algorithm Configuration**
- Constraints Handling
- Evolutionary Multi-Objective Optimization
- Scaling Up EAs
- Summary

Algorithm Configuration

- Let's start with a very basic optimization problem:

$$\text{minimize} \quad f(x)$$

- To solve it with an EA, one needs to first determine:
  - How to encode (represent) a solution x
  - What is the variation operator and its parameters
  - What is the selection scheme and its parameters

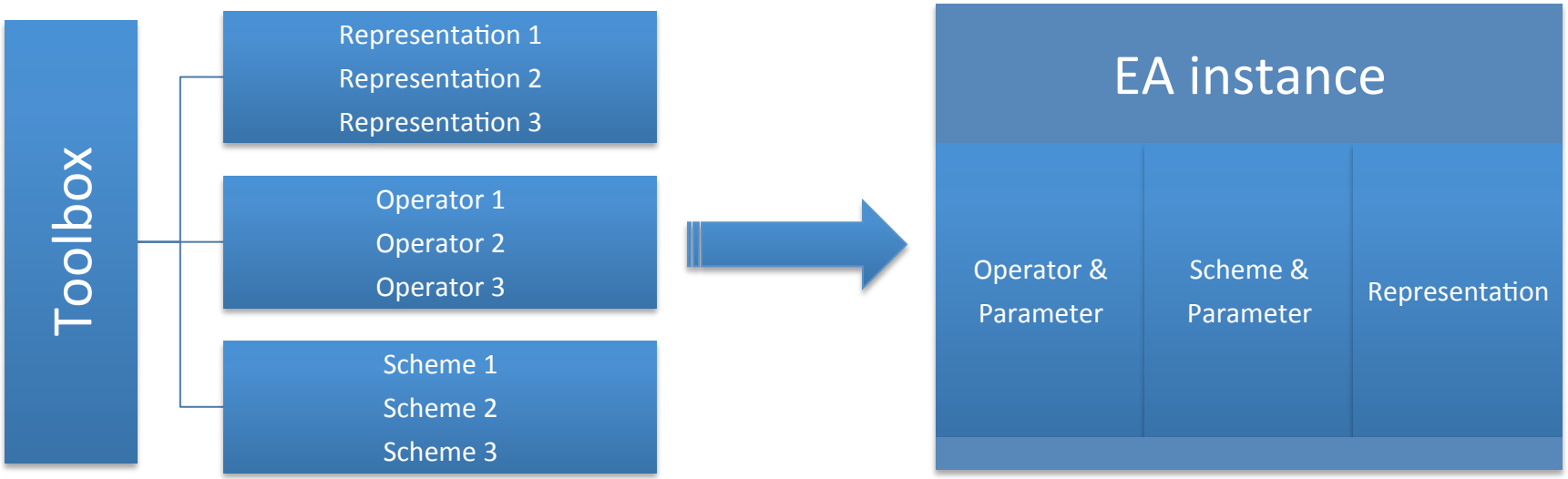
- Design my own problem-specific operators/schemes (ambitious but challenging)

- Choose operators/schemes/parameters from an EA toolbox (more realistic).

Algorithm Configuration

- *Question: If an EA toolbox is available (which is true), which operators/parameters shall I use?*

- Aim: To find an appropriate EA instance (configuration of EA) for an optimization task.



Algorithm Configuration

- Configuring an EA involves setting the values of
  - categorical variables (type of operators/schemes)
  - numerical variables (e.g., crossover rate, population-size)

- We refer both the categorical and numerical variables as parameter of EAs

- More formally, Configuring an EA is another search problem, for which the best parameter vectors are searched to maximize the **performance**.

Algorithm Configuration

- Performance measures may differ in different context:
  - The best solution quality
  - The time required to reach a threshold of solution quality
- Due to the stochastic nature of EAs, a performance measure can be viewed as a random variable.
- Multiple runs are usually required to get a reliable estimate of performance.

Algorithm Configuration

- The rough idea: Generate-and-Test

- Example 1-1: A *naïve* approach
  - Generate *p* candidate parameter vectors
  - Get *p* EA instances, each configured with a parameter vector
  - Run each EA instance for *r* times
  - In each run, *n* solution vectors is generated and tested
  - Compare the parameter vectors using performance measures to get the best configuration.

Algorithm Configuration

- The *naïve* approach requires calculating the objective function $f(x)$ for prn times. – sounds too costly.

- Smarter approach is needed to reduce search effort.
  - Reduce p
  - Reduce r
  - Reduce n

Algorithm Configuration

- Racing procedures: An approach for reducing *r*.
- Basic idea: Stop running (testing) an EA instance if there is indication that it is not promising.
- By not promising, we mean an EA instance performs significantly worse than the best EA instance identified so far.
- EA instances are compared after each run rather than after running all of them for *r* times.

Algorithm Configuration

- Example 1-2:

1. Generate p candidate parameter vectors
2. Set *E* as the set of *p* EA instances
3. REPEAT:
   - Run each EA instance once and update its (average) performance
   - Identify the EA instance with the best performance
   - **Remove** from *E* the EA instances whose performance is significantly worse than the best EA instance
4. Until size(*E*)=1 or the number of runs reaches *r*

- Many parameter vectors may be removed without running *r* times.

Algorithm Configuration

- There are a few statistical tests that can be used to indicate significant difference in performance.
  - Analysis of Variance (ANOVA).
  - Kruskal-Wallis (Rank Based Analysis of Variance).
  - Hoeffding's bound.
  - Unpaired Student T-Test.

Algorithm Configuration

- Racing procedure may not be sufficient in case:
  - p is very large
  - The set of candidate parameter vectors is infinite
- Basic idea: To apply an iterative search procedure to the parameter space.
  - Meta-EA: using EAs to search for the best configuration
  - Iterative local search
- Hopefully, the best configuration can be found with less trial parameter vectors.

Algorithm Configuration

- Example 1-3:

1. Generate *p'* candidate parameter vectors (*p'*<*p*)
2. Get *p'* EA instances
3. REPEAT
   - Run each EA instance for *r* times and assess its performance
   - Generate another *p'* parameter vectors
4. Until halting criterion is satisfied

Algorithm Configuration

- Iterative search can be combined with a racing procedure to reduce both p and r .
- Example 1-4:

1. Generate p' candidate parameter vectors ($p'<p$)
2. Get p' EA instances
3. REPEAT
   - Applying Racing procedure to compare performance of EA instances
   - Generate another p' parameter vectors
4. Until halting criterion is satisfied

Algorithm Configuration

- Further enhancement: Modeling the landscape of performance measures
  - For identifying promising parameter vectors
  - For filtering out unpromising parameter vectors
- The modeling task is a supervised learning problem
  - Input data: parameter vectors
  - Target: performance

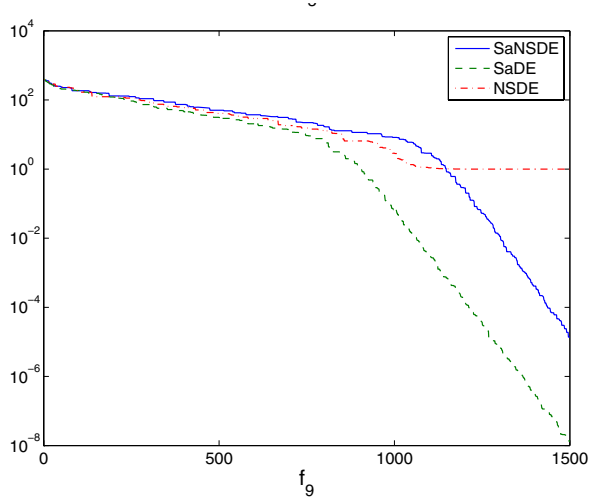
Algorithm Configuration

- Example 1-5 (Sequential Parameter Optimization):

1. Generate *p'* candidate parameter vectors (*p'*<*p*)
2. Get *p'* EA instances
3. Run each EA instance for *r* times and assess its performance
4. Build a model *M* to approximate the performance landscape
5. REPEAT
   - Generate another *p'* parameter vectors
   - Identify the most promising parameter vectors with *M*
   - Only the promising parameter vectors are run for *r* times
   - Update *M* with the newly "assessed" parameter vectors
6. Until halting criterion is satisfied

Algorithm Configuration

- Remark on reducing *n*
  - The key question involved: Will the "relative order" of two EAs maintain unchanged during the evolution?



Yes: Blue curve vs. Green Curve

No: Blue Curve vs. Red Curve

  - The lesson learned: the answer is problem and algorithm dependent, and thus few systematic approach have been investigated for this issue.

Algorithm Configuration

- So far, we mainly touched the case of configuring EA on a single optimization task (problem instance)
- What if we want to find a more general configuration that can be applied to multiple problem instances?
- Previously introduced ideas can be adapted to this scenario.
- Instead of looking at the results of *r* runs, comparisons are made mainly based on the performance on multiple problem instances.
  - F-RACE
  - ParamILS

Algorithm Configuration

- A good configuration for multiple problem instances is attractive from practical viewpoint.
- However…

Algorithm Configuration

- A good configuration for multiple problem instances is attractive from practical viewpoint.

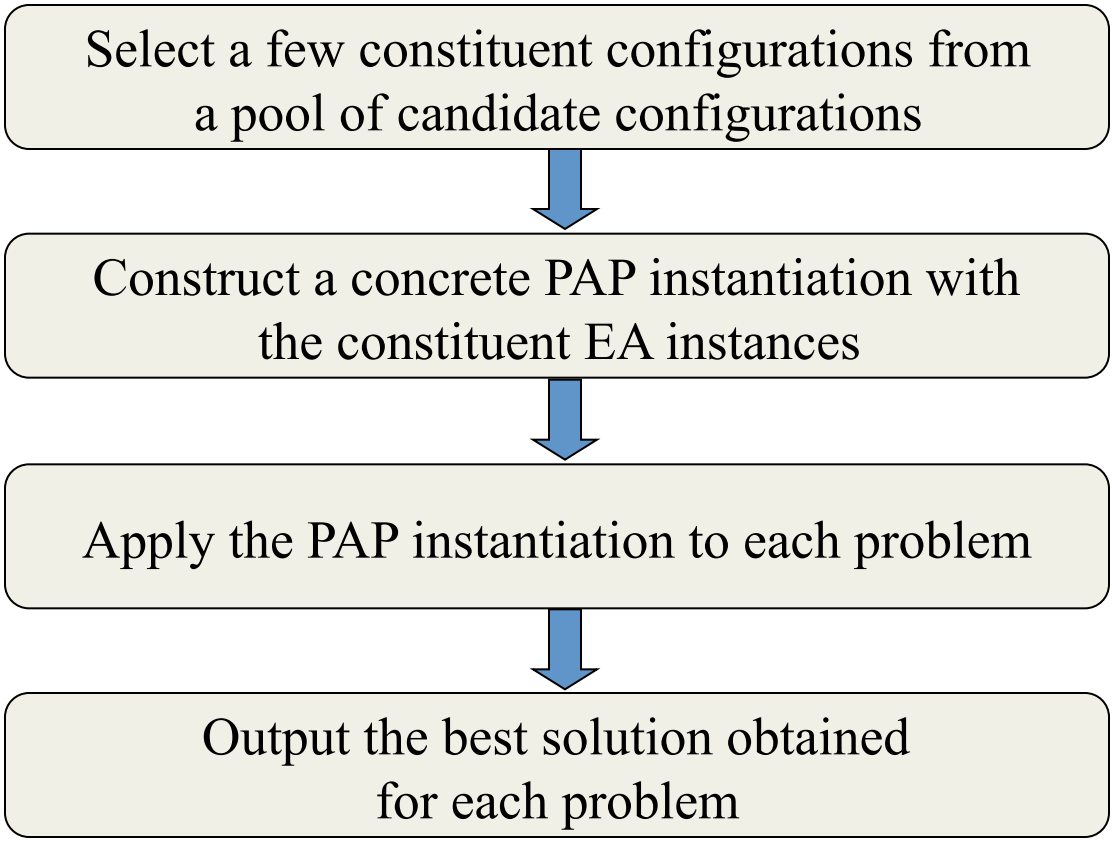
- However…
  - In many scenario it is unlikely such a configuration exists.
  - Advantages of having many alternative configurations are not fully exploited

Algorithm Configuration

- A good configuration for multiple problem instances is attractive from practical viewpoint.
- However…
  - In many scenario it is unlikely such a configuration exists.
  - Advantages of having many alternative configurations are not fully exploited
- How about establishing a good "portfolio" of configurations (e.g., a combination of multiple EA instances)?
  - Making use of advantages of different algorithms, rather than putting all the eggs (time) into a single basket (configuration).
  - Hopefully not too time-consuming since only one portfolio is needed for all problems.

