Journal of Soft Computing and Information Technology (JSCIT)
Babol Noshirvani University of Technology, Babol, Iran
Journal Homepage: jscit.nit.ac.ir
Volume 11, Number 4, Winter 2023, pp. 36-49
Received: 08/04/2022, Revised: 10/17/2022, Accepted: 01/03/2023

Improving the Performance of the City Councils Evolution Algorithm by the Linear Reduction of the Population Size and the Search Space

Einollah Pira[1*] and Alireza Rouhi [2]
1*- Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran.
2- Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran.
[1*]pira@azaruniv.ac.ir and [2]rouhi@azaruniv.ac.ir

Corresponding author's address: Einollah Pira, Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran.

Abstract- City Councils Evolution algorithm (CCE) is a metaheuristic algorithm inspired by the formation process of the supreme council of a city due to the formation of councils from the smallest neighborhoods to the largest regions. In this paper, we want to improve the performance of CCE by applying two important changes. The first change is about the continuous reduction of the population size using Linear Population Size Reduction (LPSR) technique. In this technique, the population size in the initial iterations is considered large enough such that it can explore wide areas of search space. As the algorithm progresses, the population size is gradually reduced to increase the convergence speed. The second change is related to the domain of variables, which is constantly reduced to limit the search space, and so the possibility of finding optimal solutions is increased. To evaluate and compare the performance of ICCE with CCE, Chimp Optimization, Black Widow Optimization, Political Optimizer, Barnacles Mating Optimizer, Snake Optimizer, and Aquila Optimizer, we implement them on 29 test functions from 2017 IEEE Congress on Evolutionary Computation (CEC 2017). The results of Friedman mean rank and Wilcoxon signed-rank tests confirm the higher performance of ICCE compared to other algorithms.

Keywords- Evolution, metaheuristic algorithm, convergence speed, evolutionary computing, test function.

مجله علمی پژوهشی رایانش نرم و فناوری اطلاعات
دانشگاه صنعتی نوشیروانی بابل
صفحه مجله: jscit.nit.ac.ir
جلد ۱۱، شماره ۴، زمستان ۱۴۰۱ ، صفحه ۳۶-۴۹
دریافت: ۱۴۰۱/۰۵/۱۳، بازنگری: ۱۴۰۱/۰۷/۲۵، پذیرش: ۱۴۰۱/۱۰/۱۳

بهبود کارایی الگوریتم تکامل شوراهای شهر با کاهش خطی اندازه جمعیت و فضای جستجو

عین‌الله پیرا [*۱] و علیرضا روحی [۲]
[*]۱- دانشکده فناوری اطلاعات و مهندسی کامپیوتر، دانشگاه شهید مدنی آذربایجان، تبریز، ایران.
۲- دانشکده فناوری اطلاعات و مهندسی کامپیوتر، دانشگاه شهید مدنی آذربایجان، تبریز، ایران.
[1*]pira@azaruniv.ac.ir, [2]rouhi@azaruniv.ac.ir

* نشانی نویسنده مسئول: عین‌الله پیرا، تبریز، دانشگاه شهید مدنی آذربایجان، دانشکده فناوری اطلاعات و مهندسی کامپیوتر.

چکیده- الگوریتم تکامل شوراهای شهر (CCE)، یک نوع الگوریتم فراابتکاری است که با توجه به ماهیت تشکیل شوراها از کوچکترین محله‌ها تا بزرگترین مناطق شهری، از فرآیند تشکیل شورای عالی یک شهر الهام گرفته شده است. در این مقاله می‌خواهیم کارایی الگوریتم CCE را با دو تغییر مهم در آن بهبود بدهیم. اولین تغییر مربوط به کاهش پیوستهٔ اندازهٔ جمعیت با استفاده از تکنیک کاهش خطی جمعیت (LPSR) است. در این تکنیک، اندازهٔ جمعیت در تکرارهای اولیهٔ الگوریتم به اندازهٔ کافی بزرگ در نظر گرفته می‌شود تا الگوریتم بتواند مناطق وسیعی از فضای جستجو را پیمایش کند. با پیشروی الگوریتم، اندازهٔ جمعیت به‌تدریج کاهش داده می‌شود تا سرعت همگرایی افزایش یابد. دومین تغییر به دامنهٔ متغیرها مربوط می‌شود که به‌طور پیوسته کاهش می‌یابد تا فضای جستجو محدودتر شده و در نتیجه، امکان یافتن راه‌حل‌های بهینه افزایش پیدا کند. برای ارزیابی و مقایسهٔ کارایی *الگوریتم تکامل شوراهای شهر بهبودیافته* (تحت عنوان ICCE که در این مقاله مطرح شده است) با الگوریتم‌های تکامل شوراهای شهر (CCE)، بهینه‌سازی شامپانزه، بهینه‌سازی بیوه سیاه، بهینه‌ساز سیاسی، بهینه‌ساز جفت‌گیری بارناکل‌ها، بهینه‌سازی مار و بهینه‌ساز آکیلا، آن‌ها را روی ۲۹ تابع تست از مسابقات سال ۲۰۱۷ مربوط به کنگره IEEE در زمینه محاسبات تکاملی (CEC 2017) اجرا می‌کنیم. نتایج آزمون‌های میانگین رتبهٔ فریدمن و رتبه علامت‌دار ویلکاکسون، کارایی بالای الگوریتم ICCE را نسبت به الگوریتم‌های مذکور تأیید می‌کنند.

واژه‌های کلیدی: تکامل، الگوریتم فراابتکاری، سرعت همگرایی، محاسبات تکاملی، تابع تست.

۱- مقدمه

معمولاً روش‌های بهینه‌سازی کلاسیک نمی‌توانند راه‌حل بهینه‌ای را برای مسائل پیچیدهٔ مهندسی که دارای فضای جستجوی بسیار بزرگ هستند پیدا کنند. به همین دلیل، در سال‌های اخیر، توجه زیادی به روش‌های بهینه‌سازی غیرکلاسیک از جمله الگوریتم‌های فراابتکاری[۱] شده است [۱]. این الگوریتم‌ها در زمینه‌های مختلفی اعم از مسیریابی موبایل [۲]، مسائل برنامه‌ریزی هوش مصنوعی [۳-۵]، کاربردهای ارتباطی [۶]، کاربردهای پردازش تصویر [۷] و مسائل تحلیل ایمنی سیستم‌های نرم‌افزاری [۸] به‌طور موفقیت‌آمیزی به‌کار رفته‌اند. علت جلب توجه و موفقیت این الگوریتم‌ها به چهار ویژگی *سادگی*[۲]، *انعطاف‌پذیری*[۳]، *اجتناب از بهینگی محلی*[۴] و *مستقل از گرادیان*[۵] مربوط می‌شود [۹]. ویژگی سادگی به این معنی است که این الگوریتم‌ها از مفاهیم ساده مثل رفتار حیوانات در طبیعت نشأت می‌گیرند. منظور از ویژگی انعطاف‌پذیری این است که الگوریتم‌های فراابتکاری می‌توانند بدون هیچ تغییر ساختاری خاصی برای مسائل مربوط به فیلدهای مختلف استفاده شوند. ماهیت تصادفی بودن الگوریتم‌های فراابتکاری باعث شده است که از گیرافتادن در بهینگی محلی اجتناب کنند. برخلاف الگوریتم‌های کلاسیک که از مشتق‌گیری

فضای جستجو جهت محاسبهٔ راه‌حل‌های بهینه استفاده می‌کنند، الگوریتم‌های فراابتکاری نیازی به چنین محاسبات پیچیدهٔ ریاضی ندارند.

الگوریتم‌های فراابتکاری دارای دو ویژگی مهم به نام‌های اکتشاف[۶] و انتفاع[۷] هستند [۱۰]. هدف از ویژگی *اکتشاف* این است الگوریتم بتواند به‌طور آزادانه و تصادفی، مناطق مختلفی از فضای جستجو را پیمایش کند که این امر باعث می‌شود از بهینگی محلی به دور باشد. مفهوم ویژگی *انتفاع* این است که الگوریتم بتواند حول راه‌حل‌های امیدبخش فعلی، جستجوی محلی انجام داده تا کیفیت این راه‌حل‌ها را بهبود ببخشد. اگر ویژگی اکتشاف الگوریتمی بر ویژگی انتفاع آن غلبه کند آن الگوریتم رفتار تصادفی‌تر و غیرقابل پیش‌بینی‌تری را خواهد داشت (برای مثال، الگوریتم تصادفی). برعکس، اگر در الگوریتمی، ویژگی انتفاع بر ویژگی اکتشاف غلبه کند آن الگوریتم به احتمال زیادی در بهینگی محلی گیر می‌افتد (برای مثال، الگوریتم تپه‌نوردی). معمولاً، الگوریتم‌های فراابتکاری پارامترهای قابل تنظیمی دارند که سعی می‌کنند بین این دو ویژگی، توازن مناسبی برقرار کنند. تمامی الگوریتم‌های فراابتکاری دارای عملکرد تقریباً مشابهی هستند. این الگوریتم‌ها با یک جمیعت تصادفی از راه‌حل‌های کاندید شروع کرده و با استفاده از عملگرهایی و در طی مراحل مختلف سعی می‌کنند جمعیت موجود را بهبود ببخشند تا به راه‌حل (تقریباً) بهینه برسند. بسته به اندازهٔ جمعیت، با دو گروه از الگوریتم‌ها سروکار داریم: الگوریتم‌های تک‌راه‌حل (برای مثال، الگوریتم تبرید شبیه‌سازی[۸] [۱۱]) و چندراه‌حل. الگوریتم‌های چندراه‌حل دارای سه مزیت مهم نسبت به الگوریتم‌های تک‌راه‌حل هستند: (۱) نواحی مختلفی از فضای جستجو به‌طور همزمان توسط راه‌حل‌های کاندید مورد پیمایش قرار می‌گیرند، (۲) اطلاعات بدست آمده توسط هر راه‌حل کاندید با بقیه مورد اشتراک قرار می‌گیرند، و (۳) راه‌حل‌های کاندید به همدیگر کمک می‌کنند تا در بهینگی محلی گیر نیافتند.

هرچند در سال‌های اخیر، الگوریتم‌های فراابتکاری زیادی ارائه شده‌اند، اما طبق قضیه NFL[9] تعداد زیادی از مسائل بهینه‌سازی وجود دارند که هنوز به‌طور کامل حل نشده و به الگوریتم‌های بهینه‌سازی جدیدی نیاز است [۱۲]. قضیهٔ NFL ثابت کرده است که هیچ الگوریتم بهینه‌سازی وجود ندارد که بتواند تمامی مسائل بهینه‌سازی موجود را حل کند. به‌عبارت دیگر، یک الگوریتم خاص ممکن است گروهی از مسائل بهینه‌سازی را با موفقیت حل کند و همین الگوریتم ممکن است برای گروه دیگری از مسائل بهینه‌سازی با شکست مواجه شود. با توجه به این قضیه، الگوریتم فراابتکاری جدیدی با نام تکامل شوراهای شهر (CCE) ارائه شده است که با توجه به ماهیت تشکیل شوراها از کوچکترین محله‌ها تا بزرگترین مناطق شهری، از فرآیند تشکیل شورای عالی یک شهر الهام گرفته است. اگرچه این الگوریتم دارای کارایی بهتری نسبت به الگوریتم‌های دیگر است، اما همچنان امکان بهبود کارایی آن وجود دارد [۱].

