



دانشکده مهندسی مکانیک

مقایسه عملکرد الگوریتم‌های هوشمند جستجوی فاخته، ماهی الکتریکی، کرم شبتاب، امپریالیست، غذایابی باکتریایی در طراحی مسیر ربات

پروژه کارشناسی مهندسی مکانیک

نام دانشجو

پرهام پرخیال

استاد راهنما:

دکتر اسماعیل خانمیرزا

شهریور ۱۴۰۳

بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِيْمِ

تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پایان‌نامه/رساله

نام دانشکده: دانشکده مکانیک علم و صنعت ایران

نام دانشجو: پرهام پرخیال

عنوان پایان‌نامه یا رساله: مقایسه عملکرد الگوریتم‌های هوشمند جستجوی فاخته، ماهی الکتریکی، کرم شبتاب، امپریالیست،

غذایابی باکتریابی در طراحی مسیر ربات

تاریخ دفاع:

رشته: مهندسی مکانیک

گرایش: مکاترونیک

ردیف	سمت	نام و نام خانوادگی	مرتبه	دانشگاه یا مؤسسه	امضا
۱	استاد راهنما	اسماعیل خان‌میرزا	دانشیار	دانشگاه علم و صنعت	
۲	استاد راهنما				
۳	استاد مشاور				
۴	استاد مشاور				
۵	استاد مدعو خارجی				
۶	استاد مدعو خارجی				
۷	استاد مدعو داخلی	برهان بیگ‌زاده	دانشیار	دانشگاه علم و صنعت	
۸	استاد مدعو داخلی				

تأییدیه‌ی صحت و اصالت نتایج

با اسمه تعالی

اینجانب پرهام پرخیال به شماره دانشجویی ۹۷۵۴۳۲۷۸ دانشجوی رشته مهندسی مکانیک مقطع تحصیلی کارشناسی تأیید می‌نمایم که کلیه‌ی نتایج این پژوهه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری شده از آثار دیگران را با ذکر کامل مشخصات منع ذکر کرده‌ام. در صورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انتساباتی ...) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احراق حقوق مکتب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی‌صلاح (اعم از اداری و قضایی) به عهده‌ی اینجانب خواهد بود و دانشگاه هیچ‌گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: پرهام پرخیال

امضا و تاریخ:

مجوز بهرهبرداری از پایاننامه

بهرهبرداری از این پایاننامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنمای شرح زیر تعیین میشود، بلامانع است:

- بهرهبرداری از این پایاننامه/ رساله برای همگان بلامانع است.
- بهرهبرداری از این پایاننامه/ رساله با اخذ مجوز از استاد راهنمای، بلامانع است.
- بهرهبرداری از این پایاننامه/ رساله تا تاریخ ممنوع است.

نام استاد یا استاد راهنمای: دکتر اسماعیل خانمیرزا

تاریخ:

امضا:

تشکر و قدردانی:

با سپاس فراوان از استاد گرانقدر جناب آقای دکتر خانمیرزا، که با دانش و تجربه‌ی ارزشمند خود همواره راهنمای من در مسیر انجام این تحقیق بودند. حمایت‌ها و راهنمایی‌های بی‌دریغ ایشان نقش بسیار مؤثری در پیشرفت و تکمیل این پایان‌نامه داشته است. همچنین از جناب آقای دکتر علیرضا ملکی، دانشجوی دکتری و مدیر آزمایشگاه آیداس، نهایت تشکر و قدردانی را دارم. ایشان با مشاوره‌ها و همکاری‌های صمیمانه خود در تمامی مراحل انجام این پژوهش، راه‌گشای بسیاری از مسائل بودند. از تلاش‌ها و زحمات شما بی‌نهایت سپاس‌گزارم.

چکیده

الگوریتم‌های بهینه‌سازی به دو گروه کلی تقسیم می‌شوند: الگوریتم‌های دقیق و الگوریتم‌های تقریبی. الگوریتم‌های دقیق، جواب‌های کاملاً صحیح برای مسائل ارائه می‌دهند، اما برای مسائل پیچیده، نیازمند زمان بسیار زیادی هستند. در مقابل، الگوریتم‌های تقریبی با صرف زمان کمتر، جواب‌هایی نزدیک به بهینه ارائه می‌دهند و برای مسائل پیچیده مناسب‌تر هستند. در این میان، الگوریتم‌های فرآبتكاری، دسته‌ای از الگوریتم‌های تقریبی مبتنی بر شناس هستند که برای یافتن جواب‌های بهینه در مسائل مختلف به کار گرفته می‌شوند. این الگوریتم‌ها توانایی جستجوی گسترده‌تر و غلبه بر بهینه‌های محلی را دارند. الگوریتم‌های فرآبتكاری به دو دسته عمده تقسیم می‌شوند: دسته‌ای که بر اساس مسیر حرکت می‌کنند و دسته‌ای که بر اساس تعاملات جمعیتی طراحی شده‌اند. در الگوریتم‌های مسیرمحور، جستجو به طور تدریجی انجام می‌شود و مسیر بهینه‌سازی تغییر می‌کند. در حالی که در الگوریتم‌های جمعیت‌محور، مجموعه‌ای از جواب‌ها به طور همزمان مورد بررسی قرار می‌گیرند. نمونه‌هایی از این الگوریتم‌ها شامل الگوریتم ماهی الکتریکی و الگوریتم کرم شبتاب است. برای ارزیابی و مقایسه این الگوریتم‌ها، معمولاً از تست‌های مختلف استفاده می‌شود که معیارهایی مانند سرعت همگرایی، دقت، و عملکرد کلی الگوریتم را می‌سنجند. یکی از این معیارها، تابع شانه تخم مرغی است که برای بررسی توانایی الگوریتم‌ها در یافتن بهینه‌های چندگانه و اجتناب از کمینه‌های محلی استفاده می‌شود. در این پایان‌نامه، روش‌های جدیدی برای بهینه‌سازی فرآبتكاری در مسئله مسیریابی ریات‌های متحرک معرفی می‌شود. برای این منظور، چند الگوریتم جمعیت‌محور به تفصیل مورد بحث قرار گرفته و مدل‌سازی این الگوریتم‌ها با استفاده از نرم‌افزار MATLAB انجام می‌شود. سپس عملکرد هر الگوریتم با استفاده از تابع شانه تخم مرغی ارزیابی و نتایج به دست آمده با هم مقایسه و رتبه‌بندی می‌شوند تا بهترین الگوریتم برای مسیریابی ربات مشخص گردد.