Algorithm Configuration

- Population-based Algorithm Portfolio (PAP)



Algorithm Configuration

- The way of building a PAP instantiation:
  - A PAP instantiation maintains multiple sub-populations.
  - Each sub-population is evolved with a constituent EA instances.
  - Information is shared among sub-populations by activating a migration scheme periodically.

- Sounds simple, more important…
  - If combining different EAs in this way won't lead to any advantage over using a single algorithm. Why bother seeking the so-called constituent algorithms?

Algorithm Configuration

- A preliminary experiment
  – 4 Candidate EA instances: CMA-ES, G3PCX, SaNSDE, wPSO
  – 11 PAP instantiations

- 27 Benchmark problems

- Each PAP instantiation
  - Compared to its constituent configurations alone to verify whether there would be any advantage of PAP over its constituent algorithms.
  - Compared to G-CMA-ES to verify whether portfolio of some "weak" EA instances could outperforms the state-of-the-art.

Algorithm Configuration

- Wilcoxon Test Results (Significance level 0.05): "w-d-l" stands for "win-draw-lose"

- Portfolios of configurations led to the best performance

Value-to-Reach	PAP	DE	PSO	PCX	ES	G-CMA-ES
1.0e−13	DE+PSO	7–15–5	19–7–1	–	–	10–7–10
	DE+PCX	6–18–3	–	18–4–5	–	10–8–9
	DE+ES	10–14–3	–	–	10–9–8	10–9–8
	PSO+PCX	–	7–7–13	13–5–9	–	6–4–17
	PSO+ES	–	17–8–2	–	7–12–8	7–13–7
	PCX+ES	–	–	20–5–2	5–14–8	5–13–9
	DE+PSO+PCX	7–14–6	18–7–2	18–3–6	–	9–7–11
	DE+PSO+ES	11–12–4	21–5–1	–	10–9–8	10–9–8
	DE+PCX+ES	11–14–2	–	21–5–1	10–10–7	10–10–7
	PSO+PCX+ES	–	17–6–4	19–5–3	7–12–8	7–11–9
	DE+PSO+PCX+ES	11–13–3	21–5–1	20–6–1	10–11–6	10–11–6

Algorithm Configuration

- Additional Remarks
  - Operators/parameters can also be adjusted on-the-fly. This type of EAs are usually called adaptive/self-adaptive EAs.
  - With a unified "representation" of EA's behaviors, there will huge room for involving machine learning techniques.
  - Algorithm configuration is also closely related to the emerging terminology "hyper-heuristic".

Outline

- Introduction
- Algorithm Configuration
- **Constraints Handling**
- Evolutionary Multi-Objective Optimization
- Scaling Up EAs
- Summary

Constraint Handling

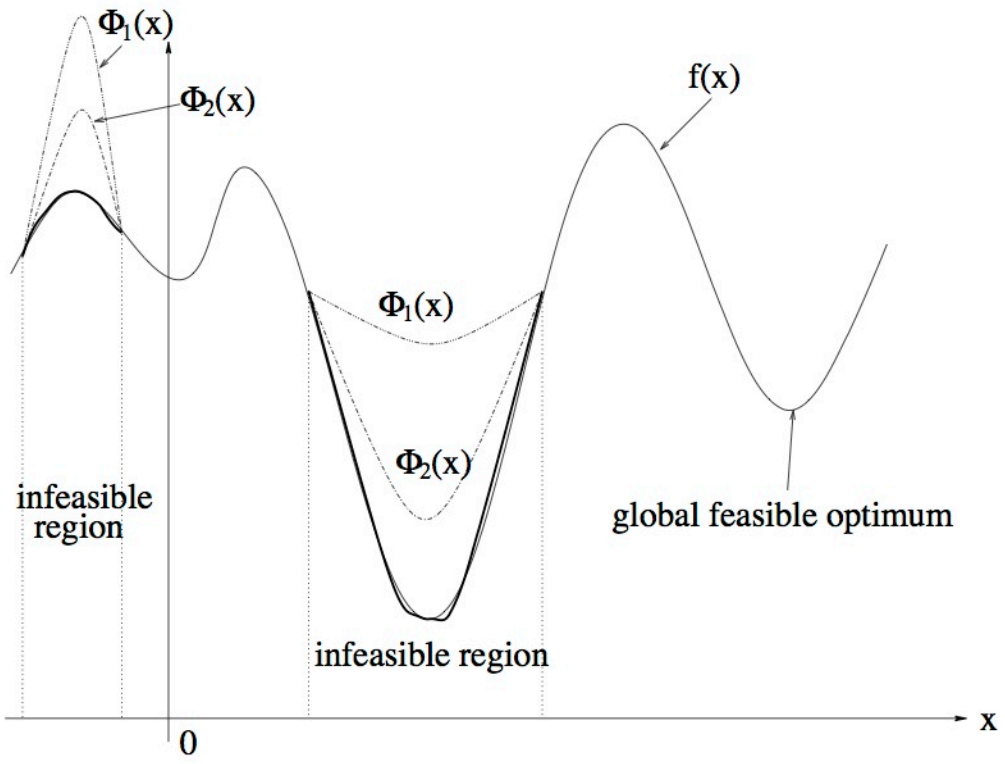
- We now consider a more challenging (realistic) problem:

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to:} \quad g_i(x) \leq 0, \quad i = 1 \ldots m \\ &\qquad\qquad\quad h_j(x) = 0, \quad j = 1 \ldots p \end{aligned}$$

- f(x): the objective function
- g_i(x): the inequality constraints
- h_j(x): the equality constraints

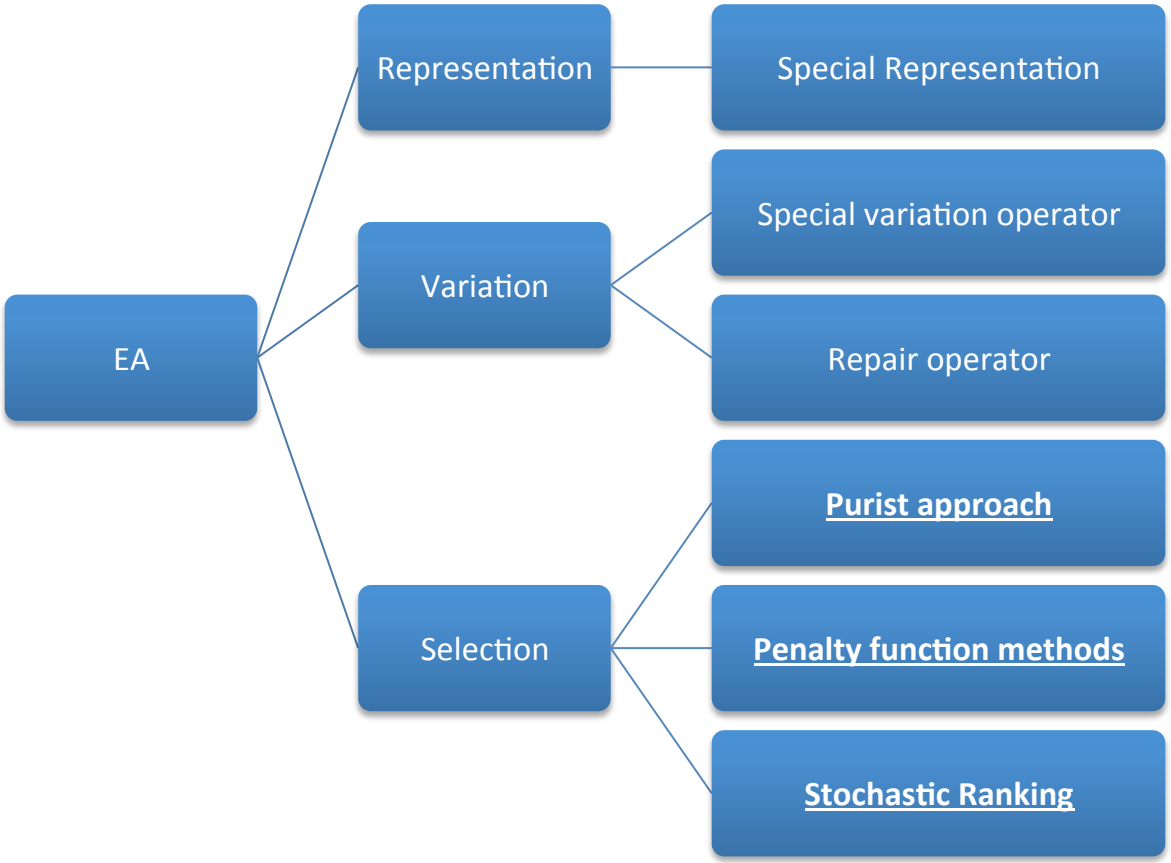
Constraint Handling

- Note: The global optimum in the feasible space might not be the same as that in the whole search space.



Constraint Handling

- Handling constraints in an EA framework



Requires deep domain knowledge, sometimes impossible to design

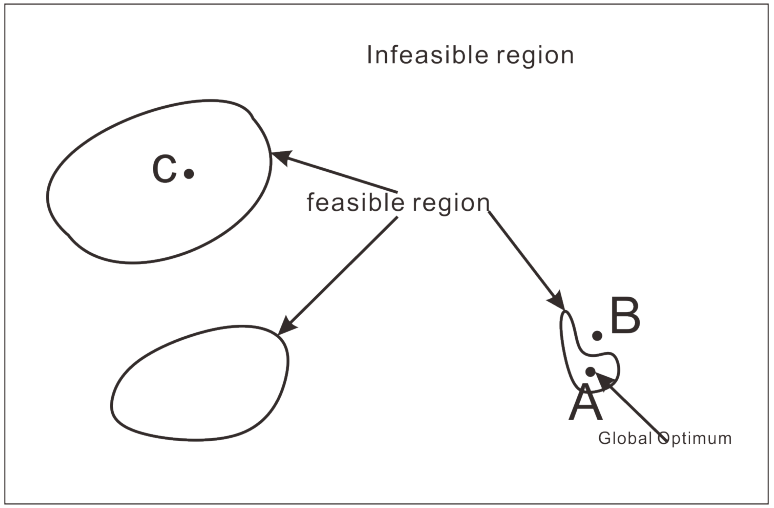
More general and easy to use, focus of this talk

Constraint Handling

- Suppose *n* individuals are to be selected from *N* individuals
- The selection procedure can be conducted via pair-wise comparisons between the *N* individuals
- The purist approach (The naïve approach)
  - Feasible individuals are always preferred over infeasible ones.
  - Between two feasible individuals, the one having a better objective value is preferred.
  - Between two infeasible individuals, the one having smaller constraint violation is preferred.
  - An example of constraint violation: $G_i(x) = \max(0,\ g_i(x))$

Constraint Handling

- Is an infeasible individual really less preferable to a feasible one?
- A counter example (which point is better, B for C?)