شرط موفقیت الگوریتم‌های فراابتکاری این است که بتوانند در تکرارهای ابتدایی، ویژگی اکتشاف را نسبت به ویژگی انتفاع تقویت کرده و بالعکس، در تکرارهای پایانی، ویژگی انتفاع را نسبت به ویژگی اکتشاف تقویت کنند. در نتیجه، در تکرارهای ابتدایی، مناطق بیشتری از فضای جستجو بررسی شده و در تکرارهای پایانی، مناطق اطراف راه‌حل‌های امیدبخش جهت یافتن راه‌حل نهایی، بیشتر مورد بررسی قرار خواهند گرفت. برای دستیابی به این اهداف دو روش وجود دارد: ۱- اندازه جمعیت در تکرارهای اولیهٔ الگوریتم به اندازه کافی بزرگ و در تکرارهای پایانی کوچکتر در نظر گرفته شود، و ۲- دامنهٔ متغیرها در تکرارهای پایانی محدودتر شود که این امر باعث کوچکترشدن فضای جستجو شده و در نتیجه، امکان یافتن راه‌حل‌های بهینه افزایش می‌یابد. قابل ذکر است که در CCE و بقیهٔ الگوریتم‌های فراابتکاری ارائه شده قبلی، معمولاً اندازه جمعیت در تمامی تکرارها ثابت در نظر گرفته می‌شود. بنابراین، این الگوریتم‌ها در مسائلی که فضای جستجوی بسیار بزرگی دارند، به عبارتی دارای ابعاد و دامنهٔ بزرگی هستند، کارایی پایینی خواهند داشت.

در این مقاله می‌خواهیم کارایی الگوریتم CCE را با دو تغییر مهم کاهش خطی جمعیت[۱۰] (LPSR) [۱۳] و کاهش فضای جستجو[۱۱] (SSR) بهبود ببخشیم. در LPSR، اندازهٔ جمعیت به‌طور پیوسته کاهش داده می‌شود. به این صورت که، اندازهٔ جمعیت در تکرارهای اولیهٔ الگوریتم، به اندازهٔ کافی بزرگ در نظر گرفته می‌شود تا بتواند مناطق وسیعی از فضای جستجو را کاوش کند. با پیشروی الگوریتم، اندازه جمعیت به‌تدریج کاهش داده می‌شود تا باعث افزایش سرعت همگرایی شود. در SSR، تلاش می‌شود با پیشروی الگوریتم، فضای جستجو محدودتر و هدفمندتر شود. برای این منظور، در تعداد ثابتی از تکرارهای اولیهٔ الگوریتم، دامنهٔ متغیرها ثابت در نظر گرفته می‌شود. سپس، با توجه به راه‌حل‌های شایسته‌تر، دامنهٔ متغیرها برای تکرارهای بعدی الگوریتم کوچک‌ترشده و در نتیجه، فضای جستجو کاهش می‌یابد که این امر باعث می‌شود احتمال رسیدن به راه‌حل‌های بهینه افزایش یابد.

برای ارزیابی و مقایسهٔ کارایی الگوریتم تکامل شوراهای شهر بهبودیافته (معروف به ICCE) با کارایی الگوریتم‌های CCE (سال انتشار: ۲۰۲۲)، CHOA (بهینه‌سازی شامپانزه: ۲۰۲۰)، BWO

(بهینه‌سازی بیوه سیاه: ۲۰۲۰)، PO (بهینه‌ساز سیاسی: ۲۰۲۰)، BMO (بهینه‌ساز جفت‌گیری بارناکل‌ها: ۲۰۲۰)، SO (بهینه‌سازی مار: ۲۰۲۲) و AO (بهینه‌ساز آکیلا: ۲۰۲۱)، آن‌ها را روی ۲۹ تابع تست متعلق به مسابقات سال ۲۰۱۷ مربوط به کنگره IEEE در زمینهٔ محاسبات تکاملی (CEC 2017) اجرا می‌کنیم. در این مقاله، کارایی هر الگوریتم از نظر رسیدن به نزدیک‌ترین راه‌حل به جواب بهینه، نرخ موفقیت و سرعت همگرایی مورد بررسی قرار می‌گیرد.

در ادامه مقاله و در بخش دوم، الگوریتم‌های فراابتکاری چندراه‌حل را مرور خواهیم کرد. در بخش سوم، ابتدا الگوریتم تکامل شوراهای شهر (CCE) تشریح خواهد شد و سپس جزئیات الگوریتم ICCE را مطرح خواهیم کرد. بخش چهارم به جزئیات پیاده‌سازی و نتایج تجربی می‌پردازد. در بخش پنجم نیز به نتیجه‌گیری و پیشنهاداتی برای کارهای بعدی می‌پردازیم.

۲- مرور ادبیات

با توجه به نوع منبع الهام، الگوریتم‌های فراابتکاری چندراه‌حل به چهار گروه *مبتنی بر تکامل*، *مبتنی بر ازدحام*، *مبتنی بر فیزیک* و *مبتنی بر انسان* تقسیم می‌شوند.

الگوریتم‌های مبتنی بر تکامل از مفاهیم تکامل بیولوژیکی در طبیعت الهام گرفته‌اند. الگوریتم ژنتیک (GA) از مشهورترین الگوریتم‌ها در این گروه است که مفهوم تکامل داروین مبتنی بر اصل بقاء، بهترین‌ها در هر نسل را شبیه‌سازی می‌کند [۱۴]. در هر تکرار از این الگوریتم، تعدادی راه‌حل کاندید شایسته از بین جمعیت فعلی انتخاب می‌شود و پس از انجام اَعمال ترکیب و جهش روی این راه‌حل‌های انتخاب شده، جمعیت جدید با استفاده از جمعیت فعلی و راه‌حل‌های جدیداً تولیدشده تشکیل می‌شود. بدیهی است که کیفیت جمیعت جدید به‌تدریج بهتر شده و رسیدن به جواب بهینه را تضمین می‌کند. برنامه‌نویسی تکاملی (EP) [۱۵]، تکامل تفاضلی (DE) [۱۶] و بهینه‌ساز مبتنی بر جغرافیای زیستی (BBO) [۱۷] از نمونه‌های دیگر این گروه هستند.

الگوریتم‌های مبتنی بر ازدحام، رفتار اجتماعی بعضی از حیوانات مانند مورچه‌ها، پرندگان و جغدها را شبیه‌سازی می‌کنند. در این الگوریتم‌ها، عامل‌ها فضای جستجو را با استفاده از هوش جمعی پیمایش می‌کنند. بهینه‌سازی ازدحام ذرات (PSO) [۱۸] و بهینه-سازی کلونی مورچه‌ها (ACO) [۱۹] جزو نمونه‌های مشهور در این گروه هستند که اولی از حرکت گروهی پرندگان و دومی از رفتارهای مورچه‌ها در حین جستجوی غذا الهام گرفته است. الگوریتم PSO هر راه‌حل را معادل یک ذره (پرنده) در نظر گرفته که دارای موقعیتی در فضای جستجو است. در هر تکرار، هر ذره تلاش می‌کند موقعیت جدید خود را با توجه به بهترین موقعیت کسب شده در تکرارهای قبلی توسط خودش و اعضای دیگر بهبود بخشد. در الگوریتم ACO نیز هر راه‌حل به‌صورت یک مورچه در نظر گرفته شده که از هوش اجتماعی مورچه‌ها برای یافتن کوتاه‌ترین مسیر بین لانه و منبع غذایی جهت بهبود میزان شایستگی راه‌حل‌ها استفاده می‌شود. الگوریتم‌های کلونی زنبورهای مصنوعی (ABC) [۲۰]، الگوریتم گرده‌افشانی گل (FPA) [۲۱]، جستجوی کلاغ (CSA) [۲۲]، بهینه‌سازی شامپانزه (CHOA) [۲۳]، بهینه‌سازی بیوه سیاه (BWO) [۲۴]، بهینه‌ساز جفت‌گیری بارناکل‌ها (BMO) [۲۵]، بهینه‌سازی مار (SO) [۲۶]، بهینه‌ساز آکیلا (AO) [۲۷]، بهینه‌سازی ملخ (GOA) [۲۸] نمونه‌های دیگر این گروه هستند.

الگوریتم‌های مبتنی بر فیزیک از قوانین فیزیکی در طبیعت نشأت می‌گیرند. تبرید شبیه‌سازی‌شده (SAA) [۱۱] از مشهورترین الگوریتم‌ها در این گروه است که مبتنی بر تبرید فلزات است. در روش تبرید تدریجی، فلزات پس از قرارگیری در معرض حرارت زیاد به‌طور تدریجی سرد می‌شوند. در واقع، این الگوریتم یک حرکت تکراری را با توجه به پارامتر دمای متغیر اتخاذ می‌کند که از تراکنش تبرید فلزات تقلید می‌کند. الگوریتم جستجوی گرانشی (GSA)، نمونهٔ دیگری از این گروه است که از برهم‌کنش گرانشی بین اجسام الهام می‌گیرد [۲۹]. در این الگوریتم، هر راه‌حل کاندید معادل یک شئ در نظر گرفته شده که جِرم شئ، میزان شایستگی آن را نشان می‌دهد. در طی روند تکرار الگوریتم، نیروی گرانشی مابین اشیاء باعث جذب آن‌ها به همدیگر می‌شود و در نتیجه، اشیاء سبک‌تر (راه‌حل‌های با برازندگی پایین) به‌آرامی به سمت اشیاء سنگین‌تر (راه‌حل‌های با برازندگی بالا) حرکت می‌کنند. بدیهی است که در پایان اجرای الگوریتم، یک شئ با بیشترین جِرم باقی می‌ماند که در واقع نشان‌دهندهٔ شایسته‌ترین راه‌حل خواهد بود. از الگوریتم‌های دیگر این گروه می‌توان به بهینه‌سازی الهام گرفته از اپتیک (OIO) [۳۰]، الگوریتم سیاه‌چاله[۱۲] (BHA) [۳۱]، الگوریتم چرخهٔ آب[۱۳] (WCA) [۳۲] و بهینه‌ساز مبتنی بر گرادیان[۱۴] [۳۳] اشاره کرد.