واژگان کلیدی: بهینه‌سازی فرآبتكاری، الگوریتم‌های مبتنی بر جمعیت، مسیریابی ربات، تابع شانه تخم مرغی

فهرست مطالب

فصل ۱: مقدمه

۱	۱- مقدمه.....
۲	۱- ۲- مسیریابی ربات.....
۳	۱- ۲- جنبه‌های کلیدی مسیریابی ربات.....
۴	۱- ۲- تاریخچه پیشرفت علم مسیریابی ربات.....
۶	۱- ۲- علت استفاده از مسیریابی ربات.....
۷	۱- ۳- بهینه‌سازی.....
۷	۱- ۳- ۱- طبقه بندی براساس استفاده از مشتق یا گرادیان.....
۸	۱- ۳- ۲- طبقه بندی براساس ساختار جستجو.....
۹	۱- ۳- ۳- طبقه بندی براساس قطعیت یا تصادفی بودن.....
۹	۱- ۳- ۴- طبقه بندی براساس جستجوی محلی یا سراسری.....
۹	۱- ۴- الگوریتم‌های فرآبتكاری.....
۱۰	۱- ۴- ۱- ویژگی اصلی ساختار الگوریتم‌های فرآبتكاری.....
۱۱	۱- ۴- ۲- تاریخچه استفاده از الگوریتم‌های فرآبتكاری.....
۱۳	۱- ۴- ۳- انواع الگوریتم‌های فرآبتكاری.....
۱۳	۱- ۴- ۳- ۱- الگوریتم‌های مبتنی بر جمعیت.....
۱۴	۱- ۴- ۳- ۲- الگوریتم‌های مبتنی بر جستجوی محلی.....
۱۴	۱- ۴- ۴- چرا باید از الگوریتم‌های فرآبتكاری استفاده کنیم.....

فصل ۲: الگوریتم‌های فرآبتكاری در بهینه‌سازی

۱۷	۲- ۱- تئوری و روند الگوریتم‌ها.....
۱۸	۲- ۱- ۱- الگوریتم بهینه‌سازی غذایابی باکتریایی.....
۱۸	۲- ۱- ۱- ۱- الگوریتم.....
۲۰	۲- ۱- ۱- ۲- مدل‌سازی الگوریتم با ورودی تصادفی.....
۲۳	۲- ۱- ۱- ۳- مدل‌سازی الگوریتم با ورودی از قبل تعیین شده.....
۲۵	۲- ۱- ۲- الگوریتم بهینه‌سازی کرم شب‌تاب.....
۲۶	۲- ۱- ۲- ۱- الگوریتم.....
۲۷	۲- ۱- ۲- ۲- مدل‌سازی الگوریتم با ورودی تصادفی.....
۲۹	۲- ۱- ۲- ۳- مدل‌سازی الگوریتم با ورودی از قبل تعیین شده.....
۳۲	۲- ۱- ۳- الگوریتم جستجوی فاخته.....
۳۳	

