Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

Dynamic Optimization in Fast-Changing Environments via Offline Evolutionary Search

Xiaofen Lu, Ke Tang, Stefan Menzel, Xin Yao

2021

Preprint:

This is an accepted article published in IEEE Transactions on Evolutionary Computation. The final authenticated version is available online at: https://doi.org/10.1109/TEVC.2021.3104343 Copyright 2021 IEEE

Dynamic Optimization in Fast-Changing Environments via Offline Evolutionary Search

Xiaofen Lu, Ke Tang, *Senior Member, IEEE*, Stefan Menzel, and Xin Yao, *Fellow, IEEE*

The article has been accepted for publication in the IEEE Transactions on Evolutionary Computation.

Copyright notice:

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

DOI: 10.1109/TEVC.2021.3104343

Dynamic Optimization in Fast-Changing Environments via Offline Evolutionary Search

Xiaofen Lu, Ke Tang, *Senior Member, IEEE*, Stefan Menzel, and Xin Yao, *Fellow, IEEE*

***Abstract*—Dynamic optimization, for which the objective functions change over time, has attracted intensive investigations due to the inherent uncertainty associated with many real-world problems. For its robustness with respect to noise, Evolutionary Algorithms (EAs) have been expected to have great potential for dynamic optimization. Many dynamic optimization methods such as diversity-driven methods, memory methods, and prediction methods have been proposed based on EAs to deal with environmental changes. However, they face difficulties in adapting to fast changes in dynamic optimization as EAs normally need quite a few fitness evaluations to find a near-optimum solution. To address this issue, this paper proposes a new framework of applying EAs in the context of dynamic optimization to deal with fast changing environments. We suggest that, instead of online evolving (searching) solutions for the ever-changing objective function, EAs are more suitable for acquiring an archive of solutions in an offline way, which could be adopted to construct a system to provide high-quality solutions efficiently in a dynamic environment. To be specific, we first re-formulate a dynamic optimization problem as a static set-oriented optimization problem. Then, a set of solutions are obtained by an EA for this set-oriented optimization problem. After this, the obtained solution set is adopted to do fast adaptation to the corresponding dynamic optimization problem. The general framework is instantiated for continuous dynamic constrained optimization problems, and the empirical results show the potential of the proposed framework.**

***Index Terms*—Dynamic Optimization, Set-Oriented Optimization, Local Search, Dynamic Constrained Optimization, Negatively Correlated Search**

I. Introduction

DYNAMIC optimization problems (DOPs) widely exist in the real world due to changing environments. For DOPs, their objective functions, constraints or even decision variables change over time, which results in changing globally optimal solutions. For example, in vehicle routing problems, the number of vehicles, the vehicle load or speed will change over time due to the wear of vehicles, meanwhile the customers' requirements or the traffic condition also change over time, which will affect the currently best delivery route continuously [1]. The challenge of DOPs exists in that an optimizer needs to fast respond to the ever-changing environment, i.e. quickly finding a new satisfying solution once the problem changes [2].

As a class of derivative-free population-based optimization methods, evolutionary algorithms (EAs) have been widely studied in the field of dynamic optimization. In the literature, many dynamic optimization methods have been developed based on EAs. Generally, these methods can be grouped into three categories: diversity-driven methods, memory methods and prediction methods [2]. Prediction methods were proposed for predictable DOPs that change over time following certain rules. They implement the prediction of new optimal solutions based on previously obtained best solutions and utilize the predicted solutions for population re-initialization once the problem changes [3], [4], [5]. Diversity-driven and memory methods are more appropriate for unpredictable DOPs. Diversity-driven methods comprise methods that focus on smart ways to introduce diversity into the population after a change happens [6], [7], maintain diversity of the population during search [8], [9], and utilize multiple subpopulations to locate and track multiple optima simultaneously [10], [11]. Memory methods store and use good-performing solutions from previous search to cope with the change of the problem [12]. They are suitable for periodical DOPs.

Although these evolutionary dynamic optimization methods differ in certain aspects, from a unified perspective, they all rely on potentially good solutions stored from the past to enhance their ability of solving DOPs in an online fashion. By online, we mean the phase in which the algorithm needs to face environmental changes and respond accordingly. However, these methods face challenges when the environments of DOPs change fast. As EAs usually need a lot of fitness evaluations to achieve a near-optimum solution, when facing fast-changing environments, the existing methods have too limited time to find satisfactory solutions for the current environment and for later use when the environment changes. Considering this, in the present paper we move one step further along the same direction of thinking, that is, we propose a novel framework for applying EAs in the context of DOPs by evolving a set of good solutions *offline* first, which builds the basis for an efficient *online* optimization of the DOPs. By offline, we mean the phase before the whole system runs (i.e. before the algorithm faces environmental changes). In this framework, our realization of the first part, i.e. the offline evolutionary search process, can be considered as the system design phase and is the biggest difference to existing evolutionary dynamic optimization methods.

The above proposed framework of applying EAs to solve DOPs might induce higher computational overhead in total

X. Lu, K. Tang (*corresponding author*) and X. Yao are with the Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China. S. Menzel is with Honda Research Institute Europe, 63073 Offenbach/Main, Germany. X. Yao is also with the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, U. K. *E-mails*: luxf@sustech.edu.cn, tangk3@sustech.edu.cn, stefan.menzel@honda-ri.de, xiny@sustech.edu.cn

(i.e. computational cost in the system design phase and computational cost for solving DOPs online). However, the additional overhead could bring critical benefits. To be more specific, by investing computational resources to obtain an archive of solutions in advance, the computational time for solving DOPs online could be significantly alleviated. The prepared solution set allows the system to respond to an environmental change with a fast (ideally in real-time) output of high-quality soluitons, which is an essential requirment for most solver systems targeting DOPs, e.g. real-time vehicle routing systems in logistics [1] and high-frequency trading systems [13].

Our main contributions in the present paper comprise:

1) The formalization of the problem statement as a static set-oriented optimization problem.
2) The proposal of a concrete and operable framework for solving DOPs, which combines an offline system design phase and an online optimization phase. The offline computation aims to evolve a set of good initial solutions in the system design phase and the online optimization aims to perform a fast adaptation to environmental changes based on the archived solutions while the system is running.
3) The instantiation of a specific algorithm for continuous dynamic constrained optimization problems (DCOPs) based on the proposed framework and validation of its efficiacy on a test suite of DCOPs.

The remainder of this paper is organized as follows. Section II presents the problem formulation. Section III introduces the proposed dynamic optimization framework followed by Section IV which provides details on an instantiation algorithm of the framework for DCOPs. In Section V, the experimental results and analysis on the DCOP test benchmark are discussed. Section VI concludes this paper and ideates on future work directions.

II. Problem Formulation

Without loss of generality, the DOPs considered in this paper have the following formulation:

$$\max_{\boldsymbol{x}} \ f(\boldsymbol{x}, \boldsymbol{\alpha}(t)) \tag{1}$$

where $\boldsymbol{x} = [x_0, x_1, ..., x_{D_x}]^T$ denotes the decision variable vector and D_x is the dimension of the search space, $\boldsymbol{\alpha}(t) = [\alpha_0(t), \alpha_1(t), ..., \alpha_{D_\alpha}(t)]$ is the vector of environmental variables that vary at a certain frequency as time goes by, and D_α is the number of environmental variables. In this work, for brevity, we assume that D_x, D_α, the search space of $\boldsymbol{x}$ and the variation space of $\boldsymbol{\alpha}$ are known beforehand. Actually, this work can be extended to DOPs with also these variables being subject to changes over time.

This work provides a new dynamic optimization framework that solves fast-changing DOPs by offline obtaining a solution set and using this solution set as starting population for fast online optimization when the environment changes. The solution set is expected to contain some individuals that are close to the optimal solution(s) for each environmental change so that conducting local search from one of these individuals can easily find the new optimal solution. Considering this, the problem to search such a solution set for the DOP defined in Eq. (1) can be formulated as follows:

$$\begin{aligned} &\max_{X} \ F(X, \boldsymbol{\alpha}(t)), \forall t \\ &\text{where } F(X, \boldsymbol{\alpha}(t)) = \max_{\boldsymbol{x} \in X} f(\boldsymbol{x}, \boldsymbol{\alpha}(t)) \end{aligned} \tag{2}$$

where $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{set\_size}\}$ denotes the solution set to search and set_size denotes the number of solutions in X.

As only the variation range for $\boldsymbol{\alpha}$ is known and the value of $\boldsymbol{\alpha}(t)$ at each time step t is unknown beforehand, we can solve the problem in Eq. (2) by considering all possible values for $\boldsymbol{\alpha}(t)$, i.e. by solving the following optimization problem:

$$\begin{aligned} &\max_{X} \ F(X, \boldsymbol{\alpha}), \forall \boldsymbol{\alpha} \\ &\text{where } F(X, \boldsymbol{\alpha}) = \max_{\boldsymbol{x} \in X} f(\boldsymbol{x}, \boldsymbol{\alpha}) \end{aligned} \tag{3}$$

However, in this problem, the optimization process is time intractable if every possible value of $\boldsymbol{\alpha}$ is considered, especially in a continuous variation space. To solve this issue, this work proposes to do sampling on $\boldsymbol{\alpha}$ to control the computational overhead explicitly. Therefore, the optimization problem in Eq. (3) is transformed to:

$$\begin{aligned} &\max_{X} \ \sum_{i=1}^{env\_size} F(X, \boldsymbol{\alpha}_i) \\ &\text{where } F(X, \boldsymbol{\alpha}_i) = \max_{\boldsymbol{x} \in X} f(\boldsymbol{x}, \boldsymbol{\alpha}_i) \end{aligned} \tag{4}$$

where env_size denotes the number of sampled environmental values and $\boldsymbol{\alpha}_i$ is the i-th sampled environment for $\boldsymbol{\alpha}$.

Through Eq. (4), the DOP defined in Eq. (1) is re-formulated as a static set-oriented optimization problem. By solving the problem in Eq. (4) to obtain a set of solutions in advance, we will show in the following parts the advantages of the new dynamic optimization framework in adapting to fast-changing environments.

III. General Framework of Set-based Dynamic Optimization

In this section, a new dynamic optimization framework based on EAs is introduced, which is called Set-based Dynamic Optimization framework (SDO). The SDO workflow is depicted in Fig. 1.

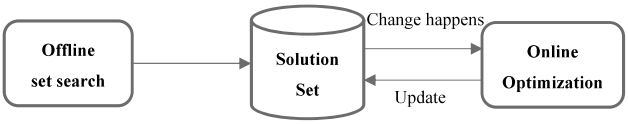


Fig. 1. The workflow of the SDO framework

The general idea of SDO for solving DOPs follows a combination of (1) an offline search for a solution set by using EAs to solve the problem in Eq. (4) before the whole system runs (i.e. system design phase), which is then stored in an archive, and (2) an online optimization that solves the DOP using the archived solutions. In the online optimization phase (i.e. when the system runs), whenever a problem is detected, the online search is restarted to find a satisfying

solution based on one or more solutions in the archive for the changed problem. Moreover, the solution set in the archive is also updated according to newly encountered environmental changes.

Algorithm 1 provides the pseudo-code for the complete dynamic optimization framework. In this algorithm, D_x and D_α are the dimensions of the solution space and environment space for the DOPs, respectively. The SDO applies an offline evolutionary process to search a solution set X in the system design phase. After this, X is used to solve the dynamic optimization online by conducting local search on some individuals in X. To detect the problem changes, $detect_k$ individuals are randomly generated from the decision space at the beginning and re-evaluation of them is conducted each time. In the remaining part, both the offline and online optimization phases are further detailed.