الگوریتم‌های مبتنی بر انسان از رفتار اجتماعی انسان‌ها و همچنین تعامل بین آن‌ها نشأت می‌گیرند. از برجسته‌ترین الگوریتم‌های این گروه می‌توان به الگوریتم بهینه‌سازی مبتنی بر آموزش و یادگیری[۱۵] (TLBO) اشاره کرد که از فرایند دومرحله‌ای یادگیری و آموزش در یک کلاس درسی الگو گرفته است [۳۴]. در مرحلهٔ آموزش، میزان شایستگی افراد کلاس (جمعیت) از طریق آموزش

توسط بهترین فرد کلاس افزایش داده می‌شود. در حالی که، در مرحلهٔ یادگیری، بقیهٔ افراد جمعیت، اطلاعات خود را به همدیگر یاد می‌دهند. بهینه‌سازی گروه اجتماعی (SGO)، الگوریتم دیگری در این گروه است که مبتنی بر تعامل اعضای مختلف گروه برای حل مسائل پیچیده می‌باشد [۳۵]. این الگوریتم دارای دو مرحلهٔ بهبود و کسب است که در اولین مرحله، دانش هر عضو از طریق تعامل با بهترین عضو افزایش می‌یابد در حالی که در دومین مرحله، دانش هر یک از اعضای گروه توسط بهترین عضو و سایر اعضایی که به‌طور تصادفی انتخاب شده‌اند افزایش می‌یابد. تکامل شوراهای شهر (CCE)، الگوریتم دیگری در این گروه است که با توجه به ماهیت تشکیل شوراها از کوچکترین محله‌ها تا بزرگترین مناطق شهری، از فرآیند تشکیل شورای عالی یک شهر الهام گرفته است. با توجه به شیوه سلسله‌مراتبی تکامل شوراها، CCE با استفاده از ساختار سلسله‌مراتبی، مانند یک درخت، اعضای شورا و رؤسا را مدل‌سازی می‌کند [۱]. نتایج اجرای این الگوریتم بر روی ۴۹ تابع تست، کارایی بالای این الگوریتم نسبت به الگوریتم‌های دیگر، از جمله بهینه‌ساز تعادل (EO) [۳۶]، بهینه‌سازی بیوه سیاه (BWO) [۲۴]، بهینه‌ساز سیاسی (PO) [۳۷]، بهینه‌ساز جفت‌گیری بارناکل‌ها (BMO) [۲۵]، بهینه‌سازی شامپانزه (CHOA) [۲۳]، بهینه‌ساز آکیلا (AO) [۲۷] و بهینه‌ساز اسب وحشی (WHO) [۳۸] را نشان می‌دهد. مشکل بزرگ این الگوریتم آن است که اگر مسأله مورد نظر دارای فضای جستجوی بسیار بزرگی باشد، به عبارتی دارای ابعاد و دامنه بزرگی باشد، کارایی پایینی خواهد داشت. از الگوریتم‌های دیگر این گروه می‌توان به بهینه‌ساز سیاسی (PO) [۳۷]، بهینه‌سازی سیاست حریصانه (GPO) [۳۹]، بهینه‌سازی مبارزات انتخاباتی (ECO) [۴۰]، الگوریتم بهینه‌سازی پارلمانی[۱۶] (POA) [۴۱] و بهینه‌سازی فقیر و غنی[۱۷] (PRO) [۴۲] اشاره کرد.

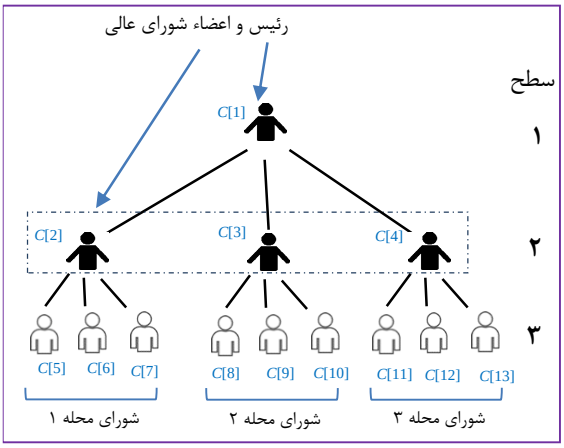
۳- الگوریتم تکامل شوراهای شهر بهبودیافته (ICCE)

در کشورهای دموکراتیک، شورای عالی هر شهر نهادی است که شهردار و مسئولان شهر را انتخاب می‌کند. بنابراین، سعی می‌شود که تک‌تک شهروندان آن شهر در تشکیل شورای عالی مشارکت کنند. برای این منظور، هر شهر به تعدادی منطقهٔ بزرگ تقسیم می‌شود که هر کدام دارای یک شورای محلی بوده و رؤسای این شوراها، شورای عالی را تشکیل می‌دهند. مشابه با شورای عالی، هر منطقهٔ بزرگ نیز به تعدادی منطقهٔ کوچک تقسیم می‌شود که هر کدام دارای شورای محلی بوده و رؤسای این شوراها، شورای محلی آن منطقه را تشکیل می‌دهند. این روند برای مناطق کوچک و نهایتاً کوچک‌ترین محله‌ها تکرار می‌شود. نکتهٔ مهم در این تکامل (فرایند) این است که برای هر شورا، یک عضو با بهترین کارایی به‌عنوان رئیس آن شورا انتخاب می‌شود. از ملاک‌های کارایی می‌توان به میزان تحصیلات، تجارب اجرایی و غیره اشاره کرد. واضح است که هر کدام از اعضای شوراها تمایل دارند در انتخابات بعدی به‌عنوان رئیس آن شورا برگزیده شوند. برای این منظور، آن‌ها باید سعی کنند کارایی‌شان را در هر ملاکی افزایش دهند تا مجموع کارایی کلی هر کدام از آن‌ها از کارایی رئیس فعلی بیشتر شود.

الگوریتم CCE یک الگوریتم فراابتکاری است که با توجه به تشکیل شوراها از کوچکترین محله‌ها تا بزرگترین مناطق شهری از فرآیند تشکیل شورای عالی یک شهر نشأت می‌گیرد [۱]. برای این منظور، این الگوریتم، فرآیند تکامل شوراهای شهر را با یک ساختار سلسله‌مراتبی، مثل درخت (اصطلاحاً، درخت شوراها نامیده می‌شود) مدل‌سازی می‌کند. شکل ۱، نمونه‌ای از درخت شوراها (C) را برای یک شهر فرضی با سه شورای محلی و یک شورای عالی نشان می‌دهد. مطابق با این شکل، رئیس شورای عالی در سطح یک (ریشهٔ درخت یعنی $C[1]$) و بقیهٔ اعضای این شورا در سطح دو قرار می‌گیرند. در واقع، اعضای یک شورا در گره‌های فرزند متعلق به گره والد (رئیس شورا) نگهداری می‌شوند. بنابراین، می‌توان گفت که گره‌های موجود در آخرین سطح درخت (سطح سوم در شکل ۱) اعضای شوراهای محله‌ها را نگهداری می‌کنند. با توجه به توضیحات ذکر شده در مورد تکامل شوراها می‌توان نتیجه گرفت که درخت شوراها باید به‌صورت یک هرم بیشینه باشد به این مفهوم که مقدار هر گره والد (کارایی رئیس یک شورا) باید از مقادیر همهٔ گره‌های فرزندان (کارایی اعضای آن شورا) بیشتر یا مساوی باشد. اگر هر شورا d عضو و ارتفاع درخت شوراها برابر h باشد اندازهٔ آرایه C برای نگهداری این درخت برابر $N=1+d^1+....+d^h=(d^h-1)/(d-1)$ خواهد بود، همچنین اگر $C[j]$ رئیس یک شورا را نگهداری کند اعضاء این شورا در گره‌های $C[d*j-d+2 \ \dots \ d*j+1]$ نگهداری خواهند شد. برعکس، اگر $C[i]$ عضوی از یک شورا باشد رئیس این شورا در خانه $C[floor(\frac{i-2}{d})+1]$ نگهداری خواهد شد که تابع $floor(x)$ بزرگترین عدد صحیح کوچک‌تر یا مساوی x را برمی‌گرداند. قابل ذکر است که هر $C[i]$ به‌صورت یک بردار است که طول آن برابر تعداد ملاک‌های کارایی و مقادیر آن، نمرات عضو i-ام شورا را در این ملاک‌ها نشان می‌دهند.

معمولاً، هر الگوریتم فراابتکاری عملگرهایی برای بهبود برازندگی جمعیت فعلی دارد که در الگوریتم CCE، تابع *improve* این کار را انجام می‌دهد. این تابع، دو عضو جمعیت با نام‌های X (به‌عنوان

عضو شورا) و Y (به‌عنوان رئیس شورا) به همراه شماره ملاک (cr) و کران‌های پایین و بالای ملاک موردنظر (متغیر) را گرفته و کارایی X را در ملاک cr-ام با توجه به Y افزایش می‌دهد. شکل ۲، شبه‌کد مربوط به این تابع را نشان می‌دهد. مطابق با این شکل، این تابع، ۴ راه‌حل جدید را تولید کرده و بعد از بررسی کران بالا و پایین متغیر شماره cr در هر کدام از این راه‌حل‌ها (با استفاده از تابع *check*)، شایسته‌ترین عضو از بین این اعضای جدید را با استفاده از تابع *max* محاسبه کرده و برمی‌گرداند. ضمناً، برای بهبود کارایی رئیس شورای عالی، یعنی $C[1]$، ابتدا یک متغیر (ژن) از آن به‌صورت تصادفی انتخاب شده و مقدار آن با یک مقدار تصادفی جدید جایگزین می‌شود ($C'[1]$)، سپس، بردارهای C و C' به تابع *improve* فرستاده می‌شوند.



شکل ۱- نمونه‌ای از درخت شوراها برای یک شهر فرضی [۱]

مشابه با بقیه الگوریتم‌های فراابتکاری، الگوریتم CCE نیز با یک جمعیت اولیه از راه‌حل‌های کاندید شروع می‌کند که در ابتدا به‌صورت تصادفی تولید شده‌اند. با فرض اینکه N و m به‌ترتیب به اندازهٔ جمعیت (تعداد اعضاء و رؤسای شورها) و تعداد متغیرها (تعداد ملاک‌های کارایی) اشاره کنند، هر راه‌حل کاندید با یک بردار به طول m و در نتیجه، جمعیت C نیز با یک آرایه به‌طول N از بردارهای به طول m (ماتریس $N\times m$) نشان داده خواهد شد. قابل ذکر است که برای هر راه‌حل (عضو/رئیس شورا)، مجموع کارایی‌های (نمرات) آن عضو در ملاک‌های ارزیابی می‌تواند به‌عنوان میزان برازندگی آن راه‌حل در نظر گرفته شود. در ادامه الگوریتم، دو مرحله زیر به‌ازای هرکدام از m متغیر (ملاک) و به تعداد مشخص‌شده توسط کاربر (*maxIterations*) تکرار می‌شوند:

مرحله ۱: بعد از محاسبه مقدار برازندگی (کارایی) تمام اعضای جمعیت، آرایه C به یک هرم بیشینه تبدیل می‌شود.

مرحله ۲: برازندگی (کارایی) همه اعضاء از سطح یک (ریشه درخت) تا آخرین سطح درخت با استفاده از تابع *improve* بهبود داده می‌شود.

در پایان الگوریتم، $C[1]$ دارای بیشترین برازندگی خواهد بود و به عنوان بهترین راه‌حل به کاربر اعلام می‌شود. شکل ۳، فلوچارت الگوریتم CCE را نشان می‌دهد.