۱- ۳- رفتار فاخته.....	۲
۳۳	
۲- ۳- پرواز لوى.....	۲
۳۴	
۱- ۲- ویژگی های توزیع لوى.....	۲
۳۴	
۲- ۳- الگوریتم.....	۲
۳۵	
۱- ۲- پرواز لوى.....	۲
۳۶	
۲- ۳- مدل سازی با ورودی تصادفی.....	۲
۳۸	
۱- ۲- مدل سازی الگوریتم با ورودی از قبل تعیین شده.....	۲
۴۲	
۲- ۴- الگوریتم امپریالیست رقابتی.....	۲
۴۲	
۱- ۴- ایجاد امپراطوری های اولیه.....	۲
۴۳	
۲- ۴- حرکت مستعمرات به سمت امپراطور خود.....	۲
۴۴	
۱- ۴- ۳- انقلاب.....	۲
۴۵	
۲- ۴- ۴- رقابت درون امپراطوری.....	۲
۴۵	
۱- ۴- ۵- قدرت کل یک امپراطوری.....	۲
۴۵	
۱- ۴- ۶- رقابت امپریالیستی.....	۲
۴۶	
۱- ۴- ۷- همگرایی.....	۲
۴۶	
۱- ۴- ۸- مدل سازی با ورودی تصادفی.....	۲
۴۷	
۱- ۴- ۹- مدل سازی الگوریتم با ورودی از قبل تعیین شده.....	۲
۵۲	
۱- ۵- بهینه سازی ماهی الکتریکی.....	۲
۵۴	
۱- ۵- ۱- الگوریتم.....	۲
۵۵	
۱- ۵- ۲- مکانیابی الکتریکی فعال.....	۲
۵۶	
۱- ۵- ۳- مکانیابی الکتریکی غیر فعال.....	۲
۵۸	
۱- ۵- ۴- مدل سازی با ورودی تصادفی.....	۲
۶۰	
۱- ۵- ۵- مدل سازی الگوریتم با ورودی از قبل تعیین شده.....	۲
۶۵	
۱- ۶- تحلیل نتایج حاصل از استفاده از ورودی های تصادفی.....	۲
۶۶	
۱- ۲- ۱- الگوریتم بهینه سازی غذایابی باکتریایی.....	۲
۶۶	
۱- ۲- ۲- الگوریتم بهینه سازی کرم شب تاب.....	۲
۶۹	
۱- ۲- ۳- الگوریتم جستجوی فاخته.....	۲
۷۱	
۱- ۲- ۴- الگوریتم امپریالیست رقابتی.....	۲
۷۳	
۱- ۲- ۵- الگوریتم ماهی الکتریکی.....	۲
۷۵	
۱- ۲- ۶- جمع بندی.....	۲
۷۷	
۱- ۳- تحلیل نتایج حاصل از استفاده از ورودی های غیر تصادفی.....	۲
۷۸	
۱- ۳- ۱- الگوریتم غذایابی باکتریایی.....	۲
۷۹	

۸۳	۲-۳-۲ الگوریتم جستجوی فاخته.....
۸۶	۳-۳-۲ الگوریتم ماهی الکتریکی.....
۸۹	۲-۴ الگوریتم کرم شب تاب.....
۹۳	۲-۵ الگوریتم امپریالیست رقابتی.....
۹۶	۲-۶ جمع‌بندی.....

۱۰۲	فصل ۳: استفاده از الگوریتم‌های فراابتکاری در مسیریابی ربات
۱۰۳	۱-۳ مقدمه.....
۱۰۳	۳-۲ مدل‌سازی مسیریابی ربات.....
۱۰۴	۳-۲-۱ توابع مشابه در پیاده‌سازی الگوریتم‌ها.....
۱۰۴	۳-۲-۲-۱ ایجاد فایل پارامترهای ثابت.....
۱۰۵	۳-۲-۲-۲ تابع هزینه.....
۱۰۷	۳-۲-۳ نحوه‌ی نمایش نتایج.....
۱۰۹	۳-۲-۲-۲-۱ پیاده‌سازی بهینه‌سازی مسیر ربات در محیط متلب به‌تفکیک الگوریتم‌ها.....
۱۱۳	۳-۲-۲-۲-۲ الگوریتم کرم شب تاب.....
۱۱۷	۳-۲-۲-۳ الگوریتم امپریالیست رقابتی.....
۱۲۵	۳-۲-۳-۱ الگوریتم جستجوی فاخته.....
۱۳۰	۳-۲-۳-۲ الگوریتم ماهی الکتریکی.....
۱۳۷	۳-۳ نتایج.....
۱۳۷	۳-۳-۱ تحلیل نتایج با صد تکرار.....
۱۴۰	۳-۳-۲ تحلیل نتایج با بیست تکرار.....
۱۴۳	۳-۳-۳ جمع‌بندی.....

۱۴۵

مراجع

۱۴۷

پیوست‌ها

۱۴۸	پیوست آ: توابع تست برای بهینه‌سازی تک هدفه.....
۱۴۸	آ-۱ تابع شانه تخم مرغی.....
۱۵۱	پیوست ب: الگوریتم‌ها.....
۱۵۱	ب-۱ الگوریتم کرم شب تاب.....
۱۵۲	ب-۲ الگوریتم جستجوی فاخته.....
۱۵۳	ب-۳ الگوریتم امپریالیست رقابتی.....

ب-۴ الگوریتم ماهی الکتریکی	۱۵۴
ب-۵ الگوریتم غذایابی باکتریایی	۱۵۵
پیوست پ: موقعیت ذرات در بررسی الگوریتم با ورودی‌های تصادفی	۱۵۷
پ-۱ بهینه‌سازی غذایابی باکتریایی	۱۵۷
پ-۲ جستجوی فاخته	۱۵۸
پ-۳ ماهی الکتریکی	۱۵۹
پ-۴ بهینه‌سازی کرم شبتاب	۱۶۰
پ-۵ بهینه‌سازی امپریالیست رقابتی	۱۶۱
پیوست ج: موقعیت ذرات در بررسی الگوریتم با ورودی‌های تعیین شده	۱۶۲
ج-۱ بهینه‌سازی غذایابی باکتریایی	۱۶۲
ج-۲ بهینه‌سازی جستجوی فاخته	۱۶۴
ج-۳ بهینه‌سازی ماهی الکتریکی	۱۶۵
ج-۴ بهینه‌سازی کرم شبتاب	۱۶۶
ج-۵ بهینه‌سازی امپریالیست رقابتی	۱۶۸
پیوست ژ	۱۷۰
ژ-۱ نتایج با صد تکرار	۱۷۰
ژ-۲ نتایج با بیست تکرار	۱۷۱