Algorithm 1 The SDO Framework

Input:
The decision space for the solution: $\mathbb{R}^{D_x}$
The change space for the environmental variables: $\mathbb{R}^{D_\alpha}$
The maximum number of generations: G_{max}
The number of individuals to detect changes: $detect_k$
The number of individuals to delete from initial population: del_size

Perform offline evolutionary search based on $\mathbb{R}^{D_x}$ and $\mathbb{R}^{D_\alpha}$ for G_{max} generations to get a solution set X
Archive the solution set X
Set $pop = X$
Evaluate pop with f under the current environment $\boldsymbol{\alpha}$
Delete the worst del_size individuals from pop
Randomly generate $detect_k$ sentinel solutions and evaluate them
while stopping criteria is not satisfied **do**
  for each individual $\boldsymbol{x}_i$ in pop **do**
    Do local search to generate a new individual: $\boldsymbol{x}_i'$
    if $\boldsymbol{x}_i'$ is better than $\boldsymbol{x}_i$ under the current environment **then**
      Set $\boldsymbol{x}_i = \boldsymbol{x}_i'$
    end if
  end for
  Re-evaluate $detect_k$ sentinel solutions to detect changes
  if change is detected **then**
    Update the solution set X with pop
    Set $pop = X$
    Evaluate pop with f under the current environment $\boldsymbol{\alpha}$
    Delete the worst del_size individuals from pop
  end if
end while

A. Offline Set Search

The offline set search phase aims to solve the optimization problem in Eq. (4) to obtain a set of solutions. There exist two questions to answer when facing this problem. One is how to perform sampling on environmental variables $\boldsymbol{\alpha}$ and the other is how to search a set of solutions X according to the sampled environments. In this work, the most straightforward sampling strategy, uniform sampling, is applied to sample $\boldsymbol{\alpha}$. Uniform sampling implies that every point in the environment space has the same probability to be sampled. After obtaining the sampled environments, a set-oriented optimization process is conducted to search X with the given size set_size.

Algorithm 2 provides the set-oriented optimization process based on EAs. At the beginning, a set of env_size environmental values, $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$, and a set of set_size solutions, $P_0 = \{\boldsymbol{x}_{j,0} | j = 1, 2, ..., set\_size\}$, are randomly generated from the environment space and decision space, respectively. The solution set X is set to P_0. After the initialization, at every generation G, evolutionary operations, e.g. mutation and crossover, are conducted on the current population P_G to generate an offspring population O_G. Then, P_G and O_G are combined to constitute U. The fitness of individuals in U are evaluated according to their contribution to the performance of U on the sampled environments $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$. The contribution of each individual $\boldsymbol{x}_j$ in U is defined as the performance drop of U on $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$ after deleting $\boldsymbol{x}_j$ from U. Hence,

$$fit(\boldsymbol{x}_j) = \sum_{i=1}^{env\_size} (F(U, \boldsymbol{\alpha}_i) - F(U \backslash \boldsymbol{x}_j, \boldsymbol{\alpha}_i)) \quad \text{where } F(U, \boldsymbol{\alpha}_i) = \max_{\boldsymbol{x} \in U} f(\boldsymbol{x}, \boldsymbol{\alpha}_i) \tag{5}$$

After fitness evaluation, set_size individuals are selected according to their fitness to enter the next generation as P_{G+1}. Then, the generated offspring population O_G is used to update X. First, O_G is added to X. Then, the contribution of each individual in X is evaluated based on their contribution to the performance of X on $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$ and set_size individuals that have the least contributions are deleted from X one by one. The above process iterates until the maximum number of generations is achieved. The final X is output as the solution set to be used in the online optimization.

Algorithm 2 Evolutionary Set Search

Input:
Size of solution set: set_size
Size of sampled environmental values: env_size
Maximum number of generations: G_{max}

Set $G = 0$
Randomly generate env_size environmental values: $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$
Randomly generate set_size solutions as $P_0 = \{\boldsymbol{x}_{j,0} | j = 1, 2, ..., set\_size\}$
Set $X = P_0$
for $G = 0, 1, ..., G_{max} - 1$ **do**
  Apply evolutionary operators on P_G to generate an offspring population: O_G
  Set $U = P_G \cup O_G$
  Evaluate the contribution of each individual in U to the performance of U on $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$
  Set the contribution of each individual in U as its fitness
  Select set_size individuals from U as P_{G+1} according to individuals' fitness
  Set $X = X \cup O_G$
  for $i = 1, 2, ..., set\_size$ **do**
    Evaluate the contribution of each individual in X to the performance of X on $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$
    Delete the individual with the least contribution from X
  end for
end for

Output: X

B. Online Optimization

In the online optimization phase, the obtained solution set X is used as the initial population pop at the beginning and each time a problem change is detected. After initialization, solutions in the population pop are evaluated under the current environment. Considering that it might be impossible that every solution in X performs well under the current environment, individuals with the lowest performance are deleted after evaluation.

In the next step, local search operations are conducted on each of the remaining individuals in pop to solve the static problem under the current environment. This enables a fast re-adaptation and parallel execution of local search on individuals to satisfy real-time requirements of some real-world applications. In the field of EAs, local search is usually used in memetic algroithms/computing [14], [15] to exploit the neighborhood of an individual. During online optimization, the set X can be updated according to each newly encountered environmental change, e.g. by adding the best solution found for each environment to X.

In our framework, we suggest utilizing EAs for obtaining the solution set in the system design phase. The re-formulation of DOPs to static set-oriented optimization problems enables us to leverage the power of EAs to the largest extent, because the set-oriented optimization problem is much harder than its corresponding static version of the DOP of interest, and EAs typically show more advantages over traditional methods on set-oriented optimization, e.g., multi-objective optimization.

C. Summary

The SDO framework has the potential of faster adaptation to environmental changes compared to the existing diversity-driven methods, memory methods and prediction methods. On one hand, the efficiency of existing diversity-driven methods is low and hard to control as introducing randomly generated individuals can not guarantee obtaining good solutions except that the solution size is big enough. In contrast, SDO maintains a solution set that is obtained by considering possible environmental changes. As a consequence, given the same population size, SDO potentially includes more high-quality solutions in the solution than diversity-driven methods when online solving DOPs. Thus, SDO promises a faster adaptation to environmental change. On the other hand, the efficiency of memory methods and prediction methods highly depends on the solutions stored from previous search, which makes them suffer from cold-start problems when solving DOPs online. In contrast, the cold-start problem is avoided in the SDO framework due to the use of an offline evolutionary process.

IV. An Instantiation of SDO for DCOPs

The SDO framework given in Algorithm 1 has many details to be determined depending on specific problems. In this paper, we instantiated and tested the framework on DCOPs considering the following two points:

1) In comparison to unconstrained DOPs, DCOPs are of more practical relevance since most real-world problems are constrained. However, DCOPs are less investigated [7].
2) For real-world DCOPs, it is often more important to get a feasible solution quickly, which in many cases enlarge the "basin of attraction". In other words, it is more likely that the advantage of the general idea could be more visible on DCOPs, if there is any advantage.

The instantiated algorithm of SDO for DCOPs is called Set-based Dynamic Constrained Optimization (SDCO). In the following parts of this section, SDCO is presented to illustrate the detailed steps of a SDO algorithm.

Without loss of generality, the DCOPs considered in this paper have the following formulation:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{\alpha}(t)) \\ \text{s.t.} \quad & g_m(\boldsymbol{x}, \boldsymbol{\alpha}(t)) \leq 0, m = 1, 2, ..., k \\ \text{s.t.} \quad & h_n(\boldsymbol{x}, \boldsymbol{\alpha}(t)) = 0, n = 1, 2, ..., l \end{aligned} \tag{6}$$

where $\boldsymbol{x} = [x_0, x_1, ..., x_{D_x}]^T$ denotes the decision variable vector and D_x is the dimension of the search space, k and l are the number of inequality and equality constraints, t denotes the time, and $\boldsymbol{\alpha}(t)$ is the vector of environmental variables, which varies at a certain frequency as time t goes by. The variation range of each environmental variable in $\boldsymbol{\alpha}$ is assumed to be known beforehand.

A. Offline Set Search in SDCO

The offline set search in SDCO for DCOPs follows the framework in Algorithm 2. As the offline optimization targets to find a solution set rather than one single solution, EAs that can search different regions of the decision space are required. In this work, the NCS-C algorithm proposed in [16] is applied in SDCO. The NCS-C algorithm is an EA that explicitly promotes negatively correlated search behaviors among individuals so that they can search diverse regions of a search space.

Algorithm 3 outlines the pseudo-code of NCS-C based set search. After initializing a population of candidate solutions, NCS-C performs randomized local search on each individual $\boldsymbol{x}_{j,G}$ through Gaussian mutation operation to generate an offspring individual $\boldsymbol{x}'_{j,G}$, as given in Eq. (7):

$$x'_{d,j,G} = x'_{d,j,G} + N(0, \sigma_j^2) \tag{7}$$

where $x_{d,j,G}$ and $x'_{d,j,G}$ denote the d-th variable of $\boldsymbol{x}_{j,G}$ and $\boldsymbol{x}'_{j,G}$, respectively. σ_j is the Gaussian mutation stepsize and $N(0, \sigma_j^2)$ represents a random variable generated from a normal distribution with zero mean and standard deviation σ_j. After generating $\boldsymbol{x}'_{j,G}$, $\boldsymbol{x}_{j,G}$ and $\boldsymbol{x}'_{j,G}$ are compared according to their fitness and search behavior difference from the other individuals in the current population. The individual, which has the higher fitness and search difference, is preferred.

As NCS-C uses Gaussian mutation, a new solution generated based on $\boldsymbol{x}_{j,G}$ or $\boldsymbol{x}'_{j,G}$ follows a multivariate normal distribution. In Algorithm 3, p_j and p'_j are used to denote the corresponding probability distribution of $\boldsymbol{x}_{j,G}$ and $\boldsymbol{x}'_{j,G}$, respectively. The search behavior difference of $\boldsymbol{x}_{j,G}$ or $\boldsymbol{x}'_{j,G}$

Algorithm 3 NCS-C based Set Search in SDCO

Input:
Size of solution set: set_size
Size of sampled environmental values: env_size
Gaussian mutation stepsize: $\boldsymbol{\sigma} = \{\sigma_1, \sigma_2, ..., \sigma_{set\_size}\}$
Maximum number of generations: G_{max}
Set $G = 0$
Randomly generate env_size environmental values: $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$
Randomly generate set_size solutions as $P_0 = \{\boldsymbol{x}_{j,G} | j = 1, 2, ..., set\_size\}$
Set $X = P_0$
for $G = 0, 1, ..., G_{max} - 1$ **do**
  Set $\lambda_G = N(1, 0.1 - 0.1 * \frac{G}{G_{max}})$
  Set $O_G = \emptyset$
  for $j = 1, 2, ..., set\_size$ **do**
    Generate a new solution $\boldsymbol{x}'_{j,G}$ by applying Gaussian mutation operator with σ_j to $\boldsymbol{x}_{j,G}$
    Set $O_G = O_G \cup \boldsymbol{x}'_{j,G}$
    Compute $Corr(p_j)$ and $Corr(p'_j)$
  end for
  Set $U = P_G \cup O_G$
  Compute $fit(\boldsymbol{x}_{j,G})$ and $fit(\boldsymbol{x}'_{j,G})$ by evaluating the contribution of each individual in U to the performance of U on $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$
  Set $X = X \cup O_G$
  for $i = 1, 2, ..., set\_size$ **do**
    Evaluate the contribution of each individual in X to the performance of X on $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$
    Delete the individual with the least contribution from X
  end for
  for $j = 1, 2, ..., set\_size$ **do**
    Normalize $fit(\boldsymbol{x}_{j,G})$ and $fit(\boldsymbol{x}'_{j,G})$ to make $fit(\boldsymbol{x}_{j,G}) + fit(\boldsymbol{x}'_{j,G})$ equal to 1
    Normalize $Corr(p_j)$ and $Corr(p'_j)$ to make $Corr(p_j) + Corr(p'_j)$ euqal to 1
    if $\frac{fit(\boldsymbol{x}_{j,G})}{Corr(p'_j)} < \lambda_G$ **then**
      Update $\boldsymbol{x}_{j,G}$ with $\boldsymbol{x}'_{j,G}$
    end if
  end for
  if $\mod (G, epoch) = 0$ **then**
    for $j = 1, 2, ..., set\_size$ **do**
      Update σ_j according to the 1/5 successful rule
    end for
  end if
end for
Output: X

from the other individuals in the current population are calculated based on Bhattacharyya distance [17], which are denoted as $Corr(p_j)$ and $Corr(p'_j)$, and are computed as follows:

$$\begin{aligned} Corr(p_j) &= \min_k \{D_B(p_j, p_k) | k \neq j\} \\ Corr(p'_j) &= \min_k \{D_B(p'_j, p_k) | k \neq j\} \end{aligned} \tag{8}$$

where D_B is the Bhattacharyya distance between two probability distributions.