- Key issue for constraint handling: balance between objective function and constraints

Constraint Handling

- Penalty function methods:

$$\Phi(x) = f(x) + \sum_{i=1}^{m} r_i G_i(x) + \sum_{j=1}^{p} c_j H_j(x)$$

New objective function

Original objective function

Penalty coefficient

Constraint violation

- where $G_i(x) = (\max(0,\ g_i(x)))^a \qquad H_i(x) = (\max(0,\ h_i(x)))^b$
- Equality constrains can be converted into inequality ones

$$h_j(x) \Rightarrow |h_j(x)| - \varepsilon \le 0$$

Constraint Handling

- Penalty function methods relies on the values of penalty coefficient.
- Static Penalties
  - Penalty coefficients are pre-defined and fixed
- Dynamic Penalties
  - Penalty coefficients changes according to a predefined sequence
- Adaptive and Self-Adaptive Penalties
  - Penalty coefficients changes according to a predefined sequence

Constraint Handling

- Example 2-1: Dynamic Penalty

$$\Phi(x) = f(x) + CT\sum_{i=1}^{m} G_i(x) + \sum_{j=1}^{p} H_j(x)$$

- T: generation number
- C: predefined constant
- *General Principle:* the penalty coefficient increase with T,
- That means, feasibility gradually becomes more and more important (similar to a coarse-to-fine search procedure)

Constraint Handling

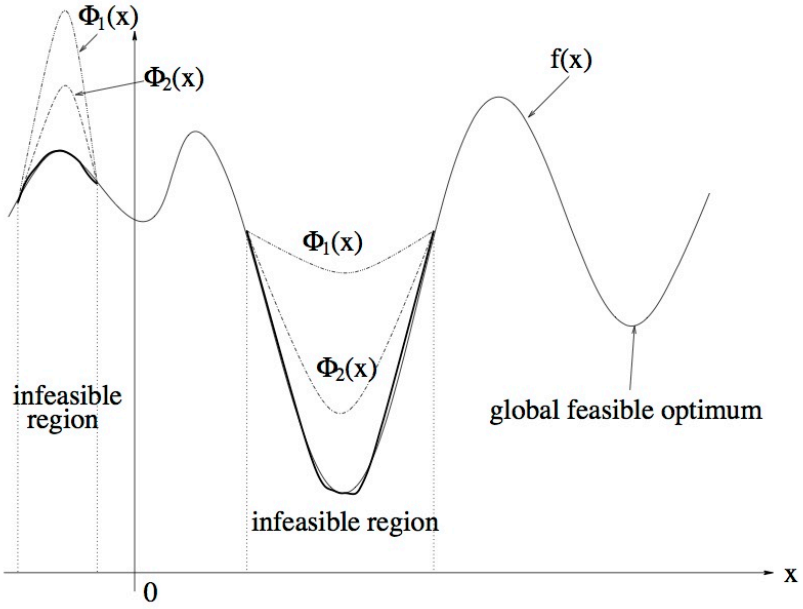
- Example 2-2: Adaptive Penalty

$$\Phi(x) = f(x) + \sum_{i=1}^{m} r_i G_i(x) + \sum_{j=1}^{p} c_i H_j(x)$$

- where $r_i = mean(G_i(x_{old})),\ c_i = mean(H_i(x_{old}))$
- x_{old} is the previously evaluated individuals
- Focus on the constraints that are more difficult to satisfy.
- Also known as co-evolution type approach.

Constraint Handling

- In essence, penalty function methods transform the the fitness landscapes



- Let $\Phi(x) = f(x) + \sum_{i=1}^{m} r_i G_i(x) + \sum_{j=1}^{p} c_i H_j(x) = f(x) + rG(x)$
- What does $\Phi(x_1) < \Phi(x_2)$ mean?

Constraint Handling

$$f(x_1) + rG(x_1) < f(x_2) + rG(x_2)$$

$f(x_1) < f(x_2) \,\&\, G(x_1) < G(x_2) \Rightarrow r$ has no impact on the comparison

$f(x_1) < f(x_2) \,\&\, G(x_1) > G(x_2) \Rightarrow$ increase r will eventually change the comparison

$f(x_1) > f(x_2) \,\&\, G(x_1) < G(x_2) \Rightarrow$ decrease r will eventually change the comparison

- Different *r*'s lead to different rankings of individuals
- Change fitness → Change ranks → Change selection
- Why not modify the ranks of individuals directly?

Constraint Handling

- Example 2-3: Stochastic Ranking
  - Using a bubble-sort-like procedure to rank individuals.
  - In one comparison, either $f(\mathrm{x})$ or $G(\mathrm{x})$ is used, but not both
  - Use a random number u to determine whether $f(\mathrm{x})$ will be use when doing a comparison.
  - Use a parameter P_f to balance between $f(\mathrm{x})$ and $G(\mathrm{x})$
  - P_f is the probability of using $f(\mathrm{x})$ in ranking

Constraint Handling

- Example 2-3: Stochastic Ranking

Input: $x_1 \ldots x_N$

for $i = 1$ to N

  sample $u \in U(0,1)$

  if $G(x_i) = G(x_{i+1}) = 0$ or $(u < P_f)$ then

    if $f(x_i) > f(x_{i+1})$ then

      swap(x_i, x_{i+1})

    end

  else if $G(x_i) > G(x_{i+1})$ then

      swap(x_i, x_{i+1})

  end

end

Constraint Handling

- Recall the key issue: balance between objective function and constraints violation.
- That means, we want to maintain a good trade-off between these two factors.
- The concept of *Pareto dominance* can also be employed to achieve this.
- Treat $G(\mathrm{x})$ as another objective function, which is to be minimized, basically:

$$f(x_1) < f(x_2) \,\&\, G(x_1) < G(x_2) \Rightarrow x_1 \text{ dominates } x_2$$
$$f(x_1) > f(x_2) \,\&\, G(x_1) > G(x_2) \Rightarrow x_1 \text{ is dominated by } x_2$$
$$\text{Otherwise}, x_1 \text{ and } x_2 \text{ are nondominated by each other}$$

Constraint Handling

- How to use Pareto dominance to handle constraints?
- Obviously, it can be used to compare individuals.
- There are a lot of other efforts, we take the so-called Adaptive Tradeoff Model (ATM) as an example.
- Now, let's move our focus from individuals to populations
  - The population contains infeasible individuals only (case 1)
  - The population contains both feasible and infeasible individuals (case 2)
  - The population contains feasible individuals only (case 3)
- The three cases can be addressed with different schemes.

Constraint Handling

- Case 1 (where the Pareto dominance concept take effect):
  - Only non-dominated individuals seem promising.
  - As there is no feasible individuals, we strive to get some ones.
- Identify the non-dominated individuals
- Sort them based on $G(x)$ in ascending order.
- Select the first half of them as the offspring individuals.

Constraint Handling

- Case 2:
  - Transform $f(x)$ of infeasible individuals.

$$f'(x) = \max\{C * f_{\min} + (1 - C) * f_{\max}, f(x)\}$$

$f_{\min}$: the smallest $f(x)$ for feasible individuals

$f_{\max}$: the smallest $f(x)$ for feasible individuals

C : Proportion of feasible individuals in the population

  - Normalize $f(x)$ and $G(x)$ to [0, 1].
  - Combine $f(x)$ and $G(x)$ using a penalty function.

$$f_{final}(x_i) = f_{norm}(x_i) + G_{norm}(x_i)$$

- Case 3: treat as an unconstrained optimization problem

Constraint Handling

- Additional Remarks
  - Stochastic ranking may serve as an off-she-shelf tool, since it is simple to implement, and involves only one parameter P_f.
  - Different types of approaches can be combined to form a hybrid CH technique for a specific problem.
  - Algorithm configuration methods are also applicable to CH techniques selection/configuration.

Outline

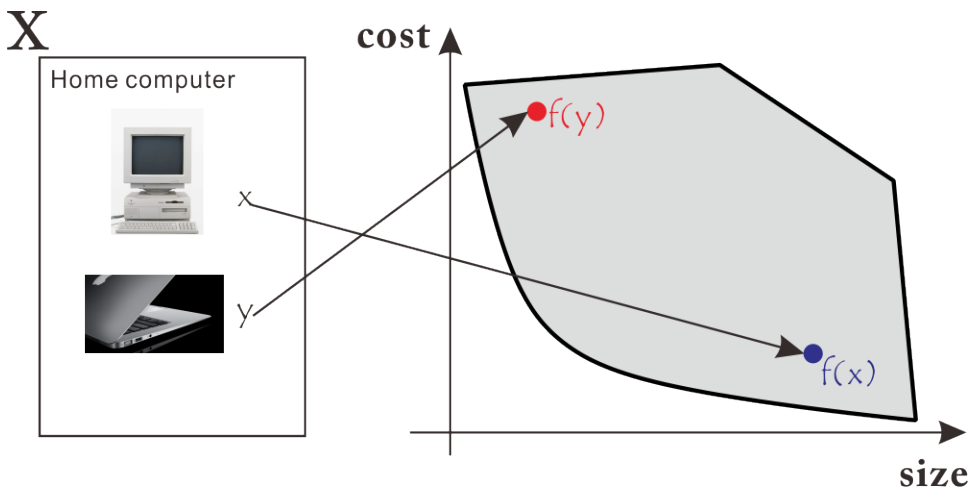
- Introduction
- Algorithm Configuration
- Constraints Handling
- **Evolutionary Multi-Objective Optimization**
- Scaling Up EAs
- Summary

Evolutionary Multi-Objective Optimization

- Multi-objective optimization problem:

$$\text{minimize} \quad F(X) = (f_1(x), \ldots, f_m(x))$$

- The objective functions may be conflicting or incommensurable.
- Instead of a single optimal solution, a set of the best trade-off solutions (Pareto optimal solutions) are sought.

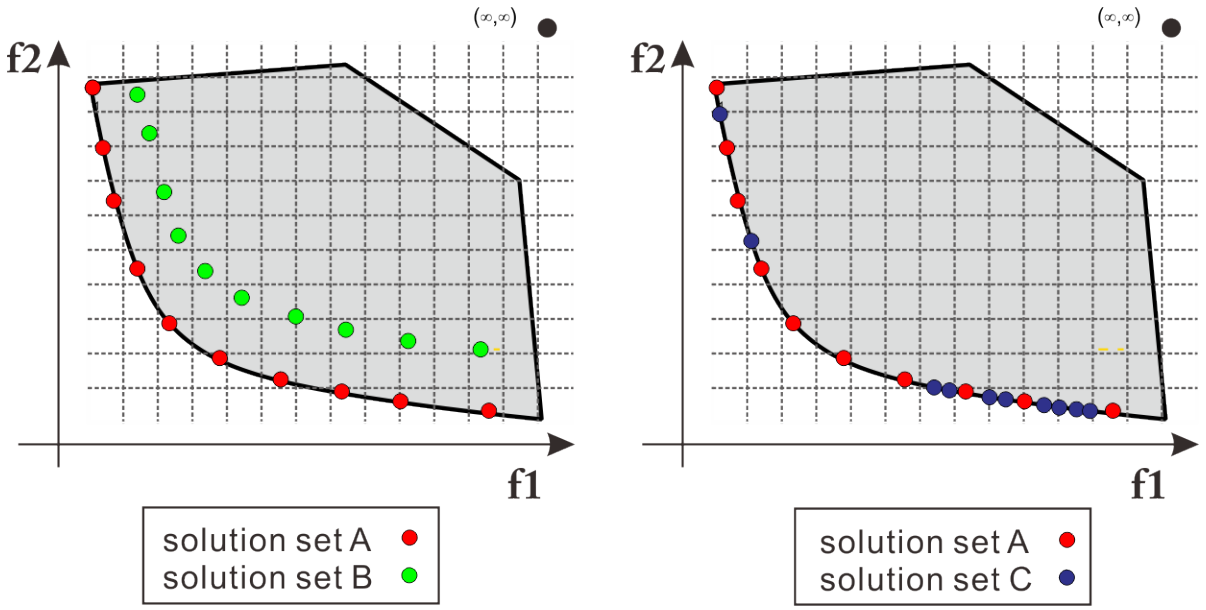


Evolutionary Multi-Objective Optimization

- EAs for MOP are usually called Multi-Objective Evolutionary Algorithms (MOEAs).
- Becoming a prosperous sub-area of EC since 1985.
- More than 5600 papers has been published by January 2011, 66.8% published after 2002.
- Almost all types of EAs have their MO version.
- Quite successful in the real world → attracted many researchers outside the EC-community (e.g., Decision Making).