فهرست اشکال

شکل ۲ - ۱: فلوچارت غذایابی باکتریابی.....	۲۲
شکل ۲ - ۲: فلوچارت الگوریتم بهینه‌سازی کرم شب تاب	۳۰
شکل ۲ - ۳: فلوچارت الگوریتم بهینه‌سازی جستجوی فاخته.....	۴۱
شکل ۲ - ۴: حرکت مستعمرات به سمت امپریالیست مربوطه خود.....	۴۴
شکل ۲ - ۵: حرکت مستعمرات به سمت امپریالیست مربوطه خود در جهت تصادفی انحرافی.....	۴۵
شکل ۲ - ۶: فلوچارت الگوریتم امپریالیست رقابتی طبق الگوریتم(۳).....	۴۷
شکل ۲ - ۷: نمایش جستجوی محلی EFO. فردی که در حالت فعل است بسته به وجود همسایه در نزدیکی خود.....	۵۸
شکل ۲ - ۸: نمایش جستجوی جهانی EFO	۵۸
شکل ۲ - ۹: فلوچارت الگوریتم بهینه‌سازی ماهی الکتریکی طبق پیوست(ج-۴).....	۶۱
شکل ۲ - ۱۰: نمودار تغییرات جواب بهینه براساس تعداد تکرار حلقه.....	۶۸
شکل ۲ - ۱۱: جایگیری باکتری‌ها در موقعیت اولیه و نهایی.....	۶۸
شکل ۲ - ۱۲: نمودار تغییرات جواب بهینه بر حسب تعداد تکرار چرخه.....	۷۰
شکل ۲ - ۱۳: جایگیری کرم‌های شب تاب در حالت اولیه و نهایی در نمایش ۲ بعدی	۷۰
شکل ۲ - ۱۴: نمودار تغییرات جواب بهینه بر حسب تعداد تکرار چرخه.....	۷۲
شکل ۲ - ۱۵: جایگیری لانه‌ها در حالت اولیه و نهایی در کانتور دو بعدی	۷۲
شکل ۲ - ۱۶: نمودار تغییرات جواب بهینه بر حسب تعداد تکرار چرخه.....	۷۴
شکل ۲ - ۱۷: جایگیری کلونی‌ها در حالت اولیه و نهایی در کانتور دو بعدی	۷۵
شکل ۲ - ۱۸: نمودار تغییرات جواب بهینه بر حسب تعداد تکرار چرخه.....	۷۶
شکل ۲ - ۱۹: جایگیری ماهی‌ها در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی	۷۷
شکل ۲ - ۲۰: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم غذایابی باکتریابی	۸۰
شکل ۲ - ۲۱: نمودار زمان اجرای برنامه در هر اجرای الگوریتم غذایابی باکتریابی	۸۰
شکل ۲ - ۲۲: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم غذایابی باکتریابی	۸۱
شکل ۲ - ۲۳: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه	۸۲
شکل ۲ - ۲۴: جایگیری باکتری‌ها در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی	۸۲
شکل ۲ - ۲۵: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم جستجوی فاخته	۸۳
شکل ۲ - ۲۶: نمودار زمان اجرای برنامه در هر اجرای الگوریتم جستجوی فاخته	۸۴
شکل ۲ - ۲۷: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم جستجوی فاخته	۸۴
شکل ۲ - ۲۸: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه	۸۵
شکل ۲ - ۲۹: جایگیری ذرات در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی	۸۵
شکل ۲ - ۳۰: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم ماهی الکتریکی	۸۷
شکل ۲ - ۳۱: نمودار زمان اجرای برنامه در هر اجرای الگوریتم ماهی الکتریکی	۸۷
شکل ۲ - ۳۲: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم ماهی الکتریکی ...	۸۸

شکل ۲ - ۳۳: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه	۸۸
شکل ۲ - ۳۴: جایگیری ذرات در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی	۸۹
شکل ۲ - ۳۵: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم کرم شبتاب	۹۰
شکل ۲ - ۳۶: نمودار زمان اجرای برنامه در هر اجرای الگوریتم کرم شبتاب	۹۱
شکل ۲ - ۳۷: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم کرم شبتاب	۹۱
شکل ۲ - ۳۸: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه کرم شبتاب	۹۲
شکل ۲ - ۳۹: جایگیری ذرات در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی	۹۲
شکل ۲ - ۴۰: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم امپریالیست رقابتی	۹۴
شکل ۲ - ۴۱: نمودار زمان اجرای برنامه در هر اجرای الگوریتم امپریالیست رقابتی	۹۴
شکل ۲ - ۴۲: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم امپریالیست رقابتی	۹۵
شکل ۲ - ۴۳: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه الگوریتم امپریالیست رقابتی	۹۵
شکل ۲ - ۴۴: جایگیری ذرات در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی	۹۶
شکل ۲ - ۴۵: مقایسه تعداد تکرار چرخه‌های الگوریتمها	۹۸
شکل ۲ - ۴۶: مقایسه زمان اجرا الگوریتمها	۹۸
شکل ۳ - ۱: نمودار میله‌ای میانگین طول مسیر الگوریتمها با ۱۰۰ تکرار	۱۳۸
شکل ۳ - ۲: نمودار میله‌ای میانگین زمان اجرا الگوریتمها با ۱۰۰ تکرار	۱۳۹
شکل ۳ - ۳: نمودار میله‌ای درصد انحراف معیار زمان و طول مسیر الگوریتمها با ۱۰۰ تکرار	۱۳۹
شکل ۳ - ۴: نمودار میله‌ای میانگین طول مسیر الگوریتمها با ۲۰ تکرار	۱۴۱
شکل ۳ - ۵: نمودار میله‌ای میانگین زمان اجرا الگوریتمها با ۲۰ تکرار	۱۴۲
شکل ۳ - ۶: نمودار میله‌ای درصد انحراف معیار زمان و طول مسیر الگوریتمها با ۲۰ تکرار	۱۴۲