Since DCOPs are concerned, both the objective function and constraints need to be considered when evaluating fitness/contribution of individuals according to Eq. (5). In this work, the fitness/contribution of each individual $\boldsymbol{x}_j$ in a set U considering the sampled environments $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$ is calculated based on both the performance drop on the objective function and constraint violation after deleting $\boldsymbol{x}_j$ from U. For the problem defined in Eq. (6), the constraint violation of an individual under the environment $\boldsymbol{\alpha}_i$ is defined as follows:

$$c(\boldsymbol{x}_j, \boldsymbol{\alpha}_i) = \sum_{m=1}^{k} \max(g_m(\boldsymbol{x}_j, \boldsymbol{\alpha}_i), 0) + \sum_{n=1}^{l} |h_n(\boldsymbol{x}_j, \boldsymbol{\alpha}_i)| \tag{9}$$

The peformance drop on the objective function and the constraint violation considering the sampled environments $\{\boldsymbol{\alpha}_i | i = 1, 2, ..., env\_size\}$ when deleting $\boldsymbol{x}_j$ from U are calculated as follows:

$$\begin{aligned} f^{drop}(\boldsymbol{x}_j) &= \sum_{i=1}^{env\_size} (F(U, \boldsymbol{\alpha}_i) - F(U \backslash \boldsymbol{x}_j, \boldsymbol{\alpha}_i)) \\ \text{where } F(U, \boldsymbol{\alpha}_i) &= \max_{\boldsymbol{x} \in U} f(\boldsymbol{x}, \boldsymbol{\alpha}_i) \end{aligned} \tag{10}$$

$$\begin{aligned} c^{drop}(\boldsymbol{x}_j) &= \sum_{i=1}^{env\_size} (C(U \backslash \boldsymbol{x}_j, \boldsymbol{\alpha}_i) - C(U, \boldsymbol{\alpha}_i)) \\ \text{where } C(U, \boldsymbol{\alpha}_i) &= \min_{\boldsymbol{x} \in U} c(\boldsymbol{x}, \boldsymbol{\alpha}_i) \end{aligned} \tag{11}$$

Then, the fitness of $\boldsymbol{x}_j$ is calculated by a weighted sum of $f^{drop}(\boldsymbol{x}_j)$ and $c^{drop}(\boldsymbol{x}_j)$. The weights are computed by normalization. The following formula presents the fitness evaluation process:

$$fit(\boldsymbol{x}_j) = \frac{f^{drop}(\boldsymbol{x}_j)}{\sum_{j=1}^{set\_size} f^{drop}(\boldsymbol{x}_j)} + \frac{c^{drop}(\boldsymbol{x}_j)}{\sum_{j=1}^{set\_size} c^{drop}(\boldsymbol{x}_j)} \tag{12}$$

For every $epoch$ generations, the Gaussian mutation stepsize σ_j for each individual $\boldsymbol{x}_j$ is adapted according to the 1/5 success rule [18]. That is,

$$\sigma_j = \begin{cases} \frac{\sigma_j}{r}, & \text{if } \frac{c}{epoch} > 0.2 \\ \sigma_j * r, & \text{if } \frac{c}{epoch} < 0.2 \\ \sigma_j, & \text{if } \frac{c}{epoch} = 0.2 \end{cases} \tag{13}$$

where r is a parameter that is set to a value smaller than 1.0 and c is the number of times that $\boldsymbol{x}'_j$ replaces $\boldsymbol{x}_j$ in the last $epoch$ generations.

B. Online Optimization in SDCO

The online optimization in SDCO uses Gaussian mutation as the local search operation, whose step size is also adjusted according to the 1/5 success rule given in Eq. (13). Algorithm 4 details the process of online optimization in SDCO.

At the beginning, SDCO initializes a population pop with the solution set X obtained by Algorithm 3. After evaluating the individuals in pop, the worst del_size individuals are deleted from pop. Then, at each generation, for each individual in the population, a Gaussian mutation is conducted to generate a mutant individual. The mutant individual will replace the original individual if it is better under the current environment. When comparing two individuals, the simple feasibility rules proposed in [19] are used. One feasible individual is better than one infeasible individual. When comparing two feasible

Algorithm 4 Online Optimization in SDCO

Input:
The decision space for the solution: $\mathbb{R}^{D_x}$
The number of individuals to detect changes: $detect_k$
Solution set obtained from Algorithm 3: X
The number of individuals to delete from initial population: del_size

Set $pop = X$, $G = 0$
Evaluate pop with f and c under the current environment $\boldsymbol{\alpha}$
Delete the worst del_size individuals from pop
Initialize σ_j for each individual $\boldsymbol{x}_j$ in pop
Randomly generate $detect_k$ sentinel solutions and evaluate them
while stopping criteria is not satisfied **do**
  for each individual $\boldsymbol{x}_j$ in pop **do**
    Generate a new individual with Guassian mutation: $\boldsymbol{x}'_j$
    if $c(\boldsymbol{x}_j, \boldsymbol{\alpha}) \leq 0$ **then**
      if $c(\boldsymbol{x}'_j, \boldsymbol{\alpha}) \leq 0$ and $f(\boldsymbol{x}'_j) < f(\boldsymbol{x}_j)$ **then**
        Set $\boldsymbol{x}_j = \boldsymbol{x}'_j$
      end if
    else
      if $c(\boldsymbol{x}'_j) < c(\boldsymbol{x}_j)$ **then**
        Set $\boldsymbol{x}_j = \boldsymbol{x}'_j$
      end if
    end if
  end for
  Set $G = G + 1$
  if $\mod(G, epoch) = 0$ **then**
    for $j = 1, 2, ..., set\_size$ **do**
      Update σ_j according to the 1/5 successful rule
    end for
  end if
  Re-evaluate $detect_k$ sentinel solutions to detect changes
  if change is detected **then**
    Update the solution set X with pop
    Set $pop = X$, $G = 0$
    Evaluate pop with f and c under the current environment $\boldsymbol{\alpha}$
    Delete the worst del_size individuals from pop
    Re-initialize σ_j for each individual $\boldsymbol{x}_j$ in pop
  end if
end while

individuals, the individual with the higher fitness is better. When comparing two infeasible individuals, the individual with the lower constraint violation is better.

To detect problem changes, SDCO randomly generates $detect_k$ individuals and uses them as sentinels. SDCO re-evaluates these sentinels every generation to detect occuring changes. If a change happens, the population is re-initialized with X.

V. Experimental Study

In the present paper, we want to answer *the research question* whether the combination of offline set search and online optimization can achieve faster adaptation than existing dynamic optimization methods that only considers online optimization especially when facing fast-changing environments. In this section, we provide details on our experiments for solving DCOP test benchmarks and comparing the performance between SDCO and existing methods proposed for DCOPs. We also replaced the online optimization part of SDCO with the search strategies used in existing methods to validate the effectiveness of offline set search in SDCO. The following parts in this section will present experimental details and results.

A. Benchmark Problems

In the literature, there exist four suites of DCOP benchmark test functions. These include G24 [7], the moving feasible region benchmark (MFRB) [11], MPB-1 [20] and MPB-2 [21]. Among the four test sets, G24 only considers 2-D optimization problems. For MFRB, the optimal function value is not given for every test function. MPB-2 and MPB-1 were both extended from the moving peak benchmark (MPB) [22] but the details about MPB-1 given in [20] are very sparse. Hence, MPB-2 was used as the test benchmark in the experiments.

There are 6 basic test functions in MPB-2, which share the following form:

$$f(\boldsymbol{x}, t) = \max_{i=1,\ldots,10} \frac{H_i(t)}{1 + W_i(t) \sum_{j=1}^{D} (x_j - X_{i,j}(t))^2} \tag{14}$$

Every basic test function has 10 peaks. The feasible regions of each basic function are the circle areas around some peaks. The differences among the 6 basic test functions exists in the indexes of peaks that feasible regions are located in. Table I summarizes the information for the feasible regions in the 6 basic test function. In Table I, l denotes the number of feasible regions and a_k denotes the indexes of peaks that are in feasible regions.

TABLE I
Information for Feasible Regions of 6 Basic Test Functions [21]

$Func$	l	$a_k (k = 1, 2, ...l)$
1	1	$a_1 = 1$
2	1	a_1 is the index of the highest peak
3	2	$a_1 = 1$ and $a_2 = 6$
4	2	a_1 and a_2 are the indexes of the two highest peaks
5	3	$a_1 = 1$, $a_2 = 6$, and $a_3 = 10$
6	3	a_1, a_2, and a_3 are the indexes of the three highest peaks

All $X_{i,j}$, H_i and W_i constitute the environmental parameters $\boldsymbol{\alpha}$ for each basic function. They change according to the folllowing rules:

$$\begin{aligned} X_{i,j}(t) &= X_{i,j}(t-1) + \boldsymbol{v}_i(t) \\ \boldsymbol{v}_i(t) &= \frac{s}{|(r) + \boldsymbol{v}_i(t-1)|}((1-\lambda) * (r) + \lambda * \boldsymbol{v}_i(t-1)) \\ H_i(t) &= H_i(t-1) + Height\_severity * \sigma \\ W_i(t) &= W_i(t-1) + Weight\_severity * \sigma \end{aligned} \tag{15}$$

where $\boldsymbol{r}$ is a randomly generated vector and σ is a random number ranging between 0 to 1.0.

For each test function, there are 6 different change severities ($s = 1.0, 2.0, 3.0, 4.0, 5.0, 6.0$). Thus, we have 36 test functions in total. In the experimental study, the dimension of each test function was set to 10 and the decision space is $[0, 100]^{10}$. The other parameter settings are: $H \in [30, 70]$, $W \in [1, 12]$, $\lambda = 0$, $Height\_severity = 7.0$, and $Weight\_severity = 1.0$.

B. Compared Methods

In the literature, there exist three categories of methods proposed to solve DCOPs. These are diversity-driven methods (e.g. HyperM+GA [7], RIGA [7], DDECV [23], EBBPSO-T [24], DDECV+Repair[25], SELS [26], LTFR-DSPSO [11], DyCoDE [21]), memory methods (e.g. blending and censor [12], LTFR-DSPSO [11] and DyCoDE [21]), and prediction methods (e.g. IDEA-FPS [5] and LTFR-DSPSO [11]). In this paper, SDCO was compared with three state-of-the art methods among them: SELS, LTFR-DSPSO and DyCoDE.

The SELS method utilizes deterministic crowding [27] to make comparisons among similar individuals and assortative mating [28] to maintain good solutions in different feasible regions. At each generation, SELS applies a local search strategy, local evolutionary search enhancement by random memorizing (LESRM) [29], on the best solution to exploit its neighborhood for the current environment.

The LTFR-DSPSO method uses multiple subpopulations to locate and track multiple feasible regions. For each subpopulation, it applies a gradient-based repair method [30] to locate feasible solutions and adaptive local search (sequential quadratic programming (SQP) [31]) for exploitation. LTFR-DSPSO introduces a memory strategy to track previously feasible regions and a prediction method to predict the locations of future feasible regions.

DyCoDE [21]) first divides the whole population into multiple subpopulations, and then applies differential evolution (DE) [32] to perform local search in each subpopulation to find feasible solutions from different directions. When DyCoDE obtains enough feasible solutions, it will select some excellent individuals from each subpopulation to constitute a new population and then evolve the population to search for the optimal solution of the current environment. Once a change has been detected, DyCoDE re-initializes a new population based on memory individuals and some randomly generated individuals.

C. Performance Metrics

Two performance metrics are considered to make comparisons between the applied algorithms. First, we consider the modified offline error averaged at each function evaluation, which is defined as follows:

$$off\_err = \frac{1}{maxFES} \sum_{j=1}^{maxFES} (f^*(j) - f(j)) \qquad (16)$$

where $maxFES$ denotes the maximum number of the function evaluations, $f^*(j)$ denotes the maximum function value at j-th evaluation, and $f(j)$ denotes the function value of the best feasible solution found for the environment at j-th evaluation. If there are no feasible solutions at the j-th function evaluation, $f(j)$ is set to 0. The smaller the modified offline error is, the better the algorithm performs.