Evolutionary Multi-Objective Optimization

- Multi-objective optimization itself involves multiple criteria.
- Intuitively, we want a set of solutions that is good in terms of:
  - Convergence (to the Pareto front)
  - Spread (along the Pareto front)



Evolutionary Multi-Objective Optimization

- Our starting point: Non-dominated Sorting Genetic Algorithm II (NSGA-II), published in 2002.
- Probably the most influential work on MOEAs, cited by 9226 (by Google Scholar)
- Majority of papers on MOEAs emerges after this seminal work.
- Many of them adopt similar framework as NSGA-II.

Evolutionary Multi-Objective Optimization

- NSGA-II

Specialized Selection Scheme

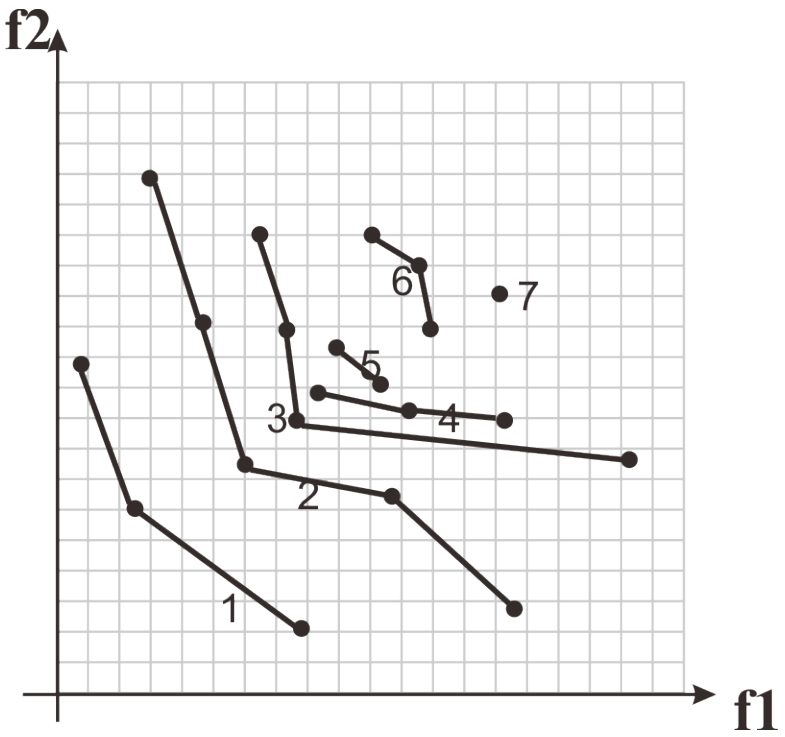
1. Generate the initial **population** $P(0)$ at random, and set $i \leftarrow 0$;
2. REPEAT
   (a) Evaluate the fitness of each individual in $P(i)$;
   (b) **Select** parents from $P(i)$ based on their fitness in $P(i)$;
   (c) **Generate** offspring from the parents using *crossover* and *mutation* to form $P(i+1)$;
   (d) $i \leftarrow i+1$;
3. UNTIL halting criteria are satisfied

- Fast Non-dominated Sorting for selection
  - Divide the population into several fronts by definition of non-domination
  - Crowding distance is utilized to select exact N individuals

Evolutionary Multi-Objective Optimization

- Divide the population into several fronts

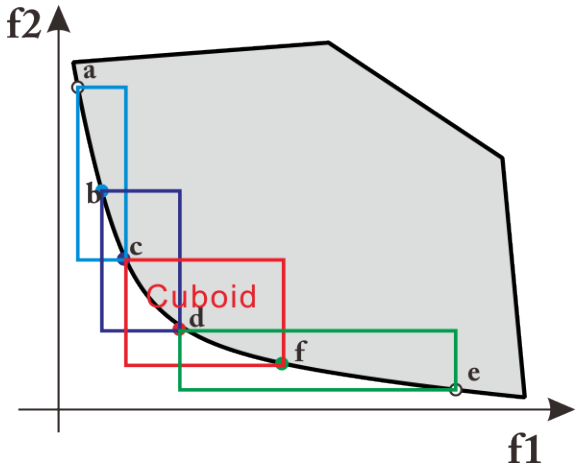
Input: $P = \{x_1 ... x_{2N}\}$
Initialize $k = 1,\ Q = \varnothing$
while $P \neq \varnothing$
    for each $x_i \in P$
        if x_i is not dominated by any $x_j \in P$
            rank(x_i)=k
            $Q = Q \cup \{x_i\}$
        end
    end
    $P = P / Q$
    $k = k + 1$
end



- *Rank* means the index of fronts
- The procedure can be implemented more efficiently (i.e., "Fast")

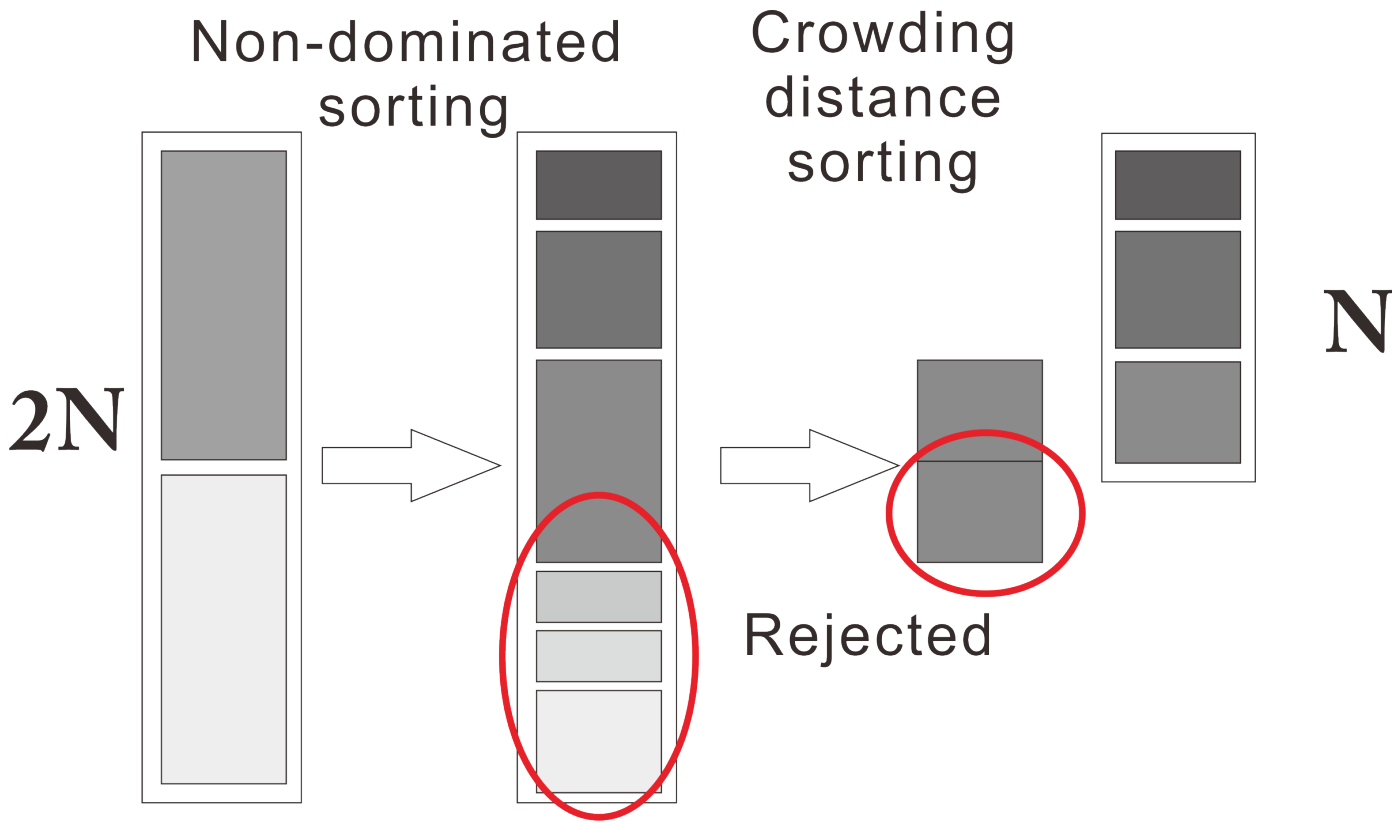
Evolutionary Multi-Objective Optimization

- Individuals are selected according to their *rank* in ascending order.
- Suppose individuals on several fronts have been selected, but not all individuals on the next front can be preserved, since we only need exactly *N* individuals.
- Crowding distance: Select the individuals close to "sparse region" (preserve diversity to get good spread)



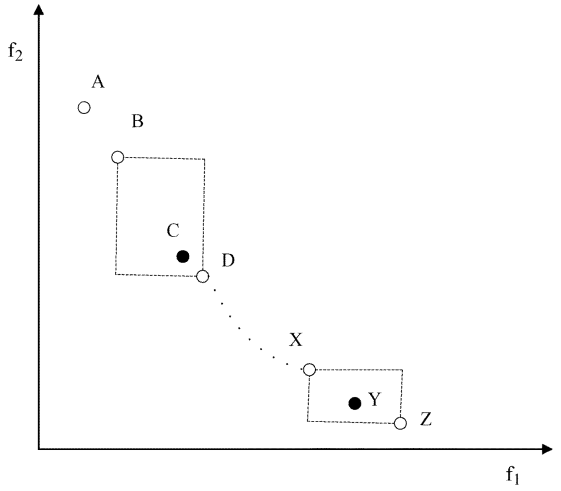
Evolutionary Multi-Objective Optimization

- The selection procedure of NSGA-II



Evolutionary Multi-Objective Optimization

- The crowding distance is a little bit *ad hoc* and may have drawbacks. (Which one should be preserved, point *C* or *Y*)?