فهرست جداول

جدول ۲ - ۱: پارامترهای الگوریتم غذایابی باکتریایی	۲۳
جدول ۲ - ۲: جدول پارامترهای الگوریتم بهینه‌سازی کرم شب‌تاب	۲۹
جدول ۲ - ۳: پارامترهای الگوریتم غذایابی باکتریایی	۳۸
جدول ۲ - ۴: جدول پارامترهای الگوریتم بهینه امپریالیست رقابتی	۴۸
جدول ۲ - ۵: جدول پارامترهای الگوریتم بهینه‌سازی ماهی الکتریکی	۶۲
جدول ۲ - ۶: نتایج بدست آمده از الگوریتم بهینه‌سازی غذایابی باکتریایی	۷۷
جدول ۲ - ۷: نتایج بدست آمده از الگوریتم بهینه‌سازی کرم شب‌تاب	۷۹
جدول ۲ - ۸: نتایج به دست آمده از الگوریتم جستجوی فاخته	۷۱
جدول ۲ - ۹: نتایج بدست آمده از الگوریتم امپریالیست رقابتی	۷۳
جدول ۲ - ۱۰: نتایج بدست آمده از الگوریتم بهینه‌سازی ماهی الکتریکی	۷۵
جدول ۲ - ۱۱: بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه	۷۹
جدول ۲ - ۱۲: میانگین نتایج اجرا الگوریتم غذایابی باکتریایی در هر اجرا (تمام داده‌ها میانگین هستند)	۸۰
جدول ۲ - ۱۳: بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه	۸۳
جدول ۲ - ۱۴: میانگین نتایج اجرا الگوریتم جستجوی فاخته در هر اجرا (تمام داده‌ها میانگین هستند)	۸۳
جدول ۲ - ۱۵: بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه	۸۶
جدول ۲ - ۱۶: میانگین نتایج اجرا الگوریتم ماهی الکتریکی در هر اجرا (تمام داده‌ها میانگین هستند)	۸۶
جدول ۲ - ۱۷: بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه	۹۰
جدول ۲ - ۱۸: میانگین نتایج اجرا الگوریتم کرم شب‌تاب در هر اجرا (تمام داده‌ها میانگین هستند)	۹۰
جدول ۲ - ۱۹: بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه	۹۳
جدول ۲ - ۲۰: میانگین نتایج اجرا الگوریتم امپریالیست رقابتی در هر اجرا (تمام داده‌ها میانگین هستند)	۹۳
جدول ۲ - ۲۱: میانگین بیشینه و کمینه الگوریتم‌ها	۹۷
جدول ۲ - ۲۲: میانگین نتایج احرافها به تفکیک الگوریتم‌ها	۹۷
جدول ۲ - ۲۳: مقایسه تعداد تکرار چرخه الگوریتم‌ها با ورودی‌های یکسان	۹۹
جدول ۲ - ۲۴: مقایسه دقت الگوریتم‌ها با ورودی‌های یکسان	۹۹
جدول ۲ - ۲۵: مقایسه زمان اجرا الگوریتم‌ها با ورودی‌های یکسان	۱۰۰
جدول ۲ - ۲۶: مقایسه کمینه و بیشینه تعداد تکرار چرخه	۱۰۰
جدول ۲ - ۲۷: مقایسه کمینه و بیشینه دقت عملکرد الگوریتم‌ها	۱۰۰
جدول ۲ - ۲۸: مقایسه کمینه و بیشینه زمان اجرا الگوریتم‌ها	۱۰۱
جدول ۳ - ۱: جدول پارامترهای الگوریتم بهینه‌سازی غذایابی باکتریایی	۱۰۹
جدول ۳ - ۲: جدول پارامترهای الگوریتم بهینه‌سازی کرم شب‌تاب	۱۱۳
جدول ۳ - ۳: جدول پارامترهای الگوریتم امپریالیست رقابتی	۱۱۷
جدول ۳ - ۴: جدول پارامترهای الگوریتم جستجوی فاخته	۱۲۵
جدول ۳ - ۵: جدول پارامترهای الگوریتم ماهی الکتریکی	۱۳۰