The second performance metric is the total number of function evaluations and constraint violation evaluations spent in obtaining a feasible solution averaged at each environment change, which is defined as follows:

$$eval\_num = \frac{1}{env\_num} \sum_{k=1}^{env\_num} (f\_num(k) + c\_num(k)) \qquad (17)$$

where env_num denotes the total number of environmental changes. $f\_num(k)$ and $c\_num(k)$ are the numbers of function evaluations and constraint violation evaluations spent before a feasible solution is obtained at the k-th environmental change, respectively. The smaller the number of evaluations needed to find a feasible solution, the better the algorithm performs.

D. Algorithm Comparison

1) Parameter Settings: When comparing SDCO with SELS, LTFR-DSPSO and DyCoDE, the number of environmental changes was set to 10, which is the same as in [21]. For each test function, two settings for the change frequency of each test function were considered: 2000 and 500 objective function evaluations (FEs). The parameters for SELS, LTFR-DSPSO and DyCoDE were set according to the values given in their original papers. The parameter settings for SDCO are given in Table II. When comparing SDCO with each other method, the size of the solution set X obtained offline by SDCO was set to the same size as the population size of the corresponding compared method. In detail, the size of the solution set was set to 24 when compared to SELS, and 45 when compared to LTFR-DSPSO and DyCoDE. The range for each environmental variable ($X_{i,j}(t)$, $H_i(t)$ or $W_i(t)$) was set to the interval between the minimum value and the maximum value that the variable can have when the number of problem changes is set to 10. Each method was run 30 times on each test function for statistical analysis. For the online optimization in SDCO, we have set different values for del_size to find a good setting, i.e. conducting local search on the remaining 1, 3, 5, 10 or 20 individuals of X. Experimental results show that SDCO performed best when del_size was set to (sol_size-1). Thus, del_size was set to $(sol\_size - 1)$ when comparing SDCO with SELS, LTFR-DSPSO and DyCoDE.

TABLE II
PARAMETER SETTINGS FOR SDCO

Parameter settings for offline set search			
G_{max}	3000	env_{size}	100
set_size	24 (when compared to SELS), 45 (when compared to LTFR-DSPSO and DyCoDE).		
$epoch$	10	r	0.95
initial σ_j	range of decision variable divided by set_size		
Parameter settings for Online optimization			
$epoch$	5	r	0.5
initial σ_j	Euclidean distance of two closest solutions in X		
del_size	$(sol\_size - 1)$, $(sol\_size - 3)$, $(sol\_size - 5)$, $(sol\_size - 10)$, $(sol\_size - 20)$		

2) Experimental Results under Change Frequency of 2000 FEs: Tables III and IV summarize the mean and standard deviation of the modified offline error (off_err) and the number of evaluations needed to find a feasible solution

TABLE III
COMPARISON BETWEEN LTFR-DSPSO, DYCODE AND SDCO WITH WITH THE OFFLINE SOLUTION SOLUTION SET SIZE sol_size SET TO 45. "MEAN" AND "STD" DENOTE THE MEAN AND STANDARD DEVIATION OF off_err OR $eval\_num$ OVER 30 RUNS. "−" MEANS THAT SDCO PERFORMED SIGNIFICANTLY BETTER THAN THE COMPARED ALGORITHM IN THAT COLUMN ON THE TEST FUNCTION IN THAT ROW.

Func	off_err (Mean±Std)			$eval\_num$ (Mean±Std)		
	SDCO	LTFR-DSPSO	DyCoDE	SDCO	LTFR-DSPSO	DyCoDE
$DCOP1-s1$	2.07e+00±1.37e-01	4.36e+01±4.52e+00 −	1.90e+01±4.08e-01 −	1.00e+00±0.00e+00	1.97e+02±2.87e+01 −	2.59e+02±2.55e+01 −
$DCOP1-s2$	4.78e+00±2.05e-01	5.61e+01±1.89e+00 −	3.02e+01±9.54e-01 −	1.00e+00±0.00e+00	2.45e+02±6.50e+01 −	2.47e+02±1.88e+01 −
$DCOP1-s3$	7.00e+00±3.23e-01	5.22e+01±3.40e+00 −	3.29e+01±9.54e-01 −	1.00e+00±0.00e+00	2.17e+02±6.02e+01 −	2.54e+02±2.64e+01 −
$DCOP1-s4$	7.43e+00±3.40e-01	5.80e+01±2.19e+00 −	3.41e+01±8.65e-01 −	1.00e+00±0.00e+00	2.00e+02±5.49e+01 −	2.57e+02±2.47e+01 −
$DCOP1-s5$	7.49e+00±3.41e-01	5.21e+01±2.47e+00 −	3.18e+01±1.24e+00 −	1.00e+00±0.00e+00	2.06e+02±4.03e+01 −	2.78e+02±2.64e+01 −
$DCOP1-s6$	7.37e+00±2.63e-01	4.95e+01±2.58e+00 −	3.05e+01±1.13e+00 −	1.00e+00±0.00e+00	2.28e+02±5.80e+01 −	2.60e+02±2.44e+01 −
$DCOP2-s1$	3.26e+00±1.60e-01	4.24e+01±2.87e+00 −	4.48e+01±7.46e-01 −	7.60e+00±3.61e-15	4.07e+02±5.19e+01 −	1.52e+03±5.17e+01 −
$DCOP2-s2$	5.80e+00±3.14e-01	5.43e+01±5.16e+00 −	4.53e+01±8.06e-01 −	3.40e+00±1.36e-15	3.39e+02±4.88e+01 −	1.28e+03±5.14e+01 −
$DCOP2-s3$	8.29e+00±3.25e-01	5.09e+01±4.30e+00 −	5.36e+01±7.43e-01 −	3.90e+00±1.81e-15	4.54e+02±6.30e+01 −	1.80e+03±5.45e+01 −
$DCOP2-s4$	8.97e+00±3.27e-01	5.49e+01±4.20e+00 −	4.76e+01±8.68e-01 −	4.30e+00±1.81e-15	3.60e+02±8.63e+01 −	1.26e+03±5.78e+01 −
$DCOP2-s5$	8.98e+00±4.96e-01	5.41e+01±3.51e+00 −	4.83e+01±1.22e+00 −	4.10e+00±1.81e-15	3.61e+02±5.88e+01 −	1.32e+03±5.07e+01 −
$DCOP2-s6$	1.02e+01±3.91e-01	5.15e+01±4.88e+00 −	4.54e+01±7.33e-01 −	3.20e+00±1.36e-15	3.24e+02±3.49e+01 −	1.03e+03±3.75e+01 −
$DCOP3-s1$	2.03e+01±1.58e-01	3.72e+01±4.17e+00 −	3.16e+01±9.09e+00 −	1.00e+00±0.00e+00	1.84e+02±4.11e+01 −	2.84e+02±3.25e+01 −
$DCOP3-s2$	5.17e+00±2.45e-01	4.75e+01±3.59e+00 −	4.82e+01±6.07e+00 −	1.00e+00±0.00e+00	2.68e+02±9.83e+01 −	2.68e+02±3.32e+01 −
$DCOP3-s3$	7.98e+00±3.08e-01	4.30e+01±4.19e+00 −	3.91e+01±7.06e+00 −	1.00e+00±0.00e+00	2.21e+02±5.20e+01 −	2.73e+02±3.65e+01 −
$DCOP3-s4$	2.00e+01±2.01e-01	4.73e+01±4.29e+00 −	4.92e+01±4.91e+00 −	1.00e+00±0.00e+00	2.21e+02±5.34e+01 −	2.75e+02±4.54e+01 −
$DCOP3-s5$	1.89e+01±2.37e-01	4.27e+01±3.91e+00 −	4.68e+01±4.38e+00 −	1.10e+00±4.52e-16	2.19e+02±5.43e+01 −	2.79e+02±3.42e+01 −
$DCOP3-s6$	1.41e+01±2.57e-01	3.74e+01±3.58e+00 −	3.40e+01±3.85e+00 −	1.00e+00±0.00e+00	2.01e+02±4.93e+01 −	2.57e+02±2.79e+01 −
$DCOP4-s1$	4.01e+00±1.50e-01	2.92e+01±3.84e+00 −	4.14e+01±2.60e+00 −	2.40e+00±0.00e+00	3.01e+02±6.51e+01 −	1.06e+03±1.45e+02 −
$DCOP4-s2$	8.08e+00±2.35e-01	4.81e+01±3.89e+00 −	4.99e+01±3.27e+00 −	1.80e+00±9.03e-16	2.80e+02±9.61e+01 −	8.18e+02±1.85e+02 −
$DCOP4-s3$	8.90e+00±3.57e-01	3.77e+01±5.83e+00 −	5.03e+01±2.59e+00 −	4.30e+00±1.81e-15	2.79e+02±5.00e+01 −	9.95e+02±1.72e+02 −
$DCOP4-s4$	9.09e+00±3.27e-01	3.97e+01±3.63e+00 −	4.88e+01±2.72e+00 −	2.90e+00±4.52e-16	2.19e+02±6.76e+01 −	1.02e+03±7.00e+01 −
$DCOP4-s5$	9.41e+00±3.80e-01	3.87e+01±3.81e+00 −	5.45e+01±2.44e+00 −	2.50e+00±0.00e+00	2.41e+02±6.39e+01 −	9.72e+02±1.73e+02 −
$DCOP4-s6$	1.07e+01±4.92e-01	4.11e+01±4.70e+00 −	5.60e+01±3.39e+00 −	1.70e+00±6.78e-16	2.96e+02±3.79e+01 −	9.01e+02±1.45e+02 −
$DCOP5-s1$	3.07e+00±1.13e-01	3.61e+01±3.39e+00 −	4.42e+01±6.10e+00 −	1.00e+00±0.00e+00	1.96e+02±5.57e+01 −	2.82e+02±3.08e+01 −
$DCOP5-s2$	4.98e+00±3.07e-01	4.66e+01±4.33e+00 −	4.84e+01±7.40e+00 −	1.00e+00±0.00e+00	2.18e+02±5.30e+01 −	2.72e+02±3.24e+01 −
$DCOP5-s3$	1.64e+01±2.58e-01	4.59e+01±3.43e+00 −	4.76e+01±9.04e+00 −	1.00e+00±0.00e+00	2.02e+02±4.54e+01 −	2.76e+02±2.55e+01 −
$DCOP5-s4$	2.67e+01±1.68e-01	4.13e+01±4.11e+00 −	5.61e+01±2.38e+00 −	1.00e+00±0.00e+00	2.35e+02±7.72e+01 −	2.89e+02±3.06e+01 −
$DCOP5-s5$	2.67e+01±2.86e-01	4.35e+01±5.68e+00 −	5.77e+01±3.23e+00 −	1.00e+00±0.00e+00	2.25e+02±5.42e+01 −	2.96e+02±3.01e+01 −
$DCOP5-s6$	1.78e+01±2.99e-01	4.08e+01±3.61e+00 −	4.35e+01±7.70e+00 −	1.00e+00±0.00e+00	2.20e+02±5.32e+01 −	2.63e+02±2.44e+01 −
$DCOP6-s1$	3.80e+00±8.60e-02	2.67e+01±4.18e+00 −	4.14e+01±5.84e+00 −	1.50e+00±0.00e+00	2.41e+02±8.25e+01 −	6.48e+02±1.83e+02 −
$DCOP6-s2$	8.25e+00±2.26e-01	4.46e+01±4.93e+00 −	5.41e+01±3.63e+00 −	1.40e+00±6.78e-16	2.62e+02±6.94e+01 −	7.30e+02±2.34e+02 −
$DCOP6-s3$	1.21e+01±2.63e-01	3.68e+01±4.87e+00 −	5.08e+01±3.90e+00 −	4.60e+00±2.71e-15	2.18e+02±8.74e+01 −	6.45e+02±1.82e+02 −
$DCOP6-s4$	9.32e+00±3.78e-01	3.79e+01±3.70e+00 −	4.81e+01±4.28e+00 −	2.70e+00±1.36e-15	2.51e+02±8.17e+01 −	4.78e+02±6.88e+01 −
$DCOP6-s5$	9.19e+00±4.19e-01	3.66e+01±3.28e+00 −	5.73e+01±3.48e+00 −	5.90e+00±2.71e-15	2.26e+02±6.40e+01 −	1.01e+03±2.54e+02 −
$DCOP6-s6$	1.24e+01±4.53e-01	3.89e+01±4.53e+00 −	5.85e+01±3.63e+00 −	4.24e+01±6.16e+00	2.81e+02±6.72e+01 −	8.09e+02±1.53e+02 −