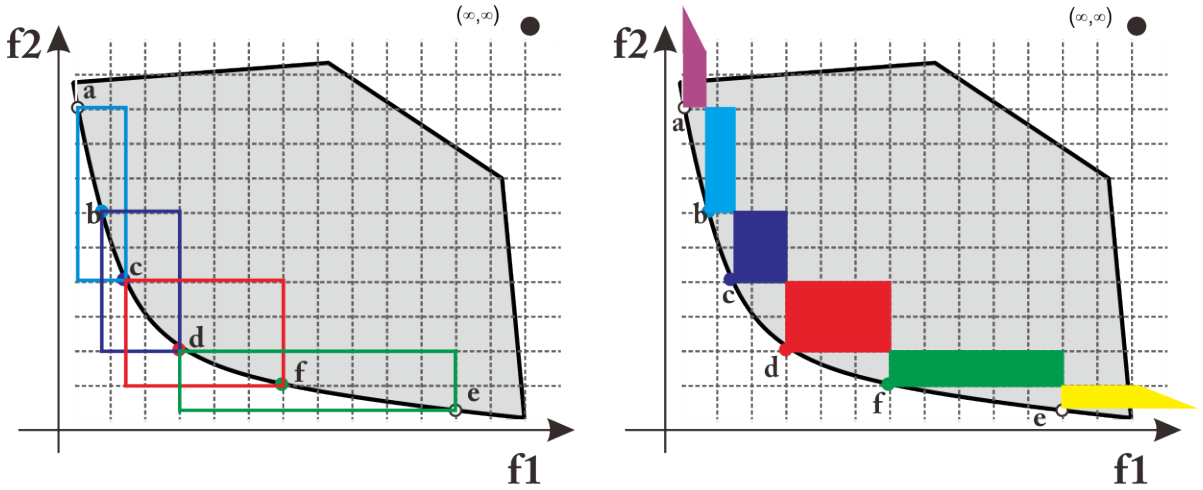
- An (intuitive) question: what is an individual's contribution to "spread".
- In fact, the "spread" itself can hardly be measured (there are several metrics).
- The (essential) source of difficulty: Selection in MOEAs involves comparison of two sets of individuals, rather than two individuals.

Evolutionary Multi-Objective Optimization

- Indicator-based MOEAs
  - Make use of quality indicators to assign every individual a single-objective fitness.
  - Typically, the fitness an individual is defined based on how much the quality indicator decreases if the individual is removed from the population (indicator loss).
  - Only involves pair-wise comparison of individuals
  - General Principle: the quality indicator should be coherent with “convergence” and “spread”.

Evolutionary Multi-Objective Optimization

- The hypervolume indicator
  - Favor large volume of the dominated portion
  - Strict Pareto compliance



Crowding-distance rank : a, e, f, d, c, b
Hypervolume rank : a, e, d, f, c, b

Evolutionary Multi-Objective Optimization

- Example 3-1: Selection with the hypervolume indicator

Input: $P = \{x_1 ... x_{2N}\}$
while $|P| > N$
  for each $x_i \in P$
    calculate its contribution to hypervolume $c(x_i)$
  end
  $x^* = \underset{x}{\operatorname{argmin}}\{c(x_i)\}$
  $P = P / \{x_i\}$
end

  - Calculate the contribution to hypervolume is in general costly, though there exist several more efficient implementations.
  - Example 3-1 is a greedy selection scheme. it may (theoretically) perform arbitrarily bad If more than 1 individual is to be removed.

Evolutionary Multi-Objective Optimization

- MOEA based on Decomposition (MOEA/D): old things become new again.
- Decomposition means convert an MOP into several single objective sub-problems.
- The objective function of each sub-problem is an aggregation of all f_i, i.e.,

$$\Phi(x) = \sum_{i=1}^{m} \lambda_i f_i(x)$$

- Decomposition is a basic idea behind many traditional mathematical programming methods for MOPs.

Evolutionary Multi-Objective Optimization

- MOEA/D: Basic characteristics
- Generating a numbers of even spread weight vectors λ's, each defines a sub-problem.
- Each individual corresponds to a sub-problem.
- The population consists of the best individuals found so far for each sub-problem.
- Maintain an external archive to store all non-dominated solution found during evolution.

Evolutionary Multi-Objective Optimization

- Form of the sub-problems: Tchebycheff approach

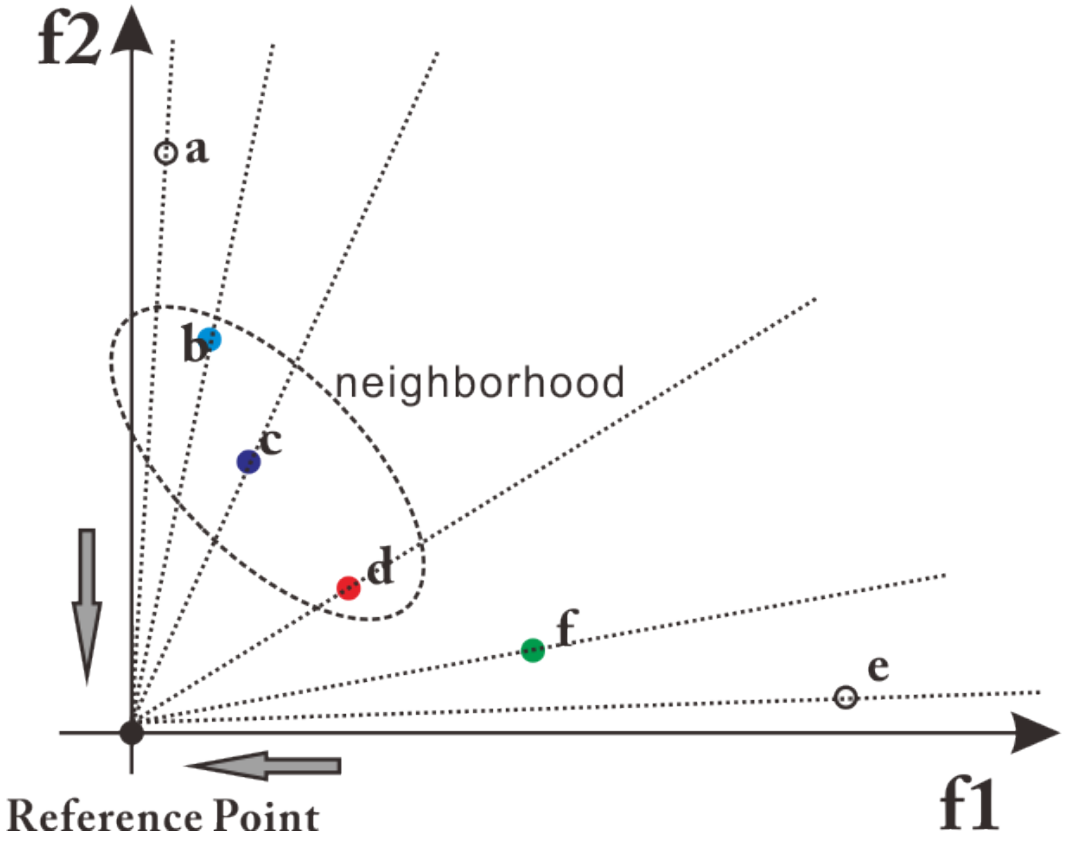
$$\text{minimize} \quad \Phi(x) = \max\left\{\lambda_i \left| f_i(x) - z_i^* \right|\right\}$$

- Since $\Phi(x)$ is continuous of $\boldsymbol{\lambda}$, the optimal solution to two sub-problems should be close to each other if the weight vectors are close to each other.

- Hence, new individuals are generated by applying variation operators to individuals of neighboring sub-problems.

Evolutionary Multi-Objective Optimization

- Illustration of MOEA/D



Evolutionary Multi-Objective Optimization

- Example 3-1: An implementation of MOEA/D
- Initialization
  1. Generate N uniformly spread weight vectors
  2. For each sub-problem, identify T neighboring sub-problems by calculating the Euclidean distance between weight vectors
  3. Initialize $\mathbf{z}^*=[z_1 \ldots z_m]$ and N individuals x

- Repeat until halting condition is satisfied
  1. For each individual, randomly select two neighboring sub-problems, applying variation operator to the 2 corresponding individuals, get x'
  2. Update $\mathbf{z}^*$: $z_i^*=\min\{f_i(\mathrm{x}), z_i^*\}$,
  3. If x' is better than an individual corresponding to a neighboring sub-problem, replace that individual with x'.
  4. Update the external archive.

Evolutionary Multi-Objective Optimization

- Additional Remarks
  - We described frameworks rather than specific MOEAs. Numerous concrete implementations can be developed by incorporating existing operators and schemes (e.g., the constraints handling techniques).
  - A post-processing procedure is required to finally pick out a solution that suits the real-world application the best.
  - So far, most successful stories of MOEAs are for problems with less than 5 objective functions.

Outline

- Introduction
- Algorithm Configuration
- Constraints Handling
- Evolutionary Multi-Objective Optimization
- **Scaling Up EAs**
- Summary

Scaling Up EAs

- Although EAs have achieved great success in the domain of optimization, most reported studies are obtained using small scale problems (e.g. numerical optimization with $D < 100$).

- The term "large scale" may refers to
  - Large number of design variables (e.g., high-dimensional solution space)
  - Large number of constraints
  - Large number of objective functions (MOP)

- Different types of large scale problem needs tailored techniques to scale up EAs. We focus on high-dimensional single objective numerical optimization problem.

Scaling Up EAs

- Are all problems of thousands of variables difficult?
  - No!
  - Only Nonseparable problems are difficult.
- A function $f(\mathrm{x})$ is separable iff [2]

$$\underset{(x_1,\cdots,x_D)}{\operatorname{argmin}} f(x_1,\cdots,x_D) =$$

$$\left(\arg\min_{x_1} f(x_1,\cdots), \ldots, \arg\min_{x_D} f(\cdots,x_D)\right)$$

- Examples:
  - Separable: $F_{rastrigin}(\mathrm{x}) = \sum_{\mathrm{i}=1}^{\mathrm{D}} [\mathrm{x_i^2}\text{-}10\cos(2\pi \mathrm{x_i}) + 10]$
  - Nonseparable: $F_{schwefel}(\mathrm{x}) = \sum_{\mathrm{i}=1}^{\mathrm{D}} (\sum_{\mathrm{j}=1}^{\mathrm{i}} \mathrm{x_j})^2$

Scaling Up EAs

What Makes Large Scale Problems Difficult?

- Solution space often increases exponentially with the growth of problem dimensionality.
- Problem complexity may increase with the growth of dimensionality, e.g., the number of local optima.
- Candidate search directions often increase exponentially. EAs might fail to find the promising search directions.
- Fitness evaluation may also becomes more costly.

- Basic (old) idea: divide-and-conquer.

Scaling Up EAs

Cooperative Coevolution (CC) for large scale problems.

- Decomposes the objective problem into some sub-problems;
- Evolves each sub-problem separately using EAs;
- Combines the solutions to all sub-problems to form the solution to the original problem.