جدول ۳ - ۶: نتایج اجرا الگوریتم‌ها با ۱۰۰ تکرار.....	۱۳۸
جدول ۳ - ۷: نتایج اجرا الگوریتم‌ها با ۲۰ تکرار.....	۱۴۱
جدول پ - ۱: موقعیت باکتری‌های الگوریتم بهینه‌سازی غذایابی باکتریابی.....	۱۵۷
جدول پ - ۲: موقعیت لانه‌های الگوریتم جستجوی فاخته.....	۱۵۸
جدول پ - ۳: موقعیت ماهی‌های الکتریکی.....	۱۵۹
جدول پ - ۴: موقعیت کرم‌های الگوریتم بهینه‌سازی کرم شب‌تاب.....	۱۶۰
جدول پ - ۵: موقعیت کشورهای الگوریتم بهنه سازی امپریالیست رقابتی.....	۱۶۱
جدول ج - ۱: داده‌های خروجی از اجرای الگوریتم غذایابی باکتریابی با ورودی تعیین شده.....	۱۶۲
جدول ج - ۲: داده‌های خروجی از اجرای الگوریتم جستجوی فاخته با ورودی تعیین شده.....	۱۶۴
جدول ج - ۳: داده‌های خروجی از اجرای الگوریتم ماهی الکتریکی با ورودی تعیین شده.....	۱۶۵
جدول ج - ۴: داده‌های خروجی از اجرای الگوریتم کرم شب‌تاب با ورودی تعیین شده.....	۱۶۷
جدول ج - ۵: داده‌های خروجی از اجرای الگوریتم امپریالیست رقابتی با ورودی تعیین شده.....	۱۶۸
جدول ژ - ۱: داده‌های الگوریتم غذایابی باکتریابی.....	۱۷۰
جدول ژ - ۲: داده‌های الگوریتم جستجوی فاخته.....	۱۷۰
جدول ژ - ۳: داده‌های الگوریتم ماهی الکتریکی.....	۱۷۰
جدول ژ - ۴: داده‌های الگوریتم کرم شب‌تاب.....	۱۷۱
جدول ژ - ۵: داده‌های الگوریتم امپریالیست رقابتی.....	۱۷۱
جدول ژ - ۶: داده‌های الگوریتم غذایابی باکتریابی.....	۱۷۱
جدول ژ - ۷: داده‌های الگوریتم جستجوی فاخته.....	۱۷۱
جدول ژ - ۸: داده‌های ماهی الکتریکی.....	۱۷۲
جدول ژ - ۹: داده‌های الگوریتم کرم شب‌تاب.....	۱۷۲
جدول ژ - ۱۰: داده‌های الگوریتم امپریالیست رقابتی.....	۱۷۲

فهرست برنامه‌ها

برنامه ۲ - ۱:	ایجاد جمعیت اولیه باکتری‌ها ۲۳
برنامه ۲ - ۲:	حلقه‌ی اصلی الگوریتم بهینه‌سازی غذایابی باکتریابی ۲۴
برنامه ۲ - ۳:	وارد کردن ورودی‌های برنامه و ایجاد حلقه‌ی اصلی ۲۵
برنامه ۲ - ۴:	ایجاد جمعیت اولیه کرم‌های شبتاب ۳۰
برنامه ۲ - ۵:	تنظیم دوباره پارامترها و جمعیت جدید ۳۱
برنامه ۲ - ۶:	حلقه اصلی الگوریتم ۳۱
برنامه ۲ - ۷:	چک کردن مرزها و بدست آوردن نتایج ۳۲
برنامه ۲ - ۸:	مرتب سازی جواب‌ها و نمایش جواب بهینه ۳۲
برنامه ۲ - ۹:	وارد کردن ورودی‌های برنامه و ایجاد حلقه‌ی اصلی ۳۲
برنامه ۲ - ۱۰:	ایجاد جمعیت اولیه لانه‌ها در الگوریتم جستجوی فاخته ۳۸
برنامه ۲ - ۱۱:	تابع محاسبه بهترین لانه ۳۹
برنامه ۲ - ۱۲:	تابع پرواز لوى ۳۹
برنامه ۲ - ۱۳:	لانه‌های خالی کشف شده ۴۰
برنامه ۲ - ۱۴:	حلقه اصلی الگوریتم جستجوی فاخته ۴۰
برنامه ۲ - ۱۵:	وارد کردن ورودی‌های برنامه و ایجاد حلقه‌ی اصلی ۴۲
برنامه ۲ - ۱۶:	ایجاد جمعیت اولیه برای الگوریتم امپریالیست رقابتی ۴۸
برنامه ۲ - ۱۷:	انتخاب با روش چرخ رولت ۴۹
برنامه ۲ - ۱۸:	جذب ۵۰
برنامه ۲ - ۱۹:	انقلاب ۵۰
برنامه ۲ - ۲۰:	رقابت درون امپراطوری ۵۱
برنامه ۲ - ۲۱:	به روزرسانی هزینه کل ۵۱
برنامه ۲ - ۲۲:	رقابت بین امپراطوری ۵۱
برنامه ۲ - ۲۳:	وارد کردن ورودی‌های برنامه ۵۲
برنامه ۲ - ۲۴:	نسبت دادن ورودی‌ها به کشورها و امپریالیستها ۵۲
برنامه ۲ - ۲۵:	فرکانس و دامنه اولیه ۶۲
برنامه ۲ - ۲۶:	آغاز مرحله مکان‌یابی الکترونیکی فعل ۶۲
برنامه ۲ - ۲۷:	محاسبه فاصله ذرات با ذرات همسایه ۶۳
برنامه ۲ - ۲۸:	تغییر موقعیت ماهی‌ها در فاز فعل ۶۳
برنامه ۲ - ۲۹:	محاسبه احتمالات و انتخاب ذرات فعل ۶۳
برنامه ۲ - ۳۰:	بروزرسانی موقعیت ذرات در فاز غیرفعال ۶۴
برنامه ۲ - ۳۱:	تغییر پارامتر دیگر به صورت تصادفی ۶۴
برنامه ۲ - ۳۲:	تغییر موقعیت ذره‌ها برای رسیدن به جواب بهینه ۶۴
برنامه ۲ - ۳۳:	وارد کردن ورودی‌های برنامه و تخصیص به جمعیت اولیه ۶۵
برنامه ۳ - ۱:	تولید و ذخیره داده‌های الگوریتم کرم شبتاب ۱۰۴
برنامه ۳ - ۲:	تابع هزینه ۱۰۵
برنامه ۳ - ۳:	ایجاد نمودار وضعیت تابع هزینه در هر تکرار ۱۰۸