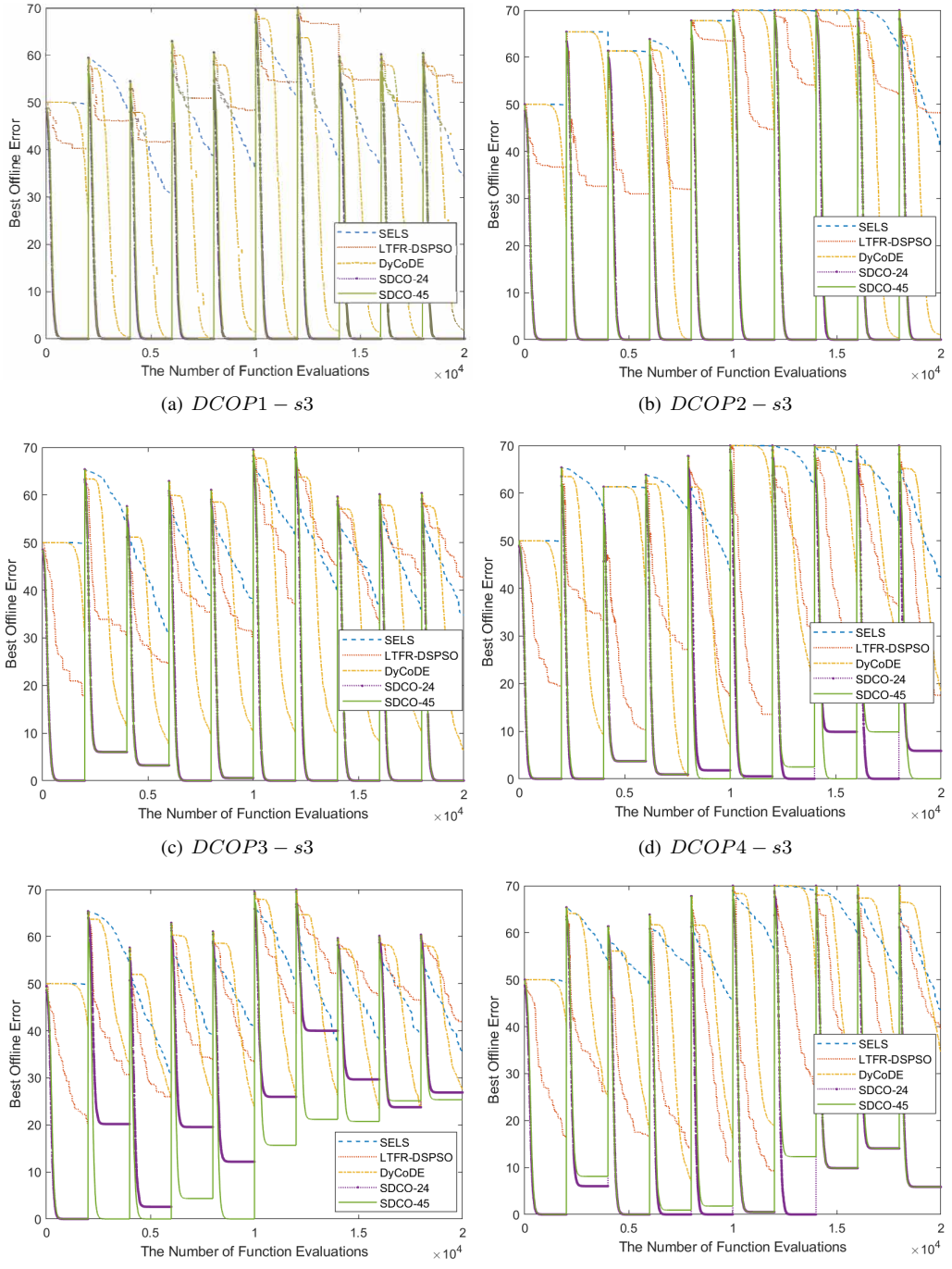
TABLE IV
COMPARISON BETWEEN SELS AND SDCO WITH THE OFFLINE SOLUTION SOLUTION SET SIZE sol_size SET TO 24. "MEAN" AND "STD" DENOTE THE MEAN AND STANDARD DEVIATION OF off_err OR $eval\_num$ OVER 30 RUNS. "−" MEANS THAT SDCO PERFORMED SIGNIFICANTLY BETTER THAN SELS ON THE TEST FUNCTION IN THAT ROW.

Func	off_err (Mean±Std)		$eval\_num$ (Mean±Std)	
	SDCO	SELS	SDCO	SELS
$DCOP1-s1$	1.95e+00±1.45e-01	2.47e+01±1.90e+00 −	3.00e+00±0.00e+00	5.31e+02±9.01e+01 −
$DCOP1-s2$	4.63e+00±2.54e-01	4.30e+01±1.23e+00 −	3.00e+00±0.00e+00	4.31e+02±8.12e+01 −
$DCOP1-s3$	6.52e+00±3.83e-01	4.83e+01±9.83e-01 −	3.00e+00±0.00e+00	4.22e+02±1.08e+02 −
$DCOP1-s4$	7.03e+00±4.08e-01	4.98e+01±9.90e-01 −	3.00e+00±0.00e+00	4.60e+02±9.78e+01 −
$DCOP1-s5$	6.86e+00±3.77e-01	4.83e+01±1.14e+00 −	3.00e+00±0.00e+00	6.35e+02±1.09e+02 −
$DCOP1-s6$	7.15e+00±4.29e-01	4.49e+01±1.05e+00 −	3.00e+00±0.00e+00	5.43e+02±9.85e+01 −
$DCOP2-s1$	4.72e+00±2.48e-01	6.14e+01±1.24e+00 −	8.41e+01±1.10e+01	2.84e+03±1.89e+02 −
$DCOP2-s2$	5.15e+00±2.54e-01	6.09e+01±1.29e+00 −	1.10e+01±0.00e+00	2.55e+03±2.17e+02 −
$DCOP2-s3$	8.00e+00±4.15e-01	6.36e+01±9.12e-01 −	1.42e+01±7.23e-15	2.98e+03±1.46e+02 −
$DCOP2-s4$	9.09e+00±4.10e-01	6.20e+01±7.34e-01 −	1.74e+01±7.23e-15	2.75e+03±1.65e+02 −
$DCOP2-s5$	8.59e+00±3.39e-01	6.33e+01±7.81e-01 −	9.20e+00±5.42e-15	2.68e+03±2.37e+02 −
$DCOP2-s6$	9.85e+00±5.67e-01	6.08e+01±1.02e+00 −	9.20e+00±5.42e-15	2.09e+03±1.59e+02 −
$DCOP3-s1$	2.80e+00±1.33e-01	3.34e+01±1.02e+01 −	3.00e+00±0.00e+00	5.44e+02±8.41e+01 −
$DCOP3-s2$	2.59e+01±2.10e-01	4.60e+01±4.08e+00 −	3.00e+00±0.00e+00	4.56e+02±7.16e+01 −
$DCOP3-s3$	7.83e+00±3.41e-01	4.93e+01±1.22e+00 −	3.00e+00±0.00e+00	4.42e+02±9.29e+01 −
$DCOP3-s4$	1.96e+01±1.79e-01	4.81e+01±2.42e+00 −	3.00e+00±0.00e+00	4.46e+02±8.81e+01 −
$DCOP3-s5$	1.89e+01±3.01e-01	4.89e+01±1.21e+00 −	3.20e+00±1.36e-15	5.45e+02±7.77e+01 −
$DCOP3-s6$	1.35e+01±2.47e-01	4.23e+01±3.43e+00 −	3.00e+00±0.00e+00	3.88e+02±1.08e+02 −
$DCOP4-s1$	3.70e+00±1.53e-01	5.14e+01±3.29e+00 −	5.40e+00±2.71e-15	1.91e+03±2.98e+02 −
$DCOP4-s2$	7.59e+00±2.43e-01	5.46e+01±5.20e+00 −	4.60e+00±2.71e-15	1.96e+03±4.09e+02 −
$DCOP4-s3$	9.01e+00±3.21e-01	6.11e+01±1.90e+00 −	9.60e+00±0.00e+00	2.01e+03±3.42e+02 −
$DCOP4-s4$	8.70e+00±3.84e-01	5.86e+01±1.56e+00 −	4.80e+00±0.00e+00	2.02e+03±2.24e+02 −
$DCOP4-s5$	8.44e+00±3.26e-01	5.93e+01±1.98e+00 −	1.04e+01±1.81e-15	1.80e+03±4.47e+02 −
$DCOP4-s6$	1.25e+01±3.54e-01	6.08e+01±1.55e+00 −	7.70e+01±1.28e+01	1.86e+03±4.37e+02 −
$DCOP5-s1$	1.99e+01±1.03e-01	3.98e+01±8.70e+00 −	3.00e+00±0.00e+00	4.44e+02±8.97e+01 −
$DCOP5-s2$	8.58e+00±1.94e-01	4.86e+01±4.80e+00 −	3.00e+00±0.00e+00	4.33e+02±7.39e+01 −
$DCOP5-s3$	2.45e+01±1.96e-01	4.99e+01±1.58e+00 −	3.00e+00±0.00e+00	4.53e+02±8.74e+01 −
$DCOP5-s4$	7.21e+00±3.47e-01	5.00e+01±3.26e+00 −	3.00e+00±0.00e+00	4.03e+02±9.09e+01 −
$DCOP5-s5$	2.60e+01±2.73e-01	5.45e+01±2.37e+00 −	3.00e+00±0.00e+00	4.99e+02±8.96e+01 −
$DCOP5-s6$	1.23e+01±4.03e-01	4.63e+01±3.80e+00 −	3.20e+00±1.36e-15	4.21e+02±6.59e+01 −
$DCOP6-s1$	3.68e+00±1.45e-01	4.18e+01±4.29e+00 −	4.20e+00±1.81e-15	1.11e+03±2.77e+02 −
$DCOP6-s2$	6.15e+00±2.92e-01	5.63e+01±6.70e+00 −	6.76e+01±9.89e+00	1.46e+03±4.77e+02 −
$DCOP6-s3$	1.02e+01±3.42e-01	5.88e+01±3.10e+00 −	1.12e+01±5.42e-15	1.35e+03±5.22e+02 −
$DCOP6-s4$	8.66e+00±4.24e-01	5.43e+01±1.46e+00 −	1.08e+01±5.42e-15	8.81e+02±2.13e+02 −
$DCOP6-s5$	8.76e+00±3.28e-01	5.82e+01±1.79e+00 −	3.40e+00±1.36e-15	1.60e+03±5.06e+02 −
$DCOP6-s6$	1.24e+01±4.12e-01	5.90e+01±1.24e+00 −	7.40e+01±1.47e+01	1.35e+03±3.09e+02 −

($eval\_num$) over 30 runs for SELS, LTFR-DSPSO and DyCoDE on each test function. In the first column of both tables, $DCOPi-sj(i,j=1,2,...,6)$ means the i-th test function with severity s equal to j. When comparing two different methods under one performance metric, the Wilcoxon rank-sum test with a confidence level at 0.05 was used. The symbol "−" means that SDCO performed significantly better than the compared algorithm in that column on the test function in that row. It can be seen that SDCO performed best on each test function when compared to SELS, LTFR-DSPSO and DyCoDE. With the help of an offline set search process, SDCO does not only need less evaluations to achieve a feasible solution but also outputs a solution of higher quality at the same cost of function evaluations when solving DCOPs online.

Fig. 2 illustrates the evolutionary curves for SELS, LTFR-DSPSO, DyCoDE and SDCO on 6 representative test functions: $DCOP1-s3$, $DCOP2-s3$, $DCOP3-s3$, $DCOP4-s3$, $DCOP5-s3$ and $DCOP6-s3$. The X-axis denotes the number of function evaluations and the Y-axis denotes the offline error of the best solution found for the current environment averaged over 30 runs. In Fig. 2, SDCO-24 and SDCO-45 represent the SDCO method with sol_size and sol_size set to 24 and 45, respectively. It can be seen that the evolutionary curves of SDCO drop very quickly each time the environment changes. In contrast, the evolutionary curves of SELS, LTFR-DSPSO and DyCoDE fall slower compared to those of SDCO. Moreover, SDCO achieved faster convergence than SELS, LTFR-DSPSO and DyCoDE. This demonstrates that an algorithm with a faster adaptation to environmental changes has been achieved for DCOPs by combining offline computation and online optimization.