Design Issues of CC for Large Scale Optimization

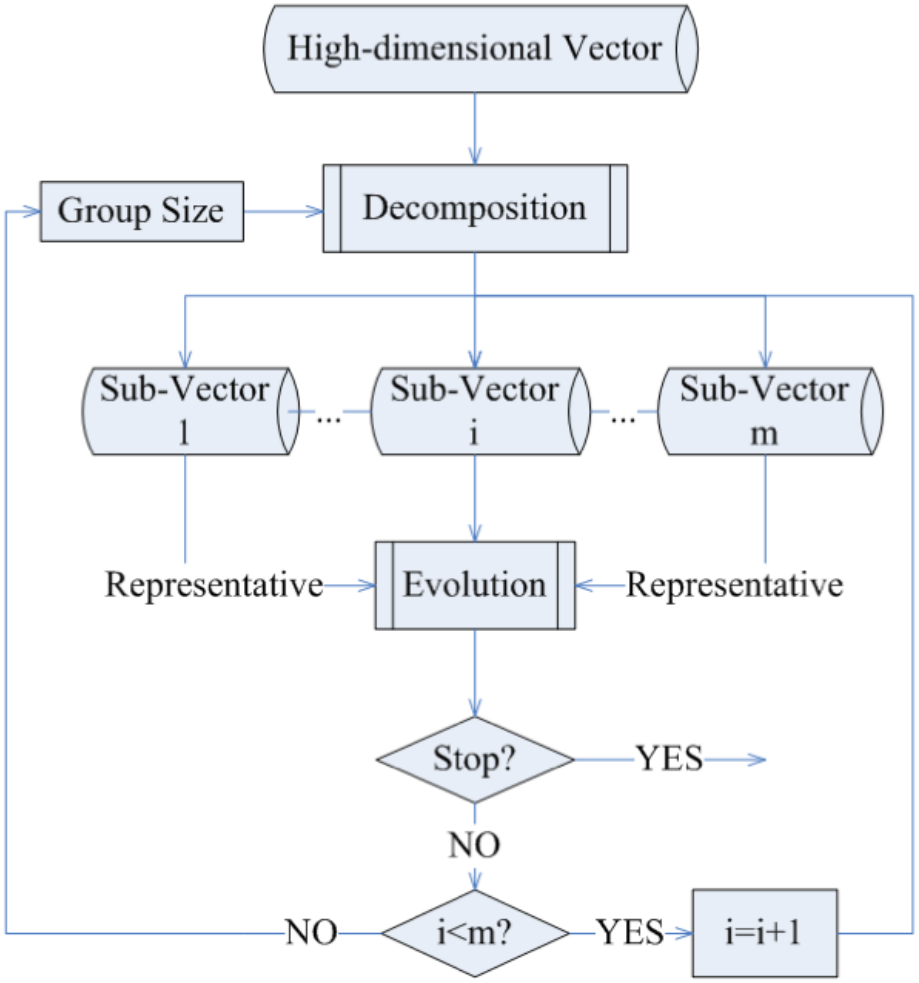
- How to group interacting decision variables into the same group?
- How to optimize each subgroup of decision variables?
- How to evaluate and combine solutions for sub-problems?

Scaling Up EAs

CC with random grouping (EACC-G):

- Without sufficient prior knowledge, the simplest way is to group decision variables randomly.
- The problem phase is done in a predefined number of cycles. Each cycle consists of the following steps:
  - Split D decision variables into m groups, each contains s variables.
  - Optimize each sub-problem with an EA.
  - Solutions for each sub-problem is evaluated by combining with the best solution obtained for the other sub-problems.
  - Assign a weight to each sub-problem.
  - Evolve the weight vectors for the best, the worst and a random members of current population.

Scaling Up EAs



Scaling Up EAs

The probability of EACC-G to assign two interacting variables x_i and x_j into the same group for at least k cycles is:

$$P_k = \sum_{r=k}^{N} \binom{N}{r} (\frac{1}{m})^r (1 - \frac{1}{m})^{N-r}$$

N: Number of Cycles; *m*: Number of Groups

- For example, given a 1000-D problem, when m = 10, P_1 =0.9948, P_2 =0.9662
- Even the simple random grouping strategy has some chance to group two interacting variables together.

Scaling Up EAs

- Test Suite: 13 minimization problems (1000-dimensional).
- Applying SaNSDE to the problem directly.
- DECC-G: using SaNSDE as basic optimizer.
- The numbers of FEs were set to 5e+06 for all algorithms.
- Results of 25 independent runs were collected for each problem.

Scaling Up EAs

Comparison between DECC-G and SaNSDE on functions f1 − f7 (unimodal), with dimension D = 1000, averaged over 25 runs.

	# of Dim	SaNSDE	DECC-G
f1	1000	6.97E+00	**2.17E-25**
f2	1000	1.24E+00	**5.37E-14**
f3	1000	6.43E+01	**3.71E-23**
f4	1000	4.99E+01	**1.01E-01**
f5	1000	3.31E+03	**9.87E+02**
f6	1000	3.93E+03	**0.00E+00**
*f*7	1000	1.18E+01	**8.40E-03**

Scaling Up EAs

Comparison between DECC-G and SaNSDE on functions f8 − f13 (multimodal), with dimension D = 1000, averaged over 25 runs.

	# of Dim	SaNSDE	DECC-G
*f*8	1000	-372991	**-418983**
*f*9	1000	8.69E+02	**3.55E-16**
*f*10	1000	1.12E+01	**2.22E-13**
*f*11	1000	4.80E-01	**1.01E-15**
*f*12	1000	8.97E+00	**6.89E-25**
*f*13	1000	7.41E+02	**2.55E-21**

Scaling Up EAs

Drawbacks of EACC-G:

- Require predefined size of group, which is difficult to determine beforehand
- Assume the sizes of all groups are equal
- The nature of random grouping remarkably limits the chance of categorizing all interacting variables into the same group

Scaling Up EAs

Variable Interaction Learning - A Bottom-Up Approach

1. Start by treating each decision variable as a single group
2. learn the interactions between variables
3. combine interacting variables into the same group
4. goto step 2 until stopping criterion is met

Target

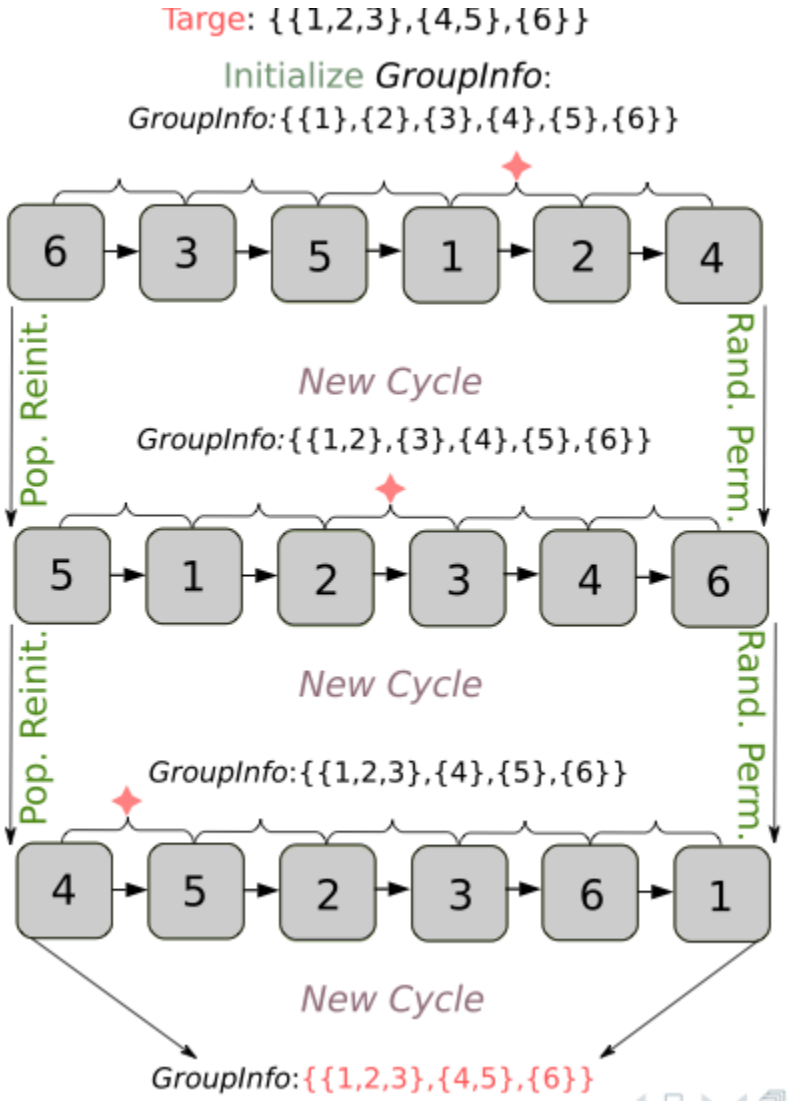
1. provide a fine-grain learning mechanism on the separability
2. every variable interaction learned by the mechanism is correct
3. make use of the separability information gathered by learning

Scaling Up EAs

Cooperative Coevolution with Variable Interaction Learning

1. **Initialization**: Randomly initialize a population of solutions, and randomly choose an individual from the population.
2. **Learning Stage**: Repeat a number of learning cycles, each leaning cycle consists of three steps:
   (1) Randomly permute the sequence of decision variables
   (2) Scan over the permuted decision variables sequence to check the interaction between each pair of successive variables. If evidence of interaction is discovered, mark the two variables as "belonging to the same group".
3. **Optimization Stage**:
   (1) Categorize the decision variables according to the information obtained in the learning stage
   (2) Solve the problem using CC framework

Scaling Up EAs



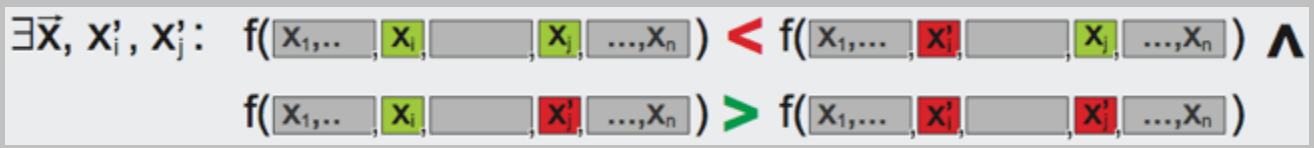
Scaling Up EAs

Definition

A function is separable, if it satisfies the Equation [9]:

$$\arg \min_{(x_1,\ldots,x_N)} f(x_1,\ldots,x_N) = (\arg \min_{(x_1)} f(x_1,\ldots),\ldots,\arg \min_{(x_N)} f(\ldots,x_N))$$

Two decision variables *i* and *j* are interacting if there is a decision vector $\vec{X}$ whose *i*th and *j*th variable can be substituted with values xi′ and xj′ so that Equation holds.[10]

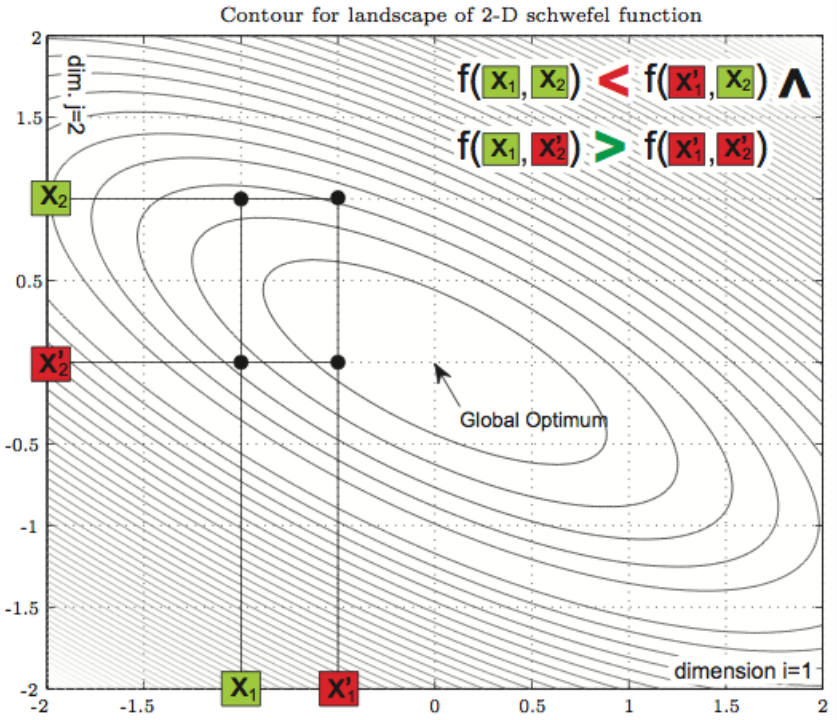


Scaling Up EAs

Example

1. suppose we start with (x_1, x_2) and (x_1', x_2), the Global Optimum is located in (0,0), and it is a minimization problem
2. $f(x_1, x_2) \leq f(x_1', x_2)$
3. move both the point along one axis towards the global optimum
4. relation changes $f(x_1, x_2') \geq f(x_1', x_2')$.