Function 1 The ***improve*** function

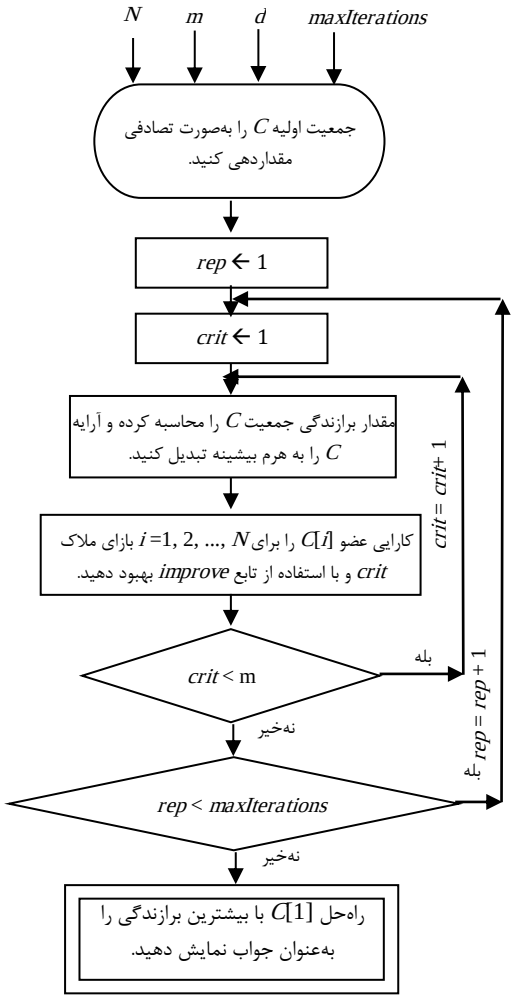
Input: X and Y: a council member and its boss respectively, *cr, lb, ub*.
Output: an individual with the highest fitness.
1: k = a value is randomly selected as either 1 or 2;
2: *rnd* = a random number between 0 and 1;
3: $Z1 = X$; $Z2 = X$; $Z3 = X$; $Z4 = X$;
4: $\alpha = (-1)^{k+1} * rnd$;
5: $Z1[cr] = X[cr] + \alpha * Y[cr]$;
6: $Z2[cr] = X[cr] + (1-\alpha) * Y[cr]$;
7: $Z3[cr] = Y[cr] + \alpha * X[cr]$;
8: $Z4[cr] = Y[cr] + (1-\alpha) * X[cr]$;
9: *check*($Z1[cr]$, *lb,ub*); *check*($Z2[cr]$, *lb,ub*);
10: *check*($Z3[cr]$, *lb,ub*); *check*($Z4[cr]$, *lb,ub*);
11: **return** *max*($Z1$, $Z2$, $Z3$, $Z4$);

شکل ۲- شبه‌کد تابع *improve* [۱]

در ادامه این بخش می‌خواهیم جزئیات نسخهٔ بهبودیافتهٔ الگوریتم تکامل شوراهای شهر (CCE) به‌نام ICCE را ارائه کنیم. این الگوریتم شامل دو تغییر مهم کاهش خطی اندازهٔ جمعیت (LPSR) و کاهش فضای جستجو (SSR) می‌باشد. معمولاً، پارامتر اندازهٔ جمعیت، نقش مهمی در کنترل سرعت همگرایی الگوریتم‌های فراابتکاری دارد، به این معنی که اندازه‌های بزرگ در تکرارهای اولیه باعث تقویت قابلیت اکتشاف می‌شوند و اندازه‌های کوچک در تکرارهای آخر، قابلیت انتفاع را تقویت کرده و باعث افزایش سرعت همگرایی می‌شوند. با توجه به این نکته، الگوریتم ICCE از تکنیک LPSR [۱۳] استفاده می‌کند تا برخلاف الگوریتم CCE، از اندازه‌های متغیر جمعیت بهره ببرد. در LPSR، اندازهٔ جمعیت در تکرار (نسل) اولیهٔ الگوریتم به مقدار بزرگ *Ninit* مقداردهی می‌شود و در تکرارهای بعدی، مقدار آن با استفاده از یک تابع خطی طوری کاهش داده می‌شود که در آخرین نسل (تکرار *maxIterations*- ام) برابر *Nmin* شود. رابطه‌ی (۱)، این تابع خطی را نشان می‌دهد که در آن، متغیر *Nnew*، اندازهٔ جمعیت برای تکرار بعدی را نشان می‌دهد. همچنین، متغیرهای *nfe* و *nfeMax*، به‌ترتیب به تعداد فراخوانی‌های تابع برازندگی تا تکرار فعلی الگوریتم و حداکثر مقدار پیش‌بینی‌شده برای تعداد کل فراخوانی‌های تابع برازندگی مربوط می‌شوند.

$$Nnew = round[(\frac{Nmin - Ninit}{nfeMax}) \times nfe + Ninit] \quad (۱)$$

بعد از پایان تکرار فعلی الگوریتم، مقادیر متغیرهای *nfe* و *nfeMax* بِروزرسانی شده و مقدار *Nnew* با استفاده از رابطه‌ی (۱) محاسبه می‌شود. اگر مقدار بدست آمده برای *Nnew* کمتر از اندازه جمعیت فعلی (*N*) باشد تعداد *Nnew* از شایسته‌ترین‌های جمعیت فعلی به جمعیت بعدی منتقل شده و بقیه راه‌حل‌ها (به تعداد *N-Nnew*) نادیده گرفته می‌شوند. ضمناً، مقدار *N* نیز به مقدار جدید *Nnew* به‌روزرسانی می‌شود.



شکل ۳- فلوچارت الگوریتم CCE [۱]

بدیهی است که هر اندازه فضای جستجو در یک مسأله بهینه‌سازی وسیع‌تر باشد، یافتن جواب بهینه در آن مسأله سخت‌تر خواهد شد. با توجه به این نکته، الگوریتم ICCE از تکنیک SSR استفاده می‌کند تا با پیشروی الگوریتم، فضای جستجو را با حذف مناطقی که احتمالِ وجود جواب بهینه در آنها پایین‌تر هست را محدودتر کرده و البته هدفمندتر سازد. فرض کنیم دامنهٔ ابتدایی همهٔ متغیرهای مسأله به‌صورت [*lb*, *ub*] باشد، که در آن، متغیرهای *lb* و *ub*، به‌ترتیب به کران پایین و بالا مربوط می‌شوند. به ازای این دامنهٔ اولیه، الگوریتم ICCE اجرا شده و به تعداد *initIterLearn* مرتبه، به صورت مداوم و بدون تغییر دامنهٔ متغیرها تکرار می‌شود. از آنجا که در هر تکرار، اعضای جمعیت به‌صورت یک هرم بیشنیه در می‌آیند به این مفهوم که شایسته‌ترین اعضاء در سطوح پایین درخت (سطوح یک، دو و ....) نگهداری می‌شوند، بنابراین به تعداد *nLearn* از این اعضای شایسته را در آرایه *history* ذخیره می‌کنیم. در واقع، بعد از اتمام *initIterLearn* بار تکرار اولیه الگوریتم، آرایه *history* شامل تعداد *nLearn*× *initIterLearn* از شایسته‌ترین راه‌حل‌ها خواهد بود. در ادامه، تکنیک SSR از اطلاعات راه‌حل‌های موجود در آرایهٔ *history* استفاده می‌کند تا دامنه جدیدی برای متغیرها محاسبه نماید. برای این‌کار، دامنه هر متغیر به دو زیردامنه چپِ [*lb*,(*lb*+*ub*)/2] و راستِ [(*lb*+*ub*)/2, *ub*] تقسیم شده و با توجه به مقادیر متغیرها در این راه‌حل‌ها مشخص می‌شود که در کدام یک از این قسمت‌ها، تعداد بیشتری از این راه‌حل‌ها قرار دارند. بیشتربودن تعداد راه‌حل‌ها در یک قسمت نشان‌دهنده این است که آن قسمت، امیدبخش‌تر از قسمت دیگر است و در ادامه الگوریتم، آن قسمت به عنوان دامنه جدید متغیر در نظر گرفته می‌شود. با شناسایی دامنه جدید برای تک‌تک متغیرها، الگوریتم به تعداد *iterLearn* بار بازای این دامنه‌های جدید اجرا شده و مشابه با قبل، تعداد *nLearn* از شایسته‌ترین‌ها در هر تکرار در آرایه *history* ذخیره می‌شود و در نتیجه، این آرایه شامل تعداد *iterLearn*×*nLearn* از شایسته‌ترین راه‌حل‌ها خواهد بود، دوباره دامنه جدیدی برای تک‌تک متغیرها شناسایی شده و این روند تا تکرار *maxIterations*-ام انجام می‌شود. در پایان الگوریتم، *C*[1] دارای بیشترین برازندگی خواهد بود و به عنوان بهترین راه‌حل به کاربر اعلام می‌شود.

شکل ۴، شبه‌کد الگوریتم ICCE را نشان می‌دهد. در خط ۱، متغیر *nfeMax* (حداکثر مقدار پیش‌بینی‌شده برای تعداد فراخوانی‌های تابع برازندگی) با عدد تقریبی 300000**dim* مقداردهی اولیه می‌شود که در آن، *dim* به تعداد متغیرها مربوط می‌شود. لازم به ذکر است که این عدد تقریبی به این صورت محاسبه شده است: در هر تکرار الگوریتم، به ازای هر راه‌حل موجود در جمعیت فعلی، تابع *improve()* به تعداد *dim* بار فراخوانی می‌شود. ضمناً، در داخل تابع *improve()* نیز، تابع برازندگی به تعداد ۴ بار فراخوانی می‌شود. از طرفی، الگوریتم با اندازه جمعیت با مقدار اولیه *Ninit* شروع شده و در هر مرحله به صورت خطی کاهش پیدا می‌کند تا نهایتاً به مقدار *Nmin* برسد. برای محاسبه بیشترین مقدار *nfeMax* به صورت تقریبی، اندازه جمعیت را در نیمه نخست تکرارهای الگوریتم (یعنی

(*maxIterations/2*) برابر *Ninit* و در نیمه دیگر، تعداد تکرارها را برابر *Ninit*/2 در نظر می‌گیریم. بنابراین، مقدار تقریبی متغیر *nfeMax* طبق رابطه (۲) محاسبه می‌شود:

$$nfeMax = \frac{maxIterations}{2} * (Ninit * 4 * dim) + \frac{maxIterations}{2} * \left(\frac{Ninit}{2} * 4 * dim\right) = maxIterations * Ninit * 3 * dim \quad (۲)$$

که با فرض حداکثر تعداد تکرارها (*maxIterations*) برابر با ۱۰۰۰ و اندازه جمعیت اولیه (*Ninit*) برابر با ۱۰۰، مقدار متغیر *nfeMax* برابر با *dim**300000 خواهد بود.