3) Experimental Results under Change Frequency of 500 FEs: Fig. 3 illustrates the evolutionary curves for SELS, LTFR-DSPSO, DyCoDE and SDCO on 6 representative test functions: $DCOP1-s3$, $DCOP2-s3$, $DCOP3-s3$, $DCOP4-s3$, $DCOP5-s3$ and $DCOP6-s3$ under the change frequency of 500 FEs. The X-axis denotes the number

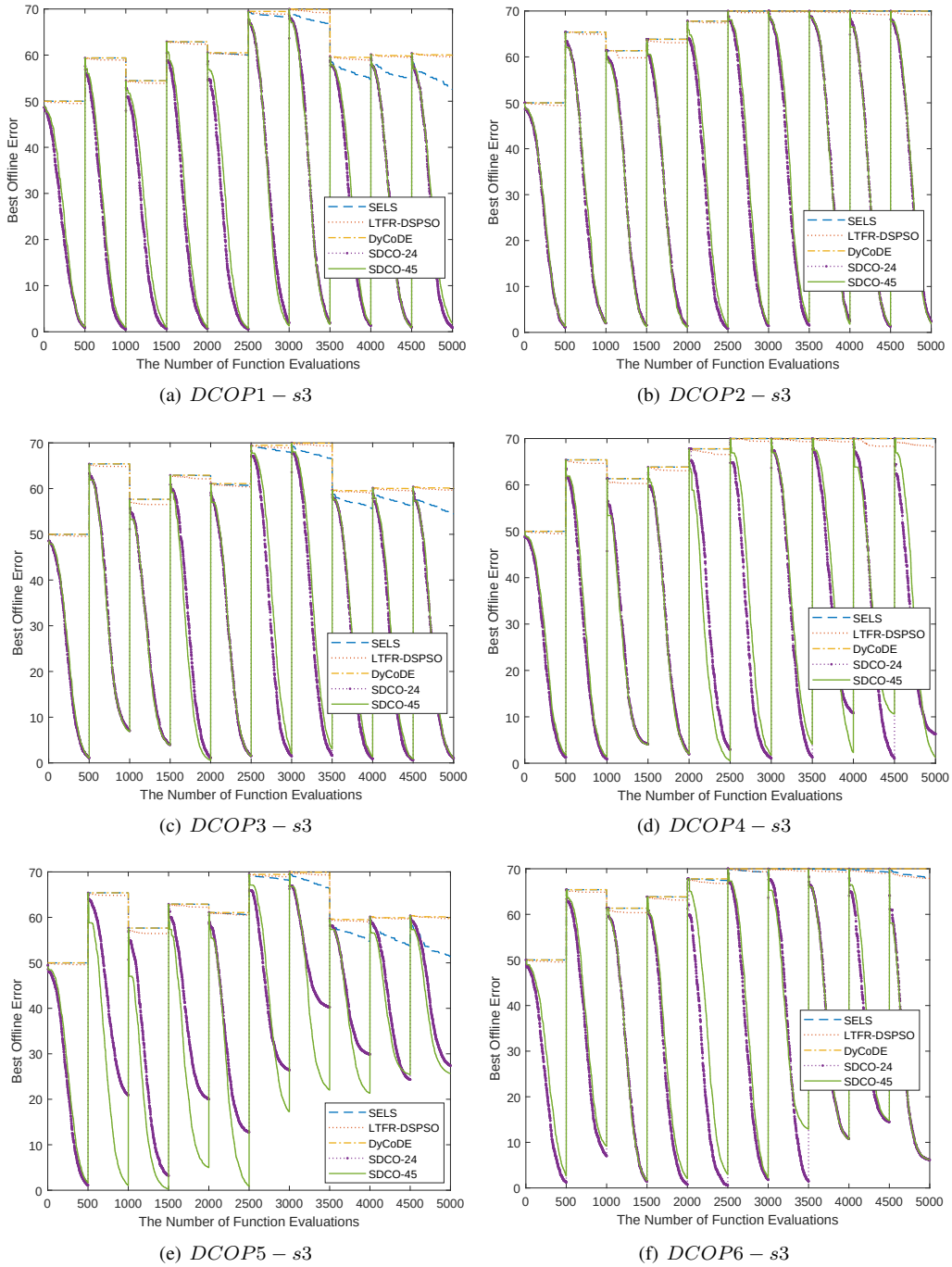


(a) $DCOP1 - s3$ (b) $DCOP2 - s3$

(c) $DCOP3 - s3$ (d) $DCOP4 - s3$

(e) $DCOP5 - s3$ (f) $DCOP6 - s3$

Fig. 2. Evolutionary Curves of SELS, LTFR-DSPSO, DyCoDE and SDCO on 6 representative functions with the change frequency of 2000 function evaluations. The X-axis denotes the number of function evaluations and the Y-axis denotes the offline error of the best solution found for the current environment averaged over 30 runs.



(a) $DCOP1 - s3$ (b) $DCOP2 - s3$

(c) $DCOP3 - s3$ (d) $DCOP4 - s3$

(e) $DCOP5 - s3$ (f) $DCOP6 - s3$

Fig. 3. Evolutionary Curves of SELS, LTFR-DSPSO, DyCoDE and SDCO on 6 representative functions with the change frequency of 500 function evaluations. The X-axis denotes the number of function evaluations and the Y-axis denotes the offline error of the best solution found for the current environment averaged over 30 runs.

of function evaluations and the Y-axis denotes the offline error of the best solution found for the current environment averaged over 30 runs. When comparing Figs. 2 and 3, it can be seen that SELS, LTFR-DSPSO and DyCoDE fail work properly when the problem changes 4 times faster. However, SDCO still shows fast adaptation in this situation. These results demonstrate the advantage of SDCO over SELS, LTFR-DSPSO and DyCoDE in fast-changing environments.

E. Effectiveness of Offline Set Search in SDCO

SDCO uses the Gaussian mutation with 1/5 success rule as the search strategy in the online optimization. This is different from the search strategies of SELS, LTFR-DSPSO and DyCoDE. To validate the effectiveness of the offline set search in SDCO, the local search strategy in SDCO was replaced by the local search strategies applied in SELS, LTFR-DSPSO and DyCoDE. When compared to SELS, SDCO used LESRM as the online search strategy. When compared to LTFR-DSPSO, SDCO used the gradient-based repair method first and then

SQP to do local search on each individual in the population. When compared to DyCoDE, SDCO divided the population into subpopulations and then applied differential evolution (DE) to perform local search in each subpopulation. The resultant methods are called SDCO+lesrm, SDCO+sqp, and SDCO+dycode. For each of SDCO+lesrm, SDCO+sqp, and SDCO+dycode, local search is conducted on each individual in the solution set X each time the environment changes. This means the population sizes of SDCO+lesrm, SDCO+sqp, and SDCO+dycode were kept the same as those of SELS, LTFR-DSPSO, and DyCoDE, respectively, so as to eliminate the effect of population size in the comparison.

Tables V, VI and VII summarize the mean and standard deviation of the offline error (off_err) and the number of evaluations needed to find a feasible solution ($eval\_num$) over 30 runs for SELS, SDCO+lersm, LTFR-DSPSO, SDCO+sqp, DyCoDE, and SDCO+dycode on the 36 test functions. The symbols "$-$", "$+$" and "a" mean that SDCO performed significantly better than, worse than, and similar to the compared algorithm in that column on the test function in that row.

TABLE V
COMPARISON BETWEEN SELS AND SDCO+LESRM. "MEAN" AND "STD" DENOTE THE MEAN AND STANDARD DEVIATION OF off_err OR $eval\_num$ OVER 30 RUNS. "$-$", "$+$" AND "a" MEAN THAT SDCO+LESRM PERFORMED SIGNIFICANTLY BETTER THAN, WORSE THAN, AND SIMILAR TO SELS ON THE TEST FUNCTION IN THAT ROW.

Func	off_err (Mean±Std)		$eval\_num$ (Mean±Std)	
	SDCO+lesrm	SELS	SDCO+lesrm	SELS
$DCOP1-s1$	1.05e+01±8.36e-01	2.47e+01±1.90e+00 −	2.20e+00±9.03e-16	5.31e+02±9.01e+01 −
$DCOP1-s2$	2.95e+01±1.83e+00	4.30e+01±1.23e+00 −	2.20e+00±9.03e-16	4.30e+02±8.12e+01 −
$DCOP1-s3$	4.19e+01±1.55e+00	4.83e+01±9.83e-01 −	2.20e+00±9.03e-16	4.21e+02±1.08e+02 −
$DCOP1-s4$	4.60e+01±1.44e+00	4.98e+01±9.90e-01 −	2.20e+00±9.03e-16	4.60e+02±9.78e+01 −
$DCOP1-s5$	4.40e+01±1.44e+00	4.83e+01±1.14e+00 −	2.20e+00±9.03e-16	6.35e+02±1.09e+02 −
$DCOP1-s6$	4.45e+01±9.72e-01	4.49e+01±1.05e+00 a	2.20e+00±9.03e-16	5.43e+02±9.85e+01 −
$DCOP2-s1$	2.28e+01±1.88e+00	6.14e+01±1.24e+00 −	4.14e+02±2.31e-13	2.73e+03±1.39e+02 −
$DCOP2-s2$	3.71e+01±2.23e+00	6.09e+01±1.29e+00 −	2.06e+01±1.08e-14	2.55e+03±2.17e+02 −
$DCOP2-s3$	5.49e+01±1.63e+00	6.36e+01±9.12e-01 −	2.12e+01±3.61e-15	2.98e+03±1.46e+02 −
$DCOP2-s4$	5.79e+01±1.30e+00	6.20e+01±7.34e-01 −	3.12e+01±1.45e-14	2.35e+03±1.65e+02 −
$DCOP2-s5$	5.60e+01±1.41e+00	6.33e+01±7.81e-01 −	1.68e+01±7.23e-15	2.68e+03±2.37e+02 −
$DCOP2-s6$	6.00e+01±6.25e-01	6.08e+01±1.02e+00 −	1.84e+01±1.08e-14	2.09e+03±1.59e+02 −
$DCOP3-s1$	1.23e+01±8.98e-01	3.34e+01±1.02e+01 −	2.20e+00±9.03e-16	5.44e+02±8.41e+01 −
$DCOP3-s2$	4.31e+01±8.24e-01	4.60e+01±4.08e+00 −	2.20e+00±9.03e-16	4.55e+02±7.16e+01 −
$DCOP3-s3$	4.39e+01±1.52e+00	4.93e+01±1.22e+00 −	2.20e+00±9.03e-16	4.41e+02±9.29e+01 −
$DCOP3-s4$	4.21e+01±1.49e+00	4.81e+01±2.42e+00 −	2.20e+00±9.03e-16	4.46e+02±8.81e+01 −
$DCOP3-s5$	4.77e+01±1.37e+00	4.89e+01±1.21e+00 −	2.20e+00±9.03e-16	5.45e+02±7.77e+01 −
$DCOP3-s6$	4.05e+01±9.85e-01	4.23e+01±3.43e+00 a	2.45e+00±5.65e-01	3.88e+02±1.08e+02 −
$DCOP4-s1$	1.42e+01±1.15e+00	5.14e+01±3.29e+00 −	8.41e+00±7.84e-01	1.90e+03±2.84e+02 −
$DCOP4-s2$	3.53e+01±1.41e+00	5.46e+01±5.20e+00 −	1.00e+01±6.98e-01	1.64e+03±3.96e+02 −
$DCOP4-s3$	4.88e+01±2.18e+00	6.11e+01±1.90e+00 −	1.65e+01±7.98e-01	1.98e+03±2.85e+02 −
$DCOP4-s4$	5.30e+01±1.01e+00	5.86e+01±1.56e+00 −	1.20e+01±0.00e+00	1.65e+03±2.30e+02 −
$DCOP4-s5$	5.27e+01±1.65e+00	5.93e+01±1.98e+00 −	1.81e+01±8.12e-01	1.80e+03±4.47e+02 −
$DCOP4-s6$	5.97e+01±9.36e-01	6.08e+01±1.55e+00 −	4.14e+02±6.71e-01	1.86e+03±4.37e+02 −
$DCOP5-s1$	3.00e+01±1.11e+00	3.98e+01±8.70e+00 −	2.20e+00±9.03e-16	4.43e+02±8.97e+01 −
$DCOP5-s2$	3.48e+01±1.79e+00	4.86e+01±4.80e+00 −	2.20e+00±9.03e-16	4.33e+02±7.39e+01 −
$DCOP5-s3$	4.84e+01±1.27e+00	4.99e+01±1.58e+00 −	2.20e+00±9.03e-16	4.53e+02±8.74e+01 −
$DCOP5-s4$	4.58e+01±1.29e+00	5.00e+01±3.26e+00 −	2.20e+00±9.03e-16	4.03e+02±9.09e+01 −
$DCOP5-s5$	5.49e+01±9.34e-01	5.45e+01±2.37e+00 a	2.20e+00±9.03e-16	4.99e+02±8.96e+01 −
$DCOP5-s6$	5.38e+01±1.16e+00	4.63e+01±3.80e+00 +	2.40e+00±0.00e+00	4.21e+02±6.59e+01 −
$DCOP6-s1$	1.35e+01±1.04e+00	4.18e+01±4.29e+00 −	4.32e+00±3.99e-01	1.11e+03±2.77e+02 −
$DCOP6-s2$	3.25e+01±1.61e+00	5.63e+01±6.70e+00 −	4.02e+02±1.55e+01	1.42e+03±4.38e+02 −
$DCOP6-s3$	4.84e+01±1.70e+00	5.88e+01±3.10e+00 −	1.70e+01±0.00e+00	1.35e+03±5.22e+02 −
$DCOP6-s4$	4.86e+01±1.75e+00	5.43e+01±1.46e+00 −	2.15e+01±2.27e-01	8.81e+02±2.13e+02 −
$DCOP6-s5$	5.21e+01±1.56e+00	5.82e+01±1.79e+00 −	7.48e+00±7.46e-01	1.60e+03±5.06e+02 −
$DCOP6-s6$	5.90e+01±8.48e-01	5.90e+01±1.24e+00 a	4.12e+02±4.06e-01	1.35e+03±3.09e+02 −