برنامه ۳ - ۴: تابع RES	
برنامه ۳ - ۵: تعریف پارامترها و داده‌های الگوریتم غذایابی باکتریایی	۱۰۸
برنامه ۳ - ۶: تعیین جمعیت اولیه باکتری‌ها	۱۱۰
برنامه ۳ - ۷: بهینه‌سازی غذایابی باکتریایی	۱۱۱
برنامه ۳ - ۸: تعریف پارامترها و داده‌های الگوریتم کرم شبتاب	۱۱۲
برنامه ۳ - ۹: تعیین جمعیت اولیه کرم‌های شبتاب	۱۱۴
برنامه ۳ - ۱۰: بهینه‌سازی کرم شبتاب	۱۱۵
برنامه ۳ - ۱۱: حرکت ذرات	۱۱۶
برنامه ۳ - ۱۲: تعریف پارامترها و داده‌های الگوریتم امپریالیست رقابتی	۱۱۷
برنامه ۳ - ۱۳: تعیین جمعیت اولیه کشورها	۱۱۹
برنامه ۳ - ۱۴: RouletteWheel	۱۲۰
برنامه ۳ - ۱۵: cal_total_fitness	۱۲۰
برنامه ۳ - ۱۶: بهینه‌سازی امپریالیست رقابتی	۱۲۱
برنامه ۳ - ۱۷: assimilation	۱۲۲
برنامه ۳ - ۱۸: Revolution	۱۲۳
برنامه ۳ - ۱۹: exchange	۱۲۴
برنامه ۳ - ۲۰: imperialistic_competition	۱۲۴
برنامه ۳ - ۲۱: تعریف پارامترها و داده‌های الگوریتم جستجوی فاخته	۱۲۶
برنامه ۳ - ۲۲: تعیین جمعیت اولیه لانه‌ها	۱۲۷
برنامه ۳ - ۲۳: بهینه‌سازی جستجوی فاخته	۱۲۷
برنامه ۳ - ۲۴: get_cuckoos	۱۲۸
برنامه ۳ - ۲۵: get_best_nest	۱۲۸
برنامه ۳ - ۲۶: empty_nests	۱۲۹
برنامه ۳ - ۲۷: تعریف پارامترها و داده‌های الگوریتم ماهی الکتروکی	۱۳۰
برنامه ۳ - ۲۸: تعیین جمعیت اولیه ماهی‌ها	۱۳۱
برنامه ۳ - ۲۹: بهینه‌سازی ماهی الکتروکی	۱۳۳
برنامه ۳ - ۳۰: activesearch	۱۳۵
برنامه ۳ - ۳۱: passivesearch	۱۳۶

فصل ۱:

مقدمه

۱ - ۱ مقدمه

بیش از نیم قرن است که انسان با علم رباتیک، که ترکیبی از رشته‌های مهندسی برق، مکانیک و کامپیوتر است، آشنا شده است. متخصصان این حوزه سال‌هاست که در تلاش برای خودکارسازی و ظایفی هستند که برای انسان دشوار، خطرناک یا پرچالش محسوب می‌شوند. برای مثال، تا حدود یک دهه پیش استفاده از نیروی انسانی در خطوط تولید خودرو بسیار رایج بود. هرچند که انسان قادر به انجام وظایفی مانند جوشکاری، رنگ‌کاری و مونتاژ قطعات خودرو است، اما هیچ‌گاه نمی‌تواند با سرعت و دقت یک ربات خودکار این کارها را انجام دهد. ورود ربات‌های خودکار به صنعت خودروسازی، بهبود کیفیت محصولات و افزایش تولید را به همراه داشته است، که این امر به کاهش قیمت محصولات در عین افزایش کیفیت منجر شده است. بنابراین می‌توان نتیجه گرفت که علم رباتیک تأثیر بسیار مثبتی بر زندگی انسان‌ها داشته است. یکی از حوزه‌های مهم پژوهشی در علم رباتیک، خودکارسازی دقیق حرکت ربات‌ها است. در حالی که دستیابی به دقت بالا برای بسیاری از ربات‌های ساده با مدل‌های دینامیکی شناخته شده و در محیط‌های آشنا به راحتی امکان‌پذیر است، پیشرفت سریع فناوری منجر به ساخت ربات‌های بسیار پیچیده با درجات آزادی زیاد و مدل‌های دینامیکی متنوع شده است. این موضوع طراحی یک الگوریتم خاص برای هر ربات را به امری بسیار طاقت‌فرسا، غیر بهینه و گاهی غیر ممکن تبدیل کرده است. امروزه تلاش می‌شود با ترکیب علم آمار و احتمال و بهره‌گیری از توان محاسباتی سخت‌افزارهای پیشرفته (که شرکت‌هایی مانند NVIDIA آن را ممکن ساخته‌اند)، این مشکل را حل کنیم. با استفاده از الگوریتم‌های بهینه‌سازی جامع، که نه تنها در علم رباتیک بلکه در علوم دیگری مانند اقتصاد و زیست‌شناسی نیز برای دستیابی به پاسخ‌های بهینه به کار می‌روند، می‌توان ربات‌ها را قادر ساخت تا با دریافت داده‌های محیطی از سنسورهای خود، بهترین مسیر را برای عبور از موانع و رسیدن به هدف انتخاب کنند. در این پژوهه تلاش شده است تا عملکرد یکی از دسته‌های الگوریتم‌های بهینه‌سازی به نام الگوریتم‌های فرالبتکاری را در بهینه‌سازی مسیر حرکت دو بعدی یک ربات سیار نقطه‌ای بررسی و نتایج این الگوریتم‌ها را با یکدیگر مقایسه کنیم. به همین منظور، در ادامه این گزارش به بررسی تاریخچه، چراجی استفاده و اصول مسیریابی ربات و بهینه‌سازی، به خصوص الگوریتم‌های فرالبتکاری خواهیم پرداخت.