Algorithm 1 The ICCE algorithm

Input: *Ninit, Nmin, maxIterations, lb, ub, initIterLearn, iterLearn, nLearn, dim, d;*
Output: A member with the highest fitness;
1: *nfeMax* = 300000**dim*; *nfe* = 0; *N* = *Ninit*;
2: Domain[] *allDoms* = new Domain [*dim*];
3: **for** *i* = *1* **to** *N* **do**
4: *allDoms*[*i*].*lb* = *lb*; *allDoms*[*i*].*ub* = *ub*;
5: **end**
6: Initialize the population of *C* randomly;
7: *fit* = compute (*C*); *nfe* = *nfe* + *N*;
8: *C* = convToMaxHeap (*C*, *fit*); *rep* =1;
9: *history* = *C*[1.. *nLearn*];
10: **while** *rep* <= *maxIterations* **do**
11: **if** *rep* >= *initIterLearn* && *rep* % *iterLearn* == *0* **then**
12: computeNewDomain (*allDoms, history, dim*);
13: clear (*history*);
14: **end**
15: **for** *crit* =1 to *dim* **do**
16: *lb* = *allDoms*[*crit*].*lb*; *ub* = *allDoms*[*crit*].*ub*;
17: *improve* (*C*[1], *findrnd*(*C*[1]), *crit, lb, ub*);
18: *nfe* = *nfe* + 4;
19: **for** *i* =2 to *N* **do**
20: *X* = *C*[*i*]; *Y* = *C*[$floor(\frac{i-2}{d}) + 1$];
21: *lb* = *allDoms*[*crit*].*lb*; *ub* = *allDoms*[*crit*].*ub*;
22: *improve* (*X, Y, crit, lb, ub*); *nfe* = *nfe* + 4;
23: **end**
24: *fit* = compute (*C*); *nfe* = *nfe* + *N*;
25: *C* = convToMaxHeap (*C*, *fit*);
26: Add (*history*, *C*[1.. *nLearn*]););
27: **end**
28: Compute *Nnew* using Formula (1);
29: **if** *Nnew* < *N* **then**
30: *C* = *C*[1..*Nnew*];
31: *N* = *Nnew*;
32: **end**
33: *rep* = *rep* +1;
34: **end**
35: **return** *C*[1];

شکل ۴- شبه‌کد الگوریتم ICCE

در خطوط ۳-۵، مقادیر *lb* و *ub* به‌عنوان دامنهٔ اولیهٔ تمام متغیرها (*allDoms*) در نظر گرفته می‌شوند. بعد از تولید تصادفی جمعیت اولیه در خط ۶، برازندگی آنها با استفاده از تابع compute محاسبه شده و از طریق تابع convToMaxHeap به یک هرم بیشینه تبدیل می‌شود. ضمناً، تعداد *nLearn* از شایسته‌ترین‌های جمعیت اولیهٔ تولید شده، در آرایهٔ *history* ذخیره می‌شوند (خطوط ۷-۹). سپس، با استفاده از تابع computeNewDomain، قسمت‌های امیدبخش دامنهٔ فعلی متغیرها شناسایی شده و آن قسمت‌ها به عنوان دامنهٔ جدید متغیرها در نظر گرفته می‌شوند (خط ۱۲)، در نتیجه، *allDoms* شامل دامنهٔ جدید متغیرها خواهد بود. تابع computeNewDomain دارای پارامترهای ورودیِ *allDoms* به‌عنوان دامنهٔ فعلی متغیرها، *history* به‌عنوان اعضای شایسته‌ی تعداد *initIterLearn* (یا *iterLearn*) مرحله از مراحل قبلی الگوریتم و *dim* به‌عنوان تعداد متغیرها می‌باشد. کارِ اصلی الگوریتم از طریق تکرار حلقهٔ while به تعداد *maxIterations* مرتبه، توسط سه مرحلهٔ زیر انجام می‌شود (خطوط ۱۰-۲۷): ۱- در صورت برقراری شرط (خط ۱۱)، دامنهٔ جدید متغیرها با استفاده از تابع computeNewDomain شناسایی می‌شود. ۲- به ازای هر متغیر موجود، مقدار برازندگی همهٔ اعضاء با استفاده از تابع *improve* بهبود داده شده و با استفاده از تابع convToMaxHeap دومرتبه به یک هرم بیشینه تبدیل می‌شود. ۳- در خط ۲۸، اندازهٔ جمعیت (*Nnew*) برای تکرار بعدی با استفاده از فرمول (۱) محاسبه می‌شود که اگر اندازهٔ آن از اندازهٔ فعلی جمعیت (*N*) کمتر باشد، تعداد *Nnew* از شایسته‌ترین‌های جمعیت فعلی به جمعیت بعدی منتقل می‌شوند. ضمناً، مقدار *N* نیز به مقدار جدید *Nnew* به‌روزرسانی می‌شود (خطوط ۲۹ تا ۳۲). بعد از پایان حلقهٔ while و در خط ۳۵، *C*[1] که دارای بیشترین برازندگی است به عنوان بهترین راه‌حل به کاربر اعلام می‌شود.

۴- نتایج تجربی

در این بخش، کارایی الگوریتم ICCE را از نظر رسیدن به نزدیک‌ترین راه‌حل به جواب بهینه، نرخ موفقیت و سرعت همگرایی مورد بررسی قرار داده و نتایج بدست آمده را با نتایج الگوریتم‌های CCE، SO، BWO، PO، BMO، CHOA و AO مقایسه می‌کنیم. برای این منظور، این الگوریتم‌ها را روی ۲۹ تابع تست متعلق به مسابقات سال ۲۰۱۷ مربوط به کنگره IEEE در زمینه محاسبات تکاملی (CEC 2017) اجرا می‌کنیم. جدول ۱، جزئیات این توابع را نشان می‌دهد. این توابع شامل یک تابع تک-وجهی (F1)، هفت تابع چندوجهی (F3 تا F9)، ده تابع ترکیبی (F10 تا F19) و یازده تابع مرکب (F20 تا F30) است. توابع تک‌وجهی، فقط یک جواب بهینه سراسری داشته و برای بررسی ویژگی انتفاع الگوریتم‌ها مناسب می‌باشند. در حالی که، توابع چندوجهی علاوه بر یک جواب بهینه سراسری دارای چندین جواب

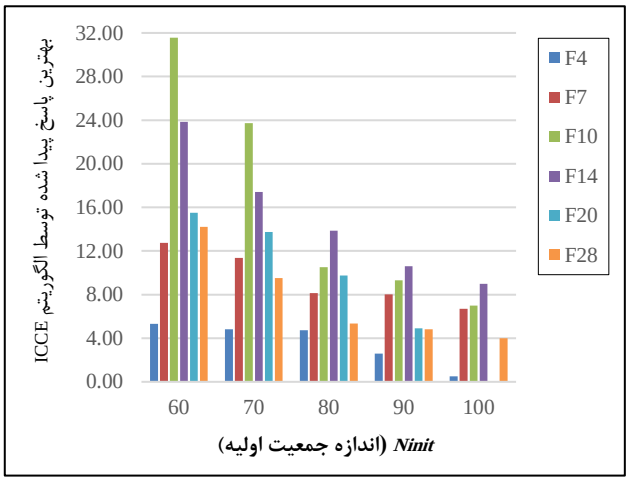
بهینه محلی بوده و جهت بررسی ویژگی اکتشاف و قابلیت اجتناب از بهینگی محلی الگوریتم‌ها مناسب هستند. جهت بدست‌آوردن مقادیر مناسب برای پارامترهای الگوریتم ICCE از روش آزمون و خطا استفاده می‌کنیم. در این روش، به ازای هر پارامتر، چندین مقدار مختلف را در نظر گرفته و الگوریتم را به ازای تک‌تک این مقادیر اجرا می‌کنیم. مقادیری که الگوریتم به ازای آن‌ها بیشترین کارایی را داشته باشد، به عنوان مقادیر مناسب در نظر می‌گیریم. به عنوان مثال، برای انتخاب مقدار مناسب پارامتر *Ninit* (اندازه جمعیت اولیه)، الگوریتم را به ازای مقادیر ۶۰، ۷۰، ۸۰، ۹۰ و ۱۰۰ بر روی توابع چندوجهیِ F4 و F7، ترکیبیِ F10 و F14 و مرکبِ F20 و F28 به تعداد ۳۰ بار اجرا می‌کنیم. شکل ۵، نمودار تأثیر این پارامتر را بر روی بهترین پاسخ پیدا شده توسط الگوریتم ICCE در توابع ذکر شده نشان می‌دهد. لازم به ذکر است که جهت خوانایی بیشتر نمودار، از مقادیر بدست آمده برای تابع F4 مقدار ۴۰۰، برای تابع F7 مقدار ۷۰۰، برای تابع F10 مقدار ۱۰۰۰، برای تابع F14 مقدار ۱۴۰۰، برای تابع F20 مقدار ۲۰۰۰ و برای تابع F28 مقدار ۲۸۰۰ کم شده است (به عبارتی، مقادیر بدست آمده شیفت داده شده‌اند). مطابق با این نمودار، به ازای مقدار *Ninit* = 100، ICCE بهترین پاسخ را تولید می‌کند. جدول ۲، مقادیر مناسب برای این پارامترها را نشان می‌دهد. در این جدول، مقادیر پارامترهای بقیه الگوریتم‌ها نیز دیده می‌شوند که از مقالات مربوطه اقتباس شده‌اند.

جدول ۱- جزئیات توابع تست CEC 2017 با دامنه [-100,100] و ابعاد ۱۰

تابع	نام	مقدار بهینه	نوع
F1	Shifted and Rotated Bent Cigar Function	100	تک‌وجهی
F3	Shifted and Rotated Rosenbrock's Function	300	
F4	Shifted and Rotated Rastrigin's Function	400	
F5	Shifted and Rotated Expanded Scaffer's Function	500	
F6	Shifted and Rotated Lunacek BiRastrigin Function	600	چندوجهی
F7	Shifted and Rotated Non-Continuous Rastrigin's Function	700	
F8	Shifted and Rotated Levy Function	800	
F9	Shifted and Rotated Schwefel's Function	900	
F10	Hybrid Function 1 (N = 3)	1000	
F11	Hybrid Function 2 (N = 3)	1100	
F12	Hybrid Function 3 (N = 3)	1200	ترکیبی (N تعداد توابع پایه را نشان می‌دهد)
F13	Hybrid Function 4 (N = 4)	1300	
F14	Hybrid Function 5 (N = 4)	1400	
F15	Hybrid Function 6 (N = 4)	1500	
F16	Hybrid Function 6 (N = 5)	1600	
F17	Hybrid Function 6 (N = 5)	1700	
F18	Hybrid Function 6 (N = 6)	1800	
F19	Hybrid Function 6 (N = 6)	1900	
F20	Composite Function 1 (N = 3)	2000	
F21	Composite Function 2 (N = 3)	2100	
F22	Composite Function 3 (N = 4)	2200	
F23	Composite Function 4 (N = 4)	2300	مرکب (N تعداد توابع پایه را نشان می‌دهد)
F24	Composite Function 5 (N = 5)	2400	
F25	Composite Function 6 (N = 5)	2500	
F26	Composite Function 7 (N = 6)	2600	
F27	Composite Function 8 (N = 6)	2700	
F28	Composite Function 9 (N = 6)	2800	
F29	Composite Function 10 (N = 3)	2900	
F30	Composite Function 11 (N = 3)	3000	

تعریف ۱: متوسط زمان اجرای (AveTime) یک الگوریتم در *runs* بار اجرا به حاصل تقسیم مجموع مدت زمان‌های سپری شده توسط الگوریتم نسبت به مقدار *runs* گفته می‌شود.