It can be seen from Table V that SDCO+lesrm outperformed SELS on almost all test functions. From Tabel VI, we can conclude that SDCO+sqp generally performed better than LTFR-DSPSO. When comparing them according to off_err, SDCO+sqp performed better on 20 test functions, worse on 11 test functions, and indistinctively on 5 test functions. When using $eval\_num$ as performance measure, SDCO+sqp needed less evaluations to obtain a feasible solution than LTFR-DSPSO on all test functions. When compared to DyCoDE, SDCO+dycode showed superior performance according to Table VII. One reason why SDCO methods performed worse on

TABLE VI
COMPARISON BETWEEN LTFR-DSPSO AND SDCO+SQP. "MEAN" AND "STD" DENOTE THE MEAN AND STANDARD DEVIATION OF off_err OR $eval\_num$ OVER 30 RUNS. "$-$", "$+$" AND "a" MEAN THAT SDCO+SQP PERFORMED SIGNIFICANTLY BETTER THAN, WORSE THAN, AND SIMILAR TO LTFR-DSPSO ON THE TEST FUNCTION IN THAT ROW.

Func	off_err (Mean±Std)		$eval\_num$ (Mean±Std)	
	SDCO+sqp	LTFR-DSPSO	SDCO+sqp	LTFR-DSPSO
$DCOP1-s1$	1.35e+01±0.00e+00	4.36e+01±4.52e+00 −	8.20e+00±3.61e-15	1.97e+02±2.87e+01 −
$DCOP1-s2$	4.48e+01±7.23e-15	5.61e+01±1.89e+00 −	1.32e+01±7.23e-15	2.45e+02±6.50e+01 −
$DCOP1-s3$	5.16e+01±2.17e-14	5.22e+01±3.40e+00 a	1.22e+01±5.42e-15	2.17e+02±6.02e+01 −
$DCOP1-s4$	5.09e+01±2.17e-14	5.80e+01±2.19e+00 −	2.28e+01±3.61e-15	2.00e+02±5.49e+01 −
$DCOP1-s5$	4.74e+01±7.23e-15	5.21e+01±2.47e+00 −	1.02e+01±5.42e-15	2.06e+02±4.03e+01 −
$DCOP1-s6$	3.25e+01±0.00e+00	4.95e+01±2.58e+00 −	1.06e+01±1.81e-15	2.28e+02±5.80e+01 −
$DCOP2-s1$	1.65e+01±1.08e-14	4.24e+01±2.87e+00 −	1.95e+02±8.67e-14	4.07e+02±5.19e+01 −
$DCOP2-s2$	5.32e+01±1.45e-14	5.43e+01±5.16e+00 a	1.29e+02±5.78e-14	3.39e+02±4.88e+01 −
$DCOP2-s3$	3.56e+01±7.23e-15	5.09e+01±4.30e+00 −	1.98e+02±5.78e-14	4.54e+02±6.30e+01 −
$DCOP2-s4$	5.25e+01±2.17e-14	5.49e+01±4.20e+00 −	2.95e+02±1.73e-13	3.60e+02±8.63e+01 −
$DCOP2-s5$	5.29e+01±7.23e-15	5.41e+01±3.51e+00 a	1.40e+02±5.78e-14	3.61e+02±5.88e+01 −
$DCOP2-s6$	3.51e+01±0.00e+00	5.15e+01±4.88e+00 −	1.86e+02±0.00e+00	3.24e+02±3.49e+01 −
$DCOP3-s1$	3.56e+01±2.89e-14	3.72e+01±4.17e+00 −	7.60e+00±3.61e-15	1.84e+02±4.11e+01 −
$DCOP3-s2$	4.98e+01±0.00e+00	4.75e+01±3.59e+00 +	1.12e+01±5.42e-15	2.68e+02±9.83e+01 −
$DCOP3-s3$	5.18e+01±2.17e-14	4.30e+01±4.19e+00 +	1.22e+01±5.42e-15	2.21e+02±5.20e+01 −
$DCOP3-s4$	4.37e+01±2.17e-14	4.73e+01±4.29e+00 −	1.86e+01±1.08e-14	2.21e+02±5.34e+01 −
$DCOP3-s5$	5.22e+01±1.45e-14	4.27e+01±3.91e+00 +	1.18e+01±5.42e-15	2.19e+02±5.43e+01 −
$DCOP3-s6$	4.61e+01±0.00e+00	3.74e+01±3.58e+00 +	2.58e+01±1.08e-14	2.01e+02±4.93e+01 −
$DCOP4-s1$	1.07e+01±5.42e-15	2.92e+01±3.84e+00 −	1.32e+02±0.00e+00	3.01e+02±6.51e+01 −
$DCOP4-s2$	5.15e+01±1.45e-14	4.81e+01±3.89e+00 +	2.09e+02±8.67e-14	2.80e+02±9.61e+01 −
$DCOP4-s3$	3.73e+01±1.45e-14	3.77e+01±5.83e+00 a	7.94e+01±4.34e-14	2.79e+02±5.00e+01 −
$DCOP4-s4$	4.34e+01±1.45e-14	3.97e+01±3.63e+00 +	1.30e+02±5.78e-14	2.19e+02±6.76e+01 −
$DCOP4-s5$	3.13e+01±1.08e-14	3.87e+01±3.81e+00 −	5.88e+01±2.89e-14	2.41e+02±6.39e+01 −
$DCOP4-s6$	4.12e+01±7.23e-15	4.11e+01±4.70e+00 a	2.83e+02±1.16e-13	2.96e+02±3.79e+01 −
$DCOP5-s1$	1.70e+01±7.23e-15	3.61e+01±3.39e+00 −	8.20e+00±3.61e-15	1.96e+02±5.57e+01 −
$DCOP5-s2$	5.13e+01±3.61e-14	4.66e+01±4.33e+00 +	1.36e+01±5.42e-15	2.18e+02±5.30e+01 −
$DCOP5-s3$	1.85e+01±7.23e-15	4.59e+01±3.43e+00 −	7.60e+00±3.61e-15	2.02e+02±4.54e+01 −
$DCOP5-s4$	5.38e+01±2.89e-14	4.13e+01±4.11e+00 +	8.20e+00±3.61e-15	2.35e+02±7.72e+01 −
$DCOP5-s5$	5.87e+01±2.17e-14	4.35e+01±5.68e+00 +	1.30e+01±0.00e+00	2.25e+02±5.42e+01 −
$DCOP5-s6$	5.23e+01±1.45e-14	4.08e+01±3.61e+00 +	1.28e+01±5.42e-15	2.20e+02±5.32e+01 −
$DCOP6-s1$	9.34e+00±3.61e-15	2.67e+01±4.18e+00 −	6.32e+01±2.89e-14	2.41e+02±8.25e+01 −
$DCOP6-s2$	5.47e+01±0.00e+00	4.46e+01±4.93e+00 +	5.10e+01±0.00e+00	2.62e+02±6.94e+01 −
$DCOP6-s3$	2.82e+01±1.45e-14	3.68e+01±4.87e+00 −	7.94e+01±4.34e-14	2.18e+02±8.74e+01 −
$DCOP6-s4$	3.03e+01±1.81e-14	3.79e+01±3.70e+00 −	1.42e+01±7.23e-15	2.51e+02±8.17e+01 −
$DCOP6-s5$	2.93e+01±1.08e-14	3.66e+01±3.28e+00 −	8.94e+01±4.34e-14	2.26e+02±6.40e+01 −
$DCOP6-s6$	2.19e+01±0.00e+00	3.89e+01±4.53e+00 −	1.47e+02±8.67e-14	2.81e+02±6.72e+01 −

TABLE VII
COMPARISON BETWEEN DYCODE AND SDCO+DYCODE. "MEAN" AND "STD" DENOTE THE MEAN AND STANDARD DEVIATION OF off_err OR $eval\_num$ OVER 30 RUNS. "$-$", "$+$" AND "a" MEAN THAT SDCO+DYCODE PERFORMED SIGNIFICANTLY BETTER THAN, WORSE THAN, AND SIMILAR TO DYCODE ON THE TEST FUNCTION IN THAT ROW.