۱ - ۲ مسیریابی ربات

مسیریابی ربات^۱ یکی از شاخه‌های بسیار مهم علم رباتیک می‌باشد که وظیفه‌ی تعیین مسیر مناسب برای حرکت ربات

^۱ Path Planning

در محیط خود می‌باشد. مسیر مناسب می‌تواند مولفه‌های گوناگونی داشته باشند از جمله: دوری از موانع، کوتاهی مسیر از نظر زمانی یا مسیر طی شده، سهولت مسیر و ... در نتیجه رسیدن به این امر مستلزم بدبست آوردن اطلاعات از محیط و موانع و نقطه‌ی شروع و پایان و علاوه بر این موارد بیرونی مستلزم آگاهی از دینامیک و سینماتیک ربات می‌باشد.

۱ - ۲ - ۱ جنبه‌های کلیدی مسیریابی ربات

۱. نمایش محیط: اولین مرحله در مسیریابی، نمایش محیط ربات است. این کار می‌تواند با استفاده از مدل‌های مختلفی مانند شبکه مشبک، گراف‌ها یا فضاهای پیوسته انجام شود. محیط ممکن است شامل موانع ثابت (مانند دیوارها و مبلمان) و موانع متحرک (مانند انسان‌های در حال حرکت و سایر ربات‌ها) باشد که ربات باید از میان آن‌ها عبور کند.

۲. فضای پیکربندی: مسیریابی معمولاً در فضای پیکربندی^۱ انجام می‌شود، که هر نقطه در آن نشان‌دهنده یک موقعیت و جهت‌گیری ممکن برای ربات است. فضای پیکربندی به ساده‌سازی محیط فیزیکی به یک فضای ریاضی کمک می‌کند که در آن الگوریتم‌های مسیریابی می‌توانند بطور موثر عمل کنند.

۳. الگوریتم‌های مسیریابی: الگوریتم‌های مختلفی برای حل مشکلات مسیریابی توسعه یافته‌اند که هر کدام دارای نقاط قوت و محدودیت‌های خاص خود هستند. برخی از پرکاربردترین الگوریتم‌ها شامل موارد زیر هستند:

- الگوریتم^{*} A: یک الگوریتم جستجوی اکتشافی که کوتاه‌ترین مسیر را در یک شبکه با کمینه‌سازی تابع هزینه پیدا می‌کند.

- نقشه‌های جاده‌ای احتمالی: یک روش که به طور تصادفی نقاطی را در فضای پیکربندی نمونه‌برداری کرده و این نقاط را به هم متصل می‌کند تا یک نقشه از مسیرهای ممکن ایجاد کند.

- درخت‌های جستجوی تصادفی سریع: یک الگوریتم که به صورت تدریجی یک درخت را با کاوش در فضا به سمت نقاط تصادفی انتخاب شده هدایت می‌کند، و برای فضاهای با ابعاد بالا مناسب است.

- روش‌های مبتنی بر بهینه‌سازی: این روش‌ها شامل تکنیک‌هایی مانند گرادیان نزولی^۲ و الگوریتم‌های فراباگاری^۳ مانند الگوریتم ژنتیک هستند که مسیر را بر اساس معیارهای خاص، مانند کمینه‌سازی مصرف انرژی یا حداکثر ایمنی، بهینه می‌کنند.

¹ Configuration space

² Gradient Descent

³ Metaheuristic Algorithm

۴. مسیریابی در محیط‌های ثابت و پویا

• مسیریابی ثابت: در محیط‌هایی که موانع حرکت نمی‌کنند، ربات می‌تواند قبل از اجرای مسیر، کل مسیر را محاسبه کند. این نوع مسیریابی معمولاً در محیط‌های ساختاریافته مانند انبارها یا کارخانه‌ها استفاده می‌شود.

• مسیریابی پویا: در محیط‌های پویا، موانع ممکن است حرکت کنند و ربات نیاز دارد که مسیر خود را در زمان واقعی بهروز کند. الگوریتم‌های این دسته معمولاً شامل حس کردن مداوم و بازنظمی مسیر برای تطبیق با تغییرات محیط هستند.

۵. مسیریابی سراسری و محلی

• مسیریابی سراسری: این نوع مسیریابی شامل برنامه‌ریزی مسیر در کل محیط است و اغلب از