تعریف ۲: سرعت همگرایی معیاری است که نشان می‌دهد کدام الگوریتم می‌تواند زودتر از سایر الگوریتم‌های موجود، نزدیک‌ترین راه‌حل به جواب بهینه را در تعداد تکرار مشخص شده پیدا کند.



شکل ۵- تأثیر پارامتر *Ninit* (اندازهٔ جمعیت اولیه) روی بهترین پاسخ پیدا شده توسط الگوریتم ICCE در توابع چندوجهیِ F4 و F7، ترکیبیِ F10 و F14 و مرکبِ F20 و F28

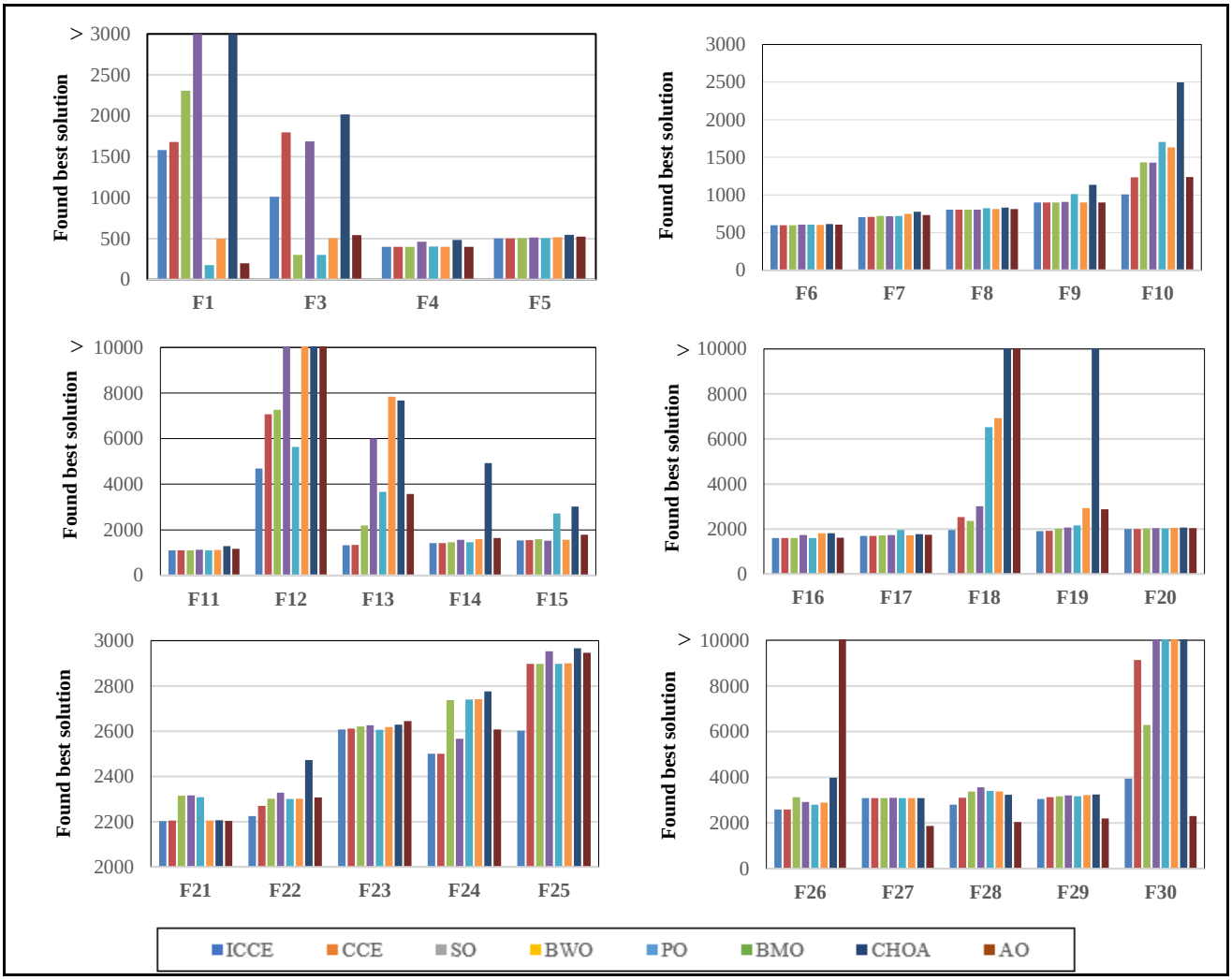
برای تولید نتایج، همه الگوریتم‌ها را به تعداد ۳۰ بار روی هر کدام از توابع تست اجرا می‌کنیم. نتایج، شامل معیار بهترین (Best) مقدار بدست آمده در ۳۰ بار اجرا می‌باشد. علاوه بر این، نتایج شامل میانگین زمان‌های همه ۳۰ بار اجرا نیز هستند. ضمناً، محیط اجرایی این الگوریتم‌ها عبارت است از: نرم‌افزار متلب نسخه ۲۰۱۷ و CPU Intel® Core™ i5 و RAM 6GB.

شکل ۶ نتایج بدست‌آمده را بصورت نمودارهایی نشان می‌دهد. علاوه بر این، خلاصه کلی نتایج که در جدول ۳ آمده است، شامل مقادیر میانگین رتبه (Mean Rank) الگوریتم در ۲۹ تابع، رتبه کلی (Rank) الگوریتم، رتبه الگوریتم از نظر میانگین زمانی (AveTime) و نرخ موفقیت الگوریتم می‌باشد.

جدول ۲- مقادیر مناسب برای پارامترهای الگوریتم‌ها

الگوریتم	پارامترها و مقادیر مناسب
ICCE	*Ninit* = 100, *Nmin* = 30, *d* = 5, *initIterLearn* = 400, *iterLearn* = 200, *nLearn* = 15
CCE	*N* = 30, *d* = 5
CHOA	*f* (i.e., updating strategy) = quadratic, r_1 and r_2: random , *m*: Chaotic , *N* = 50
BWO	*procreate rate* = 0.6, *mutation rate* = 0.4, *cannibalism rate* = 0.44, *N* = 50
PO	*n* (number of parties) = 8, λ (party switching rate) = 1.0 , *N* = 64
BMO	*N* = 50, *pl* = 7
SO	*N* = 30
AO	α = 0.1, δ = 0.1, *N* = 30

برای محاسبه میانگین رتبه هر الگوریتم از آزمون فریدمن استفاده می‌کنیم که یک آزمون فرضیه آماری ناپارامتریک است که می توان از آن برای مقایسه چندگانه نمونه‌های مرتبط استفاده کرد [۴۳]. نرخ موفقیت برای هر الگوریتم نیز نشان می‌دهد که در چنددرصد از توابع، بهترین جواب (Best) برابر با جواب بهینه (طبق جدول ۱) است. مطابق با این جدول، الگوریتم ICCE اولین رتبه را در تولید نزدیک‌ترین راه‌حل به جواب بهینه دارا می‌باشد. همچنین، این الگوریتم با نرخ موفقیت ۱۴٪ دارای بهترین کارایی از نظر تولید جواب‌های برابر با جواب‌های بهینه است. با وجود این برتری‌ها، الگوریتم ICCE دارای متوسط زمان اجرای بالایی در مقایسه با الگوریتم‌های دیگر می‌باشد. دلیل آن، این است که الگوریتم ICCE، مشابه با الگوریتم CCE، باید کارایی هر راه‌حل (عضو شورا) را نسبت به کارایی راه‌حل سطح پایین‌تر (رئیس همان شورا) در همه متغیرها (ملاک‌های کارایی) بهبود ببخشد. همچنین، با توجه به تکنیک کاهش خطی جمعیت (LPSR)، اندازه‌ی جمعیت (*Ninit*) باید به اندازهٔ کافی بزرگ در نظر گرفته شود که این امر باعث می‌شود تعداد ارزیابی‌های تابع تست افزایش پیدا کرده و در نتیجه، زمان اجرای الگوریتم ICCE افزایش پیدا کند.



شکل ۶- نتایج الگوریتم‌ها بر روی توابع تست CEC 2017

جدول ۳-خلاصه‌ای از نتایج الگوریتم‌ها روی توابع تست CEC 2017

		ICCE	CCE	SO	BWO	PO	BMO	CHOA	AO
Friedman Test	**Mean Rank** (Best)	1.69	2.71	4.03	5.59	4.50	5.31	7.45	4.72
	Rank (Best)	**1**	**2**	**3**	**7**	**4**	**6**	**8**	**5**
	Rank (AveTime)	**8**	**7**	**3**	**1**	**4**	**5**	**6**	**2**
Hit Rate (Best)		**14%**	**7%**	**7%**	**0%**	**4%**	**0%**	**0%**	**0%**

در بالا، از آزمون فریدمن برای مقایسه چندگانه رتبه الگوریتم‌ها استفاده کردیم. حال می‌خواهیم از آزمون رتبه علامت‌دار ویلکاکسون جهت مقایسه دوگانه الگوریتم ICCE با تک‌تک الگوریتم‌های دیگر استفاده کنیم. این آزمون یک آزمون فرضیه آماری ناپارامتریک است که برای مقایسه دوگانه نمونه‌های مرتبط کاربرد دارد [۴۴]. جدول ۴، نتایج حاصله از اجرای این آزمون را نشان می‌دهد. در این جدول، سطرهای $R^{-}/R^{+}/R^{=}$ وجود دارند که تعداد توابعی را مشخص می‌کنند که الگوریتم ICCE دارای عملکرد بهتر/بدتر/یکسان نسبت به الگوریتم‌های دیگر است. مطابق با این مقادیر، مقادیر R^{-} خیلی بیشتر از مقادیر R^{+} می‌باشند که نشان می‌دهد الگوریتم ICCE دارای عملکرد خیلی بهتری نسبت به الگوریتم‌های دیگر است. ضمناً، سطر چهارم در این جدول نشان‌دهنده معیار تصمیم (یا همان Asymp. Sig.) است که به مقدار p-value معروف است. چون به‌ازای تمام مقایسه‌ها، مقدار p کمتر از ۰.۰۵ است، لذا می‌توانیم نتیجه بگیریم که تفاوت معناداری بین عملکرد الگوریتم ICCE و بقیه الگوریتم‌ها وجود دارد. دلیل برتر بودن عملکرد الگوریتم ICCE نسبت به CCE و سایر الگوریتم‌ها این است که ICCE از تکنیک‌های کاهش خطی جمعیت (LPSR) و کاهش فضای جستجو (SSR) بهره می‌گیرد که باعث تقویت ویژگی اکتشاف در تکرارهای ابتدایی و تقویت ویژگی انتفاع در تکرارهای پایانی می‌شود.

جدول ۴- نتایج آزمون رتبه علامت‌دار ویلکاکسون

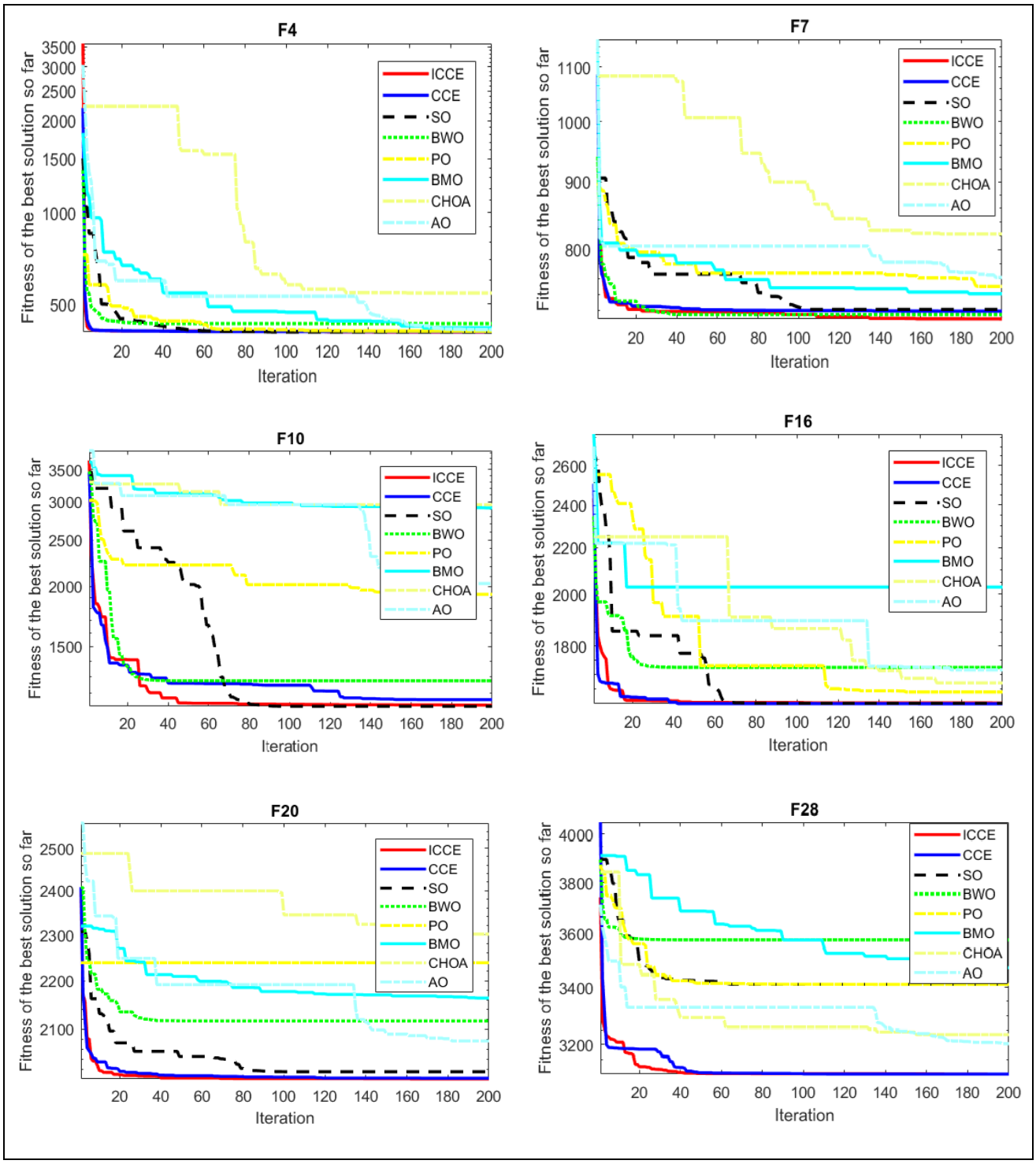
	ICCE – CCE	ICCE – SO	ICCE – BWO	ICCE – PO	ICCE – BMO	ICCE – CHOA	ICCE – AO
R^{-}	23	27	28	25	26	28	23
R^{+}	6	2	1	4	3	1	6
$R^{=}$	0	0	0	0	0	0	0
Asymp. Sig.	$p <$ 0.05	$p <$ 0.05	$p <$ 0.05	$p <$ 0.05	$p <$ 0.05	$p <$ 0.05	$p <$ 0.05

برای مقایسه سرعت همگرایی الگوریتم ICCE با الگوریتم‌های دیگر، آنها را بر روی توابع چندوجهیِ F4 و F7، ترکیبیِ F10 و F14 و مرکبِ F20 و F28 اجرا می‌کنیم. نمودارهای همگرایی موجود در شکل ۷ تأیید می‌کنند که الگوریتم ICCE سرعت همگراییِ بالاتری نسبت به بقیه الگوریتم‌ها دارد. همچنین، می‌توان گفت که الگوریتم ICCE با توجه به داشتن سرعت همگراییِ بالا و اولین رتبه در تولید نزدیک‌ترین راه‌حل به جواب بهینه (مطابق با جدول ۳)، توانایی بالایی در برقراری توازن بین ویژگی‌های اکتشاف و انتفاع و فرار از بهینگی محلی دارد.

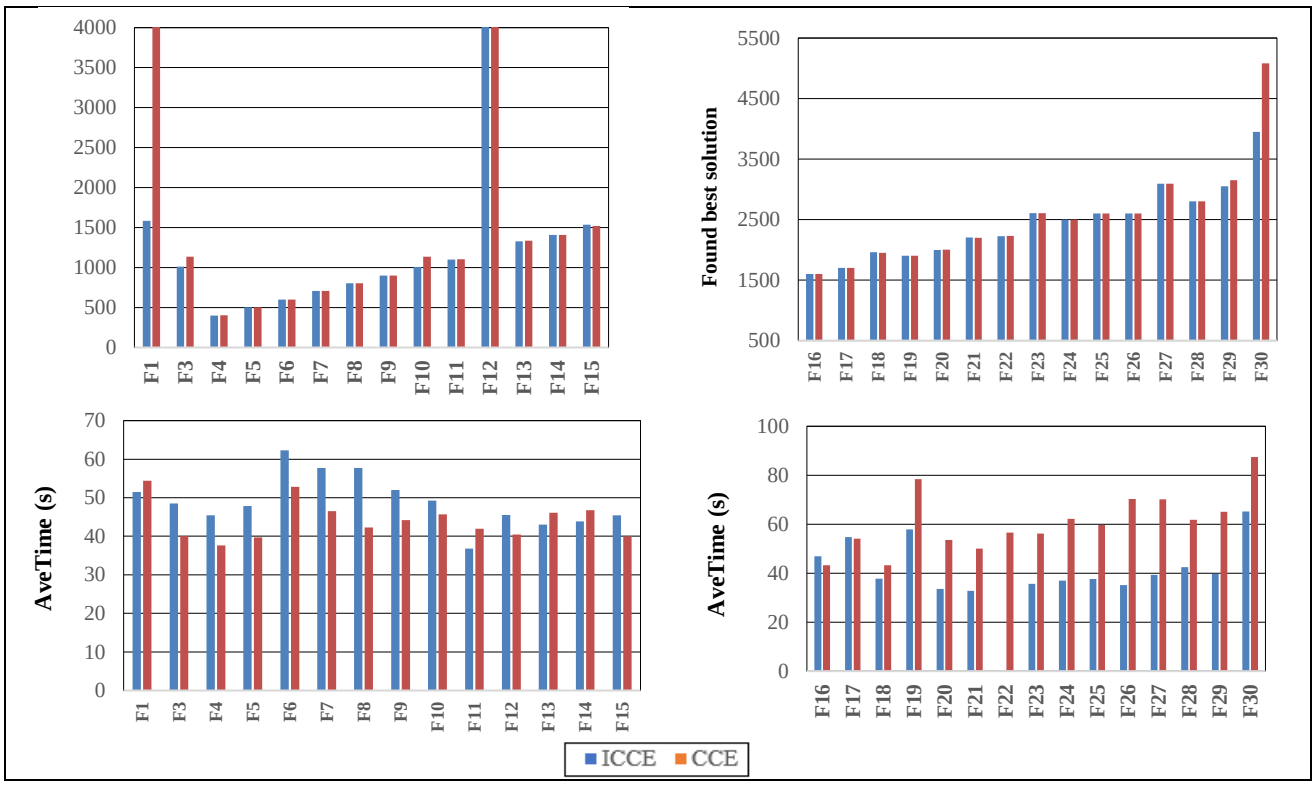
برای مقایسه کارایی دو الگوریتم CCE و ICCE به ازای اندازه جمعیت اولیه یکسان برابر با ۱۰۰، آن‌ها را به تعداد ۳۰ بار بر روی هر کدام از توابع تست اجرا می‌کنیم. نتایج، شامل معیار بهترین (Best) مقدار بدست آمده در ۳۰ بار اجرا و میانگین زمان‌های همه ۳۰ بار اجرا هستند. شکل ۸، نتایج بدست آمده را به صورت نمودار نشان می‌دهد. مطابق با این نمودارها، در اکثر این توابع، الگوریتم ICCE دارای کارایی بهتری از نظر تولید نزدیک‌ترین راه‌حل به جواب بهینه بوده و متوسط زمان اجرای پایینی دارد.

۴-۱- مسائل بهینه‌سازی مهندسی محدودیت‌دار

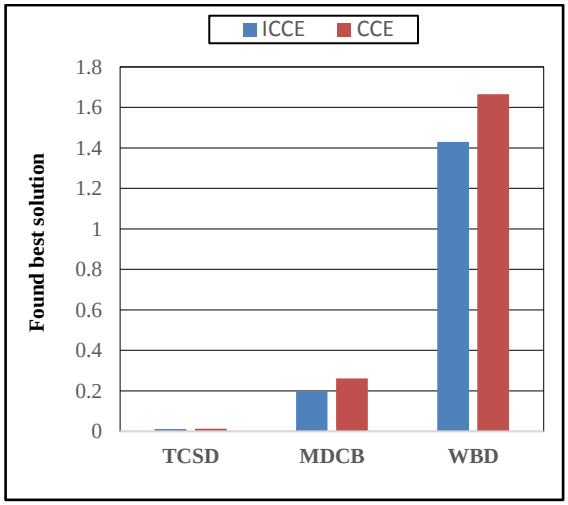
در این بخش، کارایی الگوریتم‌های ICCE و CCE بر روی سه مسأله بهینه‌سازی مهندسی محدودیت‌دار به صورت زیر مقایسه می‌شود: **(۱)** مسأله طراحی فنر کششی/فشاری (TCSD) که هدف آن، طراحی یک فنر کششی/فشاری با وزن بهینه است. این مسأله دارای محدودیت‌هایی در حداقل انحراف، تنش برشی، فرکانس موج و غیره است [۴۵]، **(۲)** مسأله طراحی کلاچ دیسکی چندگانه (MDCB) که هدف این مسأله طراحی ترمز کلاچ دیسکی چندگانه با حداقل جِرم است [۴۶]، و **(۳)** مسأله طراحی تیر جوش داده شده (WBD) است که هدف مسأله، طراحی تیر جوشی با حداقل هزینه است. این الگوریتم‌ها را به تعداد ۳۰ بار بر روی این مسائل اجرا می‌کنیم. شکل ۹، نتایج این الگوریتم‌ها را به صورت نمودارهایی برای سه مسأله مذکور نشان می‌دهد. مطابق با این نمودارها، الگوریتم ICCE می‌تواند راه‌حل‌های بهتری نسبت به الگوریتم CCE تولید کند.



شکل ۷- نمودارهای همگرایی همه الگوریتم‌ها در توابع چندوجهیِ F4 و F7، ترکیبیِ F10 و F14 و مرکبِ F20 و F28



شکل ۸- نتایج اجرای الگوریتم‌های ICCE و CCE با جمعیت اولیه یکسان ۱۰۰ بر روی توابع تست CEC 2017



شکل ۹- نتایج الگوریتم‌های ICCE و CCE روی مسائل بهینه‌سازی مهندسی محدودیت‌دار

۵- نتیجه‌گیری و کارهای آتی

در این مقاله، نسخهٔ بهبودیافته‌ای از الگوریتم تکامل شوراهای شهر (CCE) به‌نام ICCE را جهت حل مسائل بهینه‌سازی ارائه کردیم. برای این منظور، کارایی الگوریتم CCE با به کارگیری دو تغییر مهم بهبود داده شد: کاهش خطی اندازه جمعیت (LPSR) و کاهش فضای جستجو (SSR). استفاده از LPSR نه تنها قابلیت جستجوی سراسری را بهبود می‌بخشد، بلکه باعث ایجاد توازن بین ویژگی‌های اکتشاف و انتفاع می‌شود. به‌علاوه، SSR باعث افزایش کیفیت راه‌حل‌های بهینه هم می‌شود.

برای تحلیل و مقایسهٔ کارایی الگوریتم ICCE با کارایی الگوریتم‌های CCE، CHOA، BWO، PO، BMO، SO و AO از نظر یافتن نزدیک‌ترین راه‌حل به جواب بهینه و نرخ موفقیت، آن‌ها را بر روی ۲۹ تابع تست از CEC 2017 و به تعداد ۳۰ بار اجرا کرده‌ایم. نتایج آزمون‌های میانگین رتبهٔ فریدمن و رتبهٔ علامت‌دار ویلکاکسون تأیید می‌کنند که الگوریتم ICCE اولین رتبه را در تولید نزدیک‌ترین راه‌حل به جواب بهینه، نسبت به الگوریتم‌های دیگر دارا می‌باشد. همچنین، این الگوریتم با نرخ موفقیت ٪۱۴ دارای بهترین کارایی از نظر تولید جواب‌های برابر با جواب‌های بهینه است. همچنین، برای مقایسه سرعت همگرایی الگوریتم ICCE با الگوریتم‌های دیگر، آنها را بر روی توابع چندوجهیِ F4 و F7، ترکیبیِ F10 و F14 و مرکبِ F20 و F28 اجرا کرده‌ایم. نمودارهای همگرایی حاصل، سرعت همگرایی بالای این الگوریتم را نسبت به سایر الگوریتم‌ها نشان می‌دهند. با وجود این برتری‌ها، الگوریتم ICCE دارای متوسط زمان اجرای بالایی در مقایسه با الگوریتم‌های دیگر می‌باشد. بهبود زمان اجرا و تغییر عملکرد

الگوریتم ICCE برای حل مسائل بهینه‌سازی چندهدفه و مقید می‌توانند به‌عنوان کارهای آتی در نظر گرفته شوند.

مراجع

[1] E. Pira, "City councils evolution: a socio-inspired metaheuristic optimization algorithm," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1-50, 2022.

[2] S. Kundu and D. R. Parhi, "Navigation of underwater robot based on dynamically adaptive harmony search algorithm," *Memetic Computing*, vol. 8, no. 2, pp. 125-146, 2016.

[3] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 127-177, 2010.

[4] E. Pira, "A novel approach to solve AI planning problems in graph transformations," *Engineering Applications of Artificial Intelligence*, vol. 92, p. 103684, 2020.

[5] E. Pira, "Using deep learning techniques for solving AI planning problems specified through graph transformations," *Soft Computing*, pp. 1-18, 2022.

[6] O. Ozkan, M. Ermis, and I. Bekmezci, "Reliable communication network design: The hybridisation of metaheuristics with the branch and bound method," *Journal of the Operational Research Society*, vol. 71, no. 5, pp. 784-799, 2020.

[7] J. W. Zhang and G. G. Wang, "Image matching using a bat algorithm with mutation," in *Applied mechanics and materials*, 2012, vol. 203: Trans Tech Publ, pp. 88-93.

[8] E. Pira, "Using Markov Chain Based Estimation of Distribution Algorithm for Model-Based Safety Analysis of Graph Transformation," *Journal of Computer Science and Technology*, vol. 36, no. 4, pp. 839-855, 2021.

[9] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46-61, 2014.

[10] E. Alba and B. Dorronsoro, "The exploration/exploitation tradeoff in dynamic cellular genetic algorithms," *IEEE transactions on evolutionary computation*, vol. 9, no. 2, pp. 126-142, 2005.

[11] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671-680, 1983.

[12] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67-82, 1997.

[13] J. L. J. Laredo, C. Fernandes, J. J. Merelo, and C. Gagné, "Improving genetic algorithms performance via deterministic population shrinkage," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009, pp. 819-826.

[14] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66-73, 1992.

[15] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary computation*, vol. 3, no. 2, pp. 82-102, 1999.

[16] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341-359, 1997.

[17] B. Webster and P. J. Bernhard, "A local search optimization algorithm based on natural principles of gravitation," 2003.

[18] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, no. 1, pp. 33-57, 2007.

[19] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28-39, 2006.

[20] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm," *Journal of global optimization*, vol. 39, no. 3, pp. 459-471, 2007.

[21] X.-S. Yang, "Flower pollination algorithm for global optimization," in *International conference on unconventional computing and natural computation*, 2012: Springer, pp. 240-249.

[22] A. Askarzadeh, "A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm," *Computers & Structures*, vol. 169, pp. 1-12, 2016.

[23] M. Khishe and M. R. Mosavi, "Chimp optimization algorithm," *Expert systems with applications*, vol. 149, p. 113338, 2020.

[24] V. Hayyolalam and A. A. P. Kazem, "Black widow optimization algorithm: a novel meta-heuristic approach for solving engineering optimization problems," *Engineering Applications of Artificial Intelligence*, vol. 87, p. 103249, 2020.

[25] M. H. Sulaiman, Z. Mustaffa, M. M. Saari, and H. Daniyal, "Barnacles mating optimizer: a new bio-inspired algorithm for solving engineering optimization problems," *Engineering Applications of Artificial Intelligence*, vol. 87, p. 103330, 2020.

[26] F. A. Hashim and A. G. Hussien, "Snake Optimizer: A novel meta-heuristic optimization algorithm," *Knowledge-Based Systems*, vol. 242, p. 108320, 2022.

[27] L. Abualigah, D. Yousri, M. Abd Elaziz, A. A. Ewees, M. A. Al-Qaness, and A. H. Gandomi, "Aquila optimizer: a novel meta-heuristic optimization algorithm," *Computers & Industrial Engineering*, vol. 157, p. 107250, 2021.

[28] S. Saremi, S. Mirjalili, and A. Lewis, "Grasshopper optimisation algorithm: theory and application," *Advances in Engineering Software*, vol. 105, pp. 30-47, 2017.

[29] J. J. Flores, R. López, and J. Barrera, "Gravitational interactions optimization," in *International Conference on Learning and Intelligent Optimization*, 2011: Springer, pp. 226-237.

[30] A. H. Kashan, "A new metaheuristic for optimization: optics inspired optimization (OIO)," *Computers & Operations Research*, vol. 55, pp. 99-125, 2015.

[31] S. Kumar, D. Datta, and S. K. Singh, "Black hole algorithm and its applications," in *Computational intelligence applications in modeling and control*: Springer, 2015, pp. 147-170.

[32] H. Eskandar, A. Sadollah, A. Bahreininejad, and M. Hamdi, "Water cycle algorithm–A novel metaheuristic optimization method for solving constrained engineering optimization problems," *Computers & Structures*, vol. 110, pp. 151-166, 2012.

[33] I. Ahmadianfar, O. Bozorg-Haddad, and X. Chu, "Gradient-based optimizer: A new metaheuristic optimization algorithm," *Information Sciences*, vol. 540, pp. 131-159, 2020.

[34] R. V. Rao, V. J. Savsani, and D. Vakharia, "Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems," *Computer-aided design*, vol. 43, no. 3, pp. 303-315, 2011.

[35] S. Satapathy and A. Naik, "Social group optimization (SGO): a new population evolutionary optimization technique," *Complex & Intelligent Systems*, vol. 2, no. 3, pp. 173-203, 2016.

[36] A. Faramarzi, M. Heidarinejad, B. Stephens, and S. Mirjalili, "Equilibrium optimizer: A novel optimization algorithm," *Knowledge-Based Systems*, vol. 191, p. 105190, 2020.

[37] Q. Askari, I. Younas, and M. Saeed, "Political Optimizer: A novel socio-inspired meta-heuristic for global optimization," *Knowledge-based systems*, vol. 195, p. 105709, 2020.

[38] I. Naruei and F. Keynia, "Wild horse optimizer: A new meta-heuristic algorithm for solving engineering optimization problems," *Engineering with Computers*, pp. 1-32, 2021.

[39] J. L. Melvix, "Greedy politics optimization: Metaheuristic inspired by political strategies adopted during state assembly elections," in *2014 IEEE international advance computing conference (IACC)*, 2014: IEEE, pp. 1157-1162.

[40] W. Lv, C. He, D. Li, S. Cheng, S. Luo, and X. Zhang, "Election campaign optimization algorithm," *Procedia Computer Science*, vol. 1, no. 1, pp. 1377-1386, 2010.

[41] A. Borji, "A new global optimization algorithm inspired by parliamentary political competitions," in *Mexican international conference on artificial intelligence*, 2007: Springer, pp. 61-71.

[42] S. H. S. Moosavi and V. K. Bardsiri, "Poor and rich optimization algorithm: A new human-based and multi populations algorithm," *Engineering Applications of Artificial Intelligence*, vol. 86, pp. 165-181, 2019.

[43] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86-92, 1940.

[44] R. F. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1-3, 2007.

[45] X. He and Y. Zhou, "Enhancing the performance of differential evolution with covariance matrix self-adaptation," *Applied Soft Computing*, vol. 64, pp. 227-243, 2018.

[46] A. Osyczka and A. Osyczka, *Evolutionary algorithms for single and multicriteria design optimization*. Springer, 2002.

پاورقی‌ها:

[1] Metaheuristic algorithms

[2] Simplicity

[3] Flexibility

[4] Local optima avoidance

[5] Gradient-free

[6] Exploration

[7] Exploitation

[8] Simulated Annealing Algorithm

[9] No Free Lunch theorem

[10] Linear Population Size Reduction (LPSR)

[11] Search Space Reduction (SSR)

[12] Black Hole Algorithm

[13] Water Cycle Algorithm

[14] Gradient-Based Optimizer

[15] Teaching-Learning-Based Optimization

[16] Parliamentary Optimization Algorithm

[17] Poor and Rich Optimization