Func	off_err (Mean±Std)		$eval\_num$ (Mean±Std)	
	SDCO+dycode	DyCoDE	SDCO+dycode	DyCoDE
$DCOP1-s1$	7.27e+00±2.68e-01	1.90e+01±4.08e-01 −	2.00e+00±0.00e+00	2.59e+02±2.55e+01 −
$DCOP1-s2$	1.84e+01±6.57e-01	3.02e+01±9.54e-01 −	2.00e+00±0.00e+00	2.47e+02±1.88e+01 −
$DCOP1-s3$	2.83e+01±1.05e+00	3.29e+01±9.54e-01 −	2.00e+00±0.00e+00	2.54e+02±2.64e+01 −
$DCOP1-s4$	3.03e+01±9.76e-01	3.41e+01±8.65e-01 −	2.00e+00±0.00e+00	2.57e+02±2.47e+01 −
$DCOP1-s5$	2.83e+01±8.12e-01	3.18e+01±1.24e+00 −	2.00e+00±0.00e+00	2.78e+02±2.64e+01 −
$DCOP1-s6$	2.99e+01±7.87e-01	3.05e+01±1.13e+00 −	2.00e+00±0.00e+00	2.60e+02±2.44e+01 −
$DCOP2-s1$	1.93e+01±5.76e-01	4.48e+01±7.46e-01 −	3.16e+01±1.45e-14	1.52e+03±5.17e+01 −
$DCOP2-s2$	3.19e+01±1.55e+00	4.53e+01±8.06e-01 −	1.72e+01±2.21e-01	1.28e+03±5.14e+01 −
$DCOP2-s3$	4.31e+01±1.42e+00	5.36e+01±7.43e-01 −	2.42e+01±2.19e-01	1.80e+03±5.45e+01 −
$DCOP2-s4$	4.56e+01±1.57e+00	4.76e+01±8.68e-01 −	2.54e+01±1.08e-14	1.26e+03±5.78e+01 −
$DCOP2-s5$	4.26e+01±9.33e-01	4.83e+01±1.22e+00 −	1.50e+01±0.00e+00	1.32e+03±5.07e+01 −
$DCOP2-s6$	4.77e+01±1.38e+00	4.54e+01±7.33e-01 +	1.68e+01±7.23e-15	1.03e+03±3.75e+01 −
$DCOP3-s1$	2.64e+01±3.16e-01	3.16e+01±9.09e+00 a	2.00e+00±0.00e+00	2.84e+02±3.25e+01 −
$DCOP3-s2$	1.98e+01±6.36e-01	4.82e+01±6.07e+00 −	2.00e+00±0.00e+00	2.68e+02±3.32e+01 −
$DCOP3-s3$	3.00e+01±1.02e+00	3.91e+01±7.06e+00 −	2.00e+00±0.00e+00	2.73e+02±3.65e+01 −
$DCOP3-s4$	3.26e+01±7.21e-01	4.92e+01±4.91e+00 −	2.00e+00±0.00e+00	2.75e+02±4.54e+01 −
$DCOP3-s5$	3.60e+01±5.04e-01	4.68e+01±4.38e+00 −	2.20e+00±9.03e-16	2.79e+02±3.42e+01 −
$DCOP3-s6$	3.18e+01±8.31e-01	3.40e+01±3.85e+00 −	2.00e+00±0.00e+00	2.57e+02±2.79e+01 −
$DCOP4-s1$	1.43e+01±8.64e-01	4.14e+01±2.60e+00 −	9.85e+00±8.02e-01	1.06e+03±1.45e+02 −
$DCOP4-s2$	2.88e+01±9.38e-01	4.99e+01±3.27e+00 −	8.80e+00±3.61e-15	8.18e+02±1.85e+02 −
$DCOP4-s3$	3.98e+01±1.54e+00	5.03e+01±2.59e+00 −	2.85e+01±5.21e+00	9.95e+02±1.72e+02 −
$DCOP4-s4$	4.52e+01±1.22e+00	4.88e+01±2.72e+00 −	1.20e+01±2.25e+00	1.02e+03±7.00e+01 −
$DCOP4-s5$	3.89e+01±1.45e+00	5.45e+01±2.44e+00 −	1.03e+01±1.44e+00	9.72e+02±1.73e+02 −
$DCOP4-s6$	4.68e+01±1.81e+00	5.60e+01±3.39e+00 −	1.23e+01±2.55e+00	9.01e+02±1.45e+02 −
$DCOP5-s1$	8.47e+00±3.80e-01	4.42e+01±6.10e+00 −	2.00e+00±0.00e+00	2.82e+02±3.08e+01 −
$DCOP5-s2$	1.97e+01±7.38e-01	4.84e+01±7.40e+00 −	2.00e+00±0.00e+00	2.72e+02±3.24e+01 −
$DCOP5-s3$	3.19e+01±6.43e-01	4.76e+01±9.04e+00 −	2.00e+00±0.00e+00	2.76e+02±2.55e+01 −
$DCOP5-s4$	3.94e+01±6.16e-01	5.61e+01±2.38e+00 −	2.00e+00±0.00e+00	2.89e+02±3.06e+01 −
$DCOP5-s5$	4.38e+01±7.02e-01	5.77e+01±3.23e+00 −	2.00e+00±0.00e+00	2.96e+02±3.01e+01 −
$DCOP5-s6$	3.43e+01±8.64e-01	4.35e+01±7.70e+00 −	2.00e+00±0.00e+00	2.63e+02±2.44e+01 −
$DCOP6-s1$	1.05e+01±5.20e-01	4.14e+01±5.84e+00 −	2.89e+00±1.80e-01	6.48e+02±1.83e+02 −
$DCOP6-s2$	2.60e+01±1.02e+00	5.41e+01±3.63e+00 −	7.37e+00±4.87e-01	7.30e+02±2.34e+02 −
$DCOP6-s3$	4.14e+01±1.22e+00	5.08e+01±3.90e+00 −	8.94e+00±4.00e+00	6.45e+02±1.82e+02 −
$DCOP6-s4$	4.04e+01±1.31e+00	4.81e+01±4.28e+00 −	1.28e+01±3.51e+00	4.78e+02±6.88e+01 −
$DCOP6-s5$	3.72e+01±1.11e+00	5.73e+01±3.48e+00 −	2.10e+01±1.10e-01	1.01e+03±2.54e+02 −
$DCOP6-s6$	4.65e+01±1.33e+00	5.85e+01±3.63e+00 −	3.42e+02±4.09e+01	8.09e+02±1.53e+02 −

some test functions is that the SDCO methods conducted local search on every individual in X, some of which are not helpful but cost extra function evaluations. However, SDCO methods performed better than SELS, LTFR-DSPSO and DyCoDE in general. This demonstrates the effectiveness of offline set search in SDCO. Furthermore, the SDCO methods can conduct local search on each individual in the population in a parallel fashion. This will largely accelerate the reaction of SDCO methods to environmental changes in a parallel computing environment. All these results show the potential of the SDO framework, which combines offline computation and online optimization when solving DOPs.

VI. Conclusions and Future Work

In this paper, we proposed a new dynamic optimization framework SDO based on EAs to solve DOPs. The SDO framework introduces an offline evolutionary process to search a set of potential solutions beforehand, which can be utilized to solve DOPs online. After obtaining the solution set based on a set-oriented optimization process, SDO performs local search on some solutions in the set to find a satisfying solution for each environmental change. To show the potential of the new framework, SDO was instantiated on DCOPs to propose a new algorithm, SDCO, for DCOPs. To evaluate the efficacy of SDCO, we have conducted experiments on 36 DCOP benchmark test functions and compared the performance of SDCO with 3 state-of-the-art algorithms, namely SELS, LTFR-DSPSO and DyCoDE. The experimental results showed that SDCO reacts faster to environmental changes than the 3 compared algorithms. Furthermore, to validate the effectiveness of offline set search in SDCO, we have replaced the local search strategy of SDCO with the local search strategies used in SELS, LTFR-DSPSO and DyCoDE. The experimental results showed the benefits of obtaining a solution set beforehand when solving DOPs. All these results showed the potential of the newly proposed dynamic optimization framework.

In future work, SDCO will be tested on more dynamic test problems and real-world applications to further evaluate its performance. As the SDO framework is generally applied to dynamic optimization problems, it will be further instantiated for dynamic constraint satisfaction problems and dynamic multi-objective optimization problems. Some open issues also arise from this work. First, for dynamic optimization problems, it is important to investigate whether a solution set of limited size exists for which optimal solutions for any given environmental change can be found by doing local search on them. Second, whether the uniform sampling on the experimental parameters is the most effective method to search a good solution set for a dynamic problem needs further investigation. Third, when the solution set obtained in the offline computation does not contain contain reasonably close solutions for some environments, local search might need to be replaced by an alternative optimization method, which needs further study.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant No. 61906082), Honda Research Institute Europe (HRI-EU), National Key R&D Program of China (Grant No. 2017YFC0804003), Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531), and the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008).

References

[1] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno, "Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies," *European Journal of Operational Research*, vol. 151, no. 1, pp. 1–11, 2003.

[2] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012.

[3] I. Hatzakis and D. Wallace, "Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach," in *Proceedings of the 2006 Genetic and evolutionary computation conference (GECCO'06)*. ACM, 2006, pp. 1201–1208.

[4] C. Rossi, M. Abderrahim, and J. C. Díaz, "Tracking moving optima using kalman-based predictions," *Evolutionary computation*, vol. 16, no. 1, pp. 1–30, 2008.

[5] P. Filipiak and P. Lipinski, "Infeasibility driven evolutionary algorithm with feed-forward prediction strategy for dynamic constrained optimization problems," in *Applications of Evolutionary Computation*. Springer, 2014, pp. 817–828.

[6] H. G. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," DTIC Document, Tech. Rep., 1990.

[7] T. T. Nguyen and X. Yao, "Continuous dynamic constrained optimization–the challenges," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 6, pp. 769–786, 2012.

[8] J. J. Grefenstette *et al.*, "Genetic algorithms for changing environments," in *Proceedings of the 1992 International Conference on Parallel Problem Solving from Nature (PPSN'92)*, vol. 2, 1992, pp. 137–144.

[9] M. Campos and R. Krohling, "Bare bones particle swarm with scale mixtures of gaussians for dynamic constrained optimization," in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC'14)*. IEEE, 2014, pp. 202–209.

[10] C. Li, T. T. Nguyen, M. Yang, S. Yang, and S. Zeng, "Multi-population methods in unconstrained continuous dynamic environments: The challenges," *Information Sciences*, vol. 296, pp. 95–118, 2015.

[11] C. Bu, W. Luo, and L. Yue, "Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 1, pp. 14–33, 2017.

[12] H. Richter, "Memory design for constrained dynamic optimization problems," pp. 552–561, 2010.

[13] P. Filipiak and P. Lipinski, "Dynamic portfolio optimization in ultra-high frequency environment," in *Proceedings of the 2017 European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 34–50.

[14] F. Neri, J. Toivanen, G. L. Cascella, and Y.-S. Ong, "An adaptive multimeme algorithm for designing hiv multidrug therapies," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 2, pp. 264–278, 2007.

[15] V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, and T. Rossi, "An enhanced memetic differential evolution in filter design for defect detection in paper production," *Evolutionary Computation*, vol. 16, no. 4, pp. 529–555, 2008.

[16] K. Tang, P. Yang, and X. Yao, "Negatively correlated search," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 542–550, 2016.

[17] T. Kailath, "The divergence and bhattacharyya distance measures in signal selection," *IEEE transactions on communication technology*, vol. 15, no. 1, pp. 52–60, 1967.

[18] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[19] E. Mezura-Montes, C. A. C. Coello, and E. I. Tun-Morales, "Simple feasibility rules and differential evolution for constrained optimization," in *MICAI 2004: Advances in Artificial Intelligence*. Springer, 2004, pp. 707–716.

[20] G. Pamparà and A. P. Engelbrecht, "A generator for dynamically constrained optimization problems," in *Proceedings of the 2019 Genetic and Evolutionary Computation Conference (GECCO'19)*, 2019, pp. 1441–1448.

[21] Y. Wang, J. Yu, S. Yang, S. Jiang, and S. Zhao, "Evolutionary dynamic constrained optimization: Test suite construction and algorithm comparisons," *Swarm and Evolutionary Computation*, vol. 50, p. 100559, 2019.

[22] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proceedings of the 1999 IEEE Congress on Evolutionary Computation (CEC'99)*, vol. 3. IEEE, 1999, pp. 1875–1882.

[23] M.-Y. Ameca-Alducin, E. Mezura-Montes, and N. Cruz-Ramirez, "Differential evolution with combined variants for dynamic constrained optimization," in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC'14)*. IEEE, 2014, pp. 975–982.

[24] M. Campos and R. A. Krohling, "Entropy-based bare bones particle swarm for dynamic constrained optimization," *Knowledge-Based Systems*, vol. 000, pp. 1–21, 2015.

[25] M.-Y. Ameca-Alducin, E. Mezura-Montes, and N. Cruz-Ramírez, "A repair method for differential evolution with combined variants to solve dynamic constrained optimization problems," in *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference (GECCO'15)*. ACM, 2015, pp. 241–248.

[26] X. Lu, K. Tang, and X. Yao, "Speciated evolutionary algorithm for dynamic constrained optimisation," in *International Conference on Parallel Problem Solving from Nature (PPSN'16)*. Springer, 2016, pp. 203–213.

[27] S. W. Mahfoud, "Niching methods for genetic algorithms," *Urbana*, vol. 51, no. 95001, pp. 62–94, 1995.

[28] S. De, S. K. Pal, and A. Ghosh, "Genotypic and phenotypic assortative mating in genetic algorithm," *Information Sciences*, vol. 105, no. 1, pp. 209–226, 1998.

[29] H.-M. Voigt and J. M. Lange, "Local evolutionary search enhancement by random memorizing," in *Proceedings of the 1998 IEEE International Conference on Computational Intelligence (ICCI'98)*. IEEE, 1998, pp. 547–552.

[30] P. Chootinan and A. Chen, "Constraint handling in genetic algorithms using a gradient-based repair method," *Computers & operations research*, vol. 33, no. 8, pp. 2263–2281, 2006.

[31] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[32] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Springer Publishing Company, Incorporated, 2014.