This is how we learn the interaction in the case of 2-D Schwefel 1.2 Function.



Scaling Up EAs

The Learning stage costs FEs and a trade-off between learning and evolution (optimization) needs to be set.

Appropriate setting for learning cycle can deal with both separable functions and non-separable functions:

Termination Conditions for Learning Stage

- If no interactions were learned after $K^{\check{}}$ cycles, we treat it as separable function and thus the learning stage will terminate.
- If any interaction has been learned before reaching the $K^{\check{}}$ cycles, we treat it as a non-separable function. In this case, learning stage only stops if:
  - all N dimensions have been combined into one group
  - 60% of FEs has been consumed in learning stage

Scaling Up EAs

- CEC2010 Test Suite: 20 minimization problems (1000-*D*) with different degree of separability
- Basic optimizer: JADE.
- The numbers of FEs were set to 3e+06 for all algorithms.
- Results of 25 independent runs were collected for each problem.

Scaling Up EAs

	CCVIL		DECC-G		MLCC		Comparison Result
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	
f_1	1.55e-17	7.75e-17	2.93e-07	8.62e-08	1.53e-27	7.66e-27	–
f_2	6.71e-09	2.31e-08	1.31e+03	3.24e+01	5.55e-01	2.20e+00	W
f_3	7.52e-11	6.58e-11	1.39e+00	9.59e-02	9.86e-13	3.69e-12	L
f_4	5.00e+12	3.38e+12	9.62e+12	3.43e+12	1.70e+13	5.38e+12	W
f_5	1.76e+08	6.47e+07	2.63e+08	8.44e+07	3.84e+08	6.93e+07	W
f_6	2.94e+05	6.09e+05	4.96e+06	8.02e+05	1.62e+07	4.97e+06	W
f_7	8.00e+08	2.48e+09	1.63e+08	1.38e+08	6.89e+05	7.36e+05	–
f_8	6.50e+07	3.07e+07	6.44e+07	2.89e+07	4.38e+07	3.45e+07	–
f_9	6.66e+07	1.60e+07	3.21e+08	3.39e+07	1.23e+08	1.33e+07	W
f_{10}	1.28e+03	7.95e+01	1.06e+04	2.93e+02	3.43e+03	8.72e+02	W
f_{11}	3.48e+00	1.91e+00	2.34e+01	1.79e+00	1.98e+02	6.45e-01	W
f_{12}	8.95e+03	5.39e+03	8.93e+04	6.90e+03	3.48e+04	4.91e+03	W
f_{13}	5.72e+02	2.55e+02	5.12e+03	3.95e+03	2.08e+03	7.26e+02	W
f_{14}	1.74e+08	2.68e+07	8.08e+08	6.06e+07	3.16e+08	2.78e+07	W
f_{15}	2.65e+03	9.34e+01	1.22e+04	9.10e+02	7.10e+03	1.34e+03	W
f_{16}	7.18e+00	2.23e+00	7.66e+01	8.14e+00	3.77e+02	4.71e+01	W
f_{17}	2.13e+04	9.16e+03	2.87e+05	1.97e+04	1.59e+05	1.43e+04	W
f_{18}	1.33e+04	1.00e+04	2.46e+04	1.05e+04	7.09e+03	4.77e+03	–
f_{19}	3.52e+05	2.04e+04	1.11e+06	5.00e+04	1.36e+06	7.31e+04	W
f_{20}	1.11e+03	3.04e+02	4.06e+03	3.66e+02	2.05e+03	1.79e+02	W

W implies that CCVIL is significantly better than the other two algorithm.

L represents CCVIL is significantly worse than at least one compared algorithms.

"–" means the difference is insignificant.

Scaling Up EAs

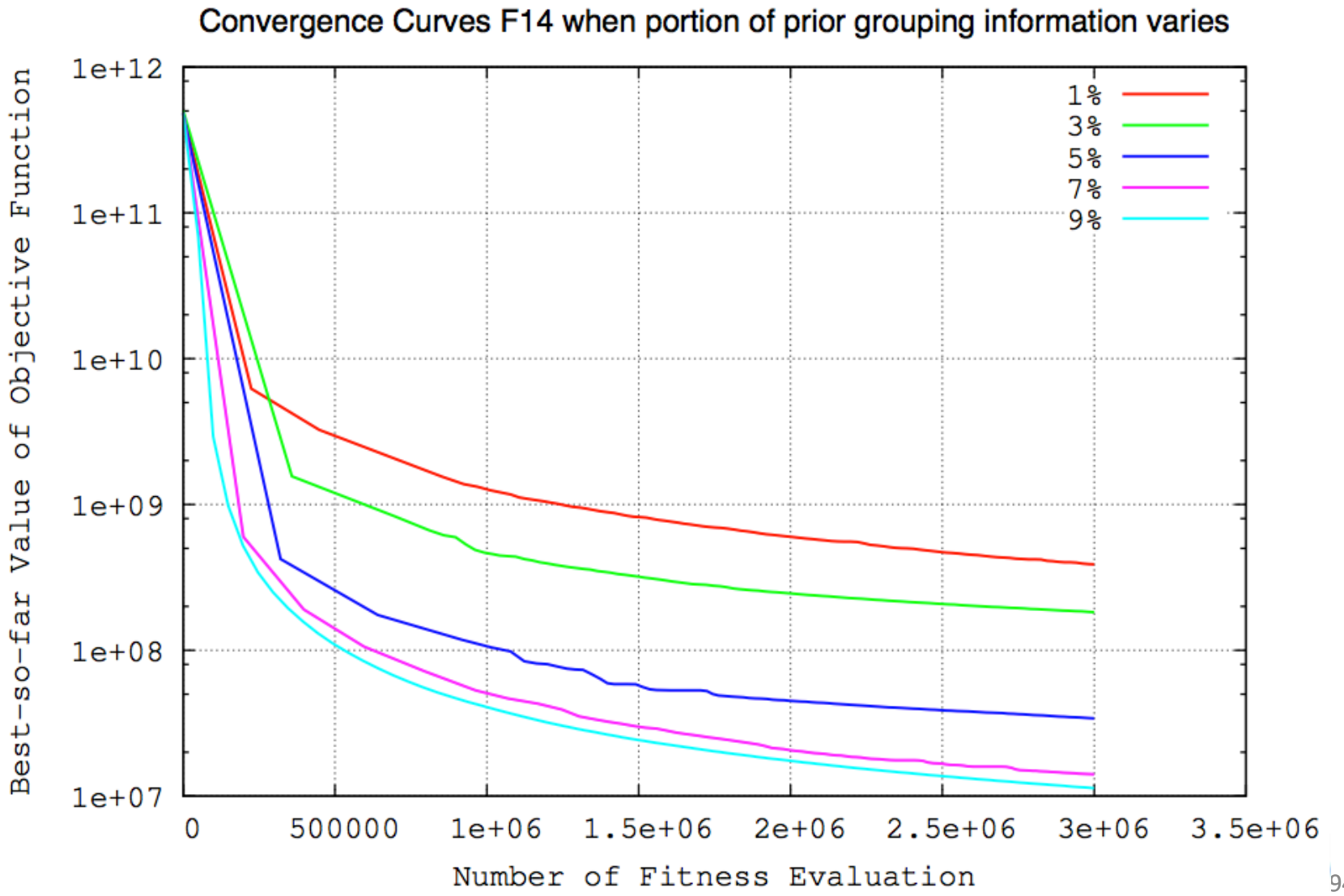
Are Interactions between variables **worthy of learning**?

- Learning is itself a hard problem.
- Unlikely that all the interactions can be learned.
- Is partial interaction information still beneficial?

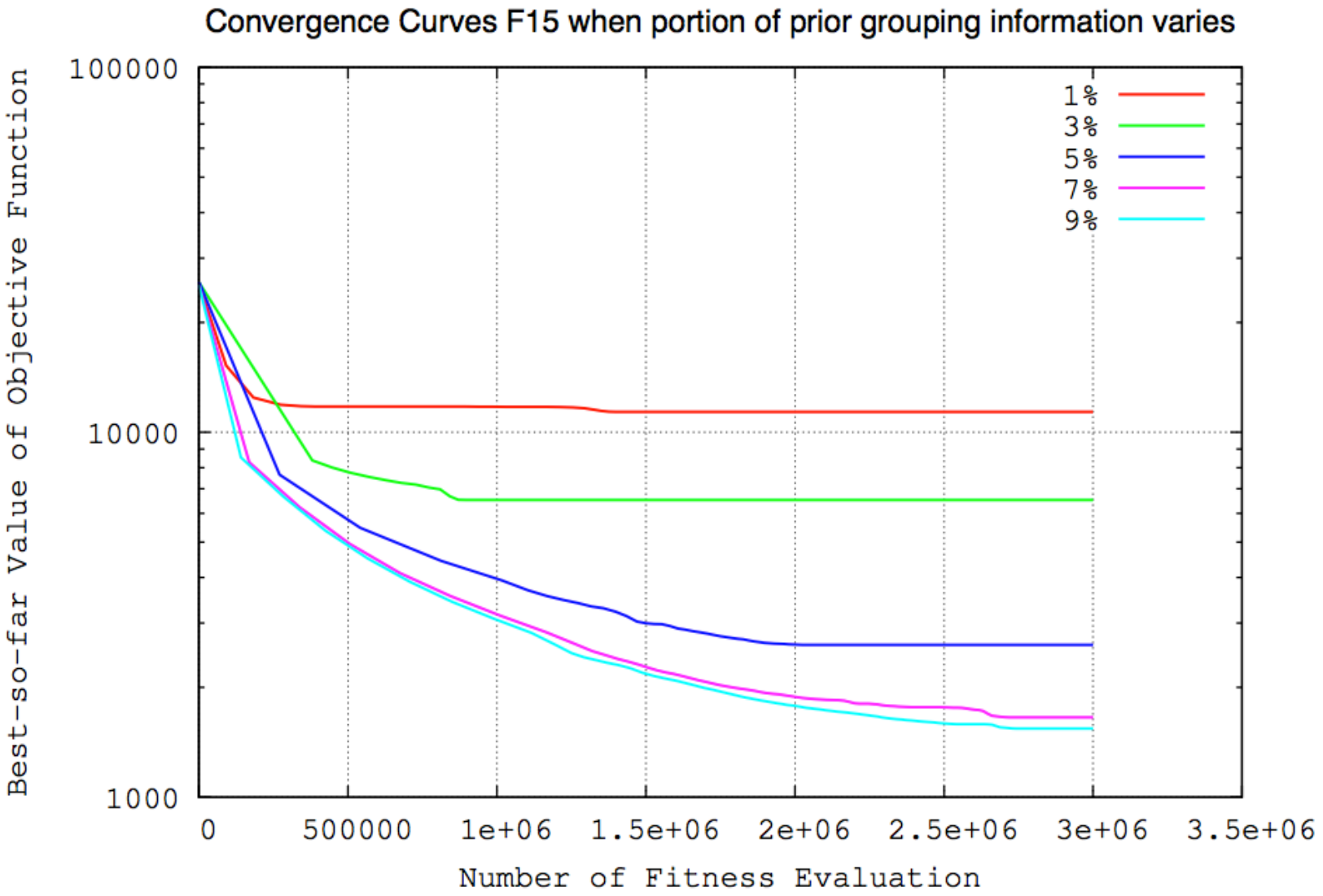
A preliminary experiments:

- For a problem with D variables, interaction information can be represented as a binary string of length D^2
- Tune the "portion" of prior grouping information.
- Would more prior grouping information lead to better performance?

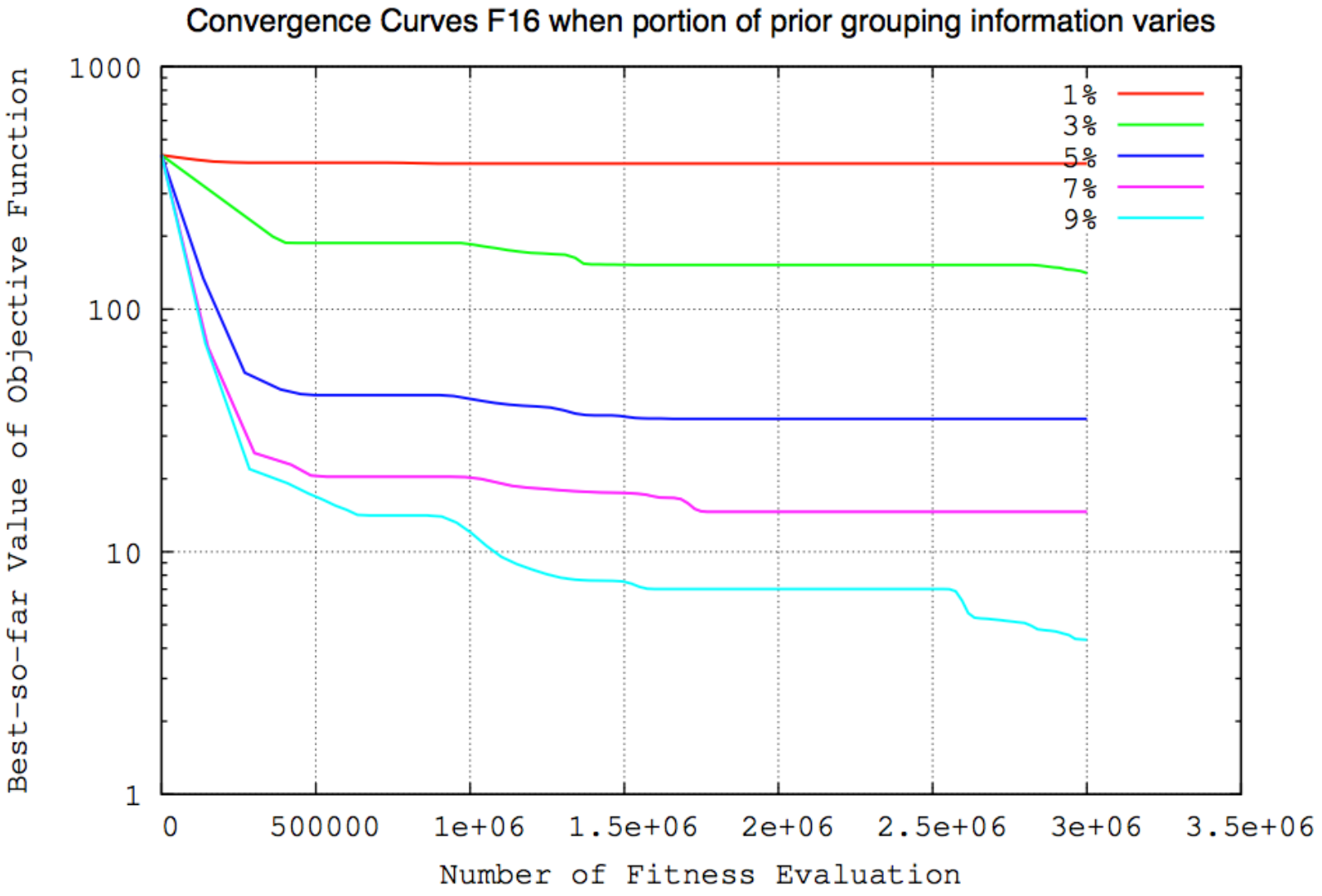
Scaling Up EAs



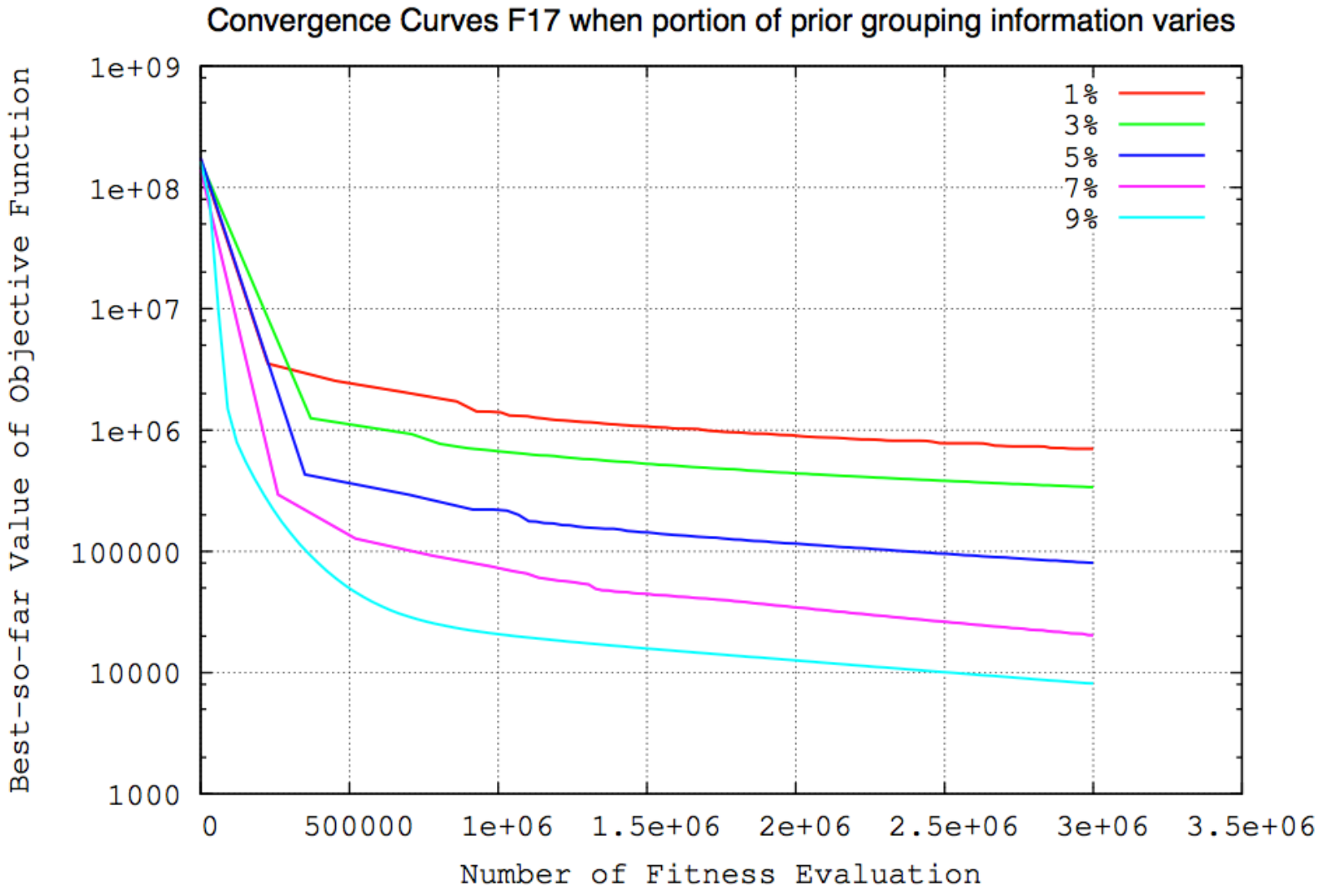
Scaling Up EAs



Scaling Up EAs



Scaling Up EAs



Scaling Up EAs

- Additional Remarks
  - Distributed/parallel structure is another way for scaling up EAs. These methods usually divide the population (into several sub-populations), rather than the problem.
  - How to divide a problem is problem-dependent (especially for combinatorial optimization).
  - Scaling up EAs in case of many objective functions or constraints is also attracting more and more investigations.

Outline

- Introduction
- Algorithm Configuration
- Constraints Handling
- Evolutionary Multi-Objective Optimization
- Scaling Up EAs
- **Summary**

Summary

We revisited the most commonly encountered issues when using EAs (as a search method) in practice:

- How to determine the appropriate EC model, search operators and schemes for our specific task.
- How to handle constraints in the framework of EAs.
- How to deal with multi-objective optimization problems.
- How to improve the efficiency of EAs if we are unsatisfied with their performance on some large scale problems.

Summary

Some important Topics that are not cover by this tutorial:

- Handling Uncertainty
  - Robust optimization
  - Dynamic optimization
  - Robust optimization over time (ROOT)
- Integrating EAs with other search techniques
  - EA+ local search (especially useful for combinatorial optimization)
  - Memetic Algorithm
  - Surrogate Assisted EA

References for Further Reading

Algorithm Configuration

1. A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, Vol. 1, no. 1, pp. 19-31, 2011.

2. T. Bartz-Beielstein, C. Lasarczyk and M. Preuss, "Sequential parameter optimization," in *Proceedings of 2005 IEEE Congress on Evolutionary Computation (CEC'05)*, Edinburgh, Scotland, IEEE Press, vol. 1, pp. 773–780, 2005.

3. F. Hutter, H. Hoos and T. Stützle, "Automatic algorithm configuration based on local search," in *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI-07)*, pp. 1152–1157, 2007.

4. M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, "F-race and IteratedF-Race: An overview," in T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (eds), *Experimental Methods for the Analysis of Optimization Algorithms*, Springer, Berlin, Germany, pp. 311–336, 2010.

References for Further Reading

Constraints Handling

1. C. A. Coello Coello, “Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art,” *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11/12, pp. 1245–1287, 2002.

2. T. P. Runarsson and X. Yao, “Stochastic Ranking for Constrained Evolutionary Optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 284-294, 2000.

References for Further Reading

Evolutionary Multi-Objective Optimization

1. A. Zhou, B. Y. Qu, H. Li, S. Z. Zhao, P. N. Suganthan and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation,* vol. 1, no. 1, pp. 32-49, 2011.

2. Q. Zhang and H. Li, "MOEA/D:Amultiobjectiveevolutionaryalgorithm based on decomposition," *IEEE Transactions Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

3. K. Bringmann, T. Friedrich, F. Neumann and M. Wagner, "Approximation-guided evolutionary multi-objective optimization," in *Proceeding of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-2011)*, pp. 1198–1203, Barcelona, Spain, 2011.

References for Further Reading

Scaling Up EAs

1. Z. Yang, K. Tang and X. Yao, "Large Scale Evolutionary Optimization Using Cooperative Coevolution," *Information Sciences*, vol. 178, no. 15, pp. 2985-2999, 2008.

2. W. Chen, T. Weise, Z. Yang and K. Tang, "Large-Scale Global Optimization using Cooperative Coevolution with Variable Interaction Learning," in *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN)*, Kraków, Poland, September 11–15, 2010, pp. 300–309, Lecture Notes in Computer Science, Volume 6239, Part II, Springer-Verlag, Berlin, Germany.

Thanks for your time!

Q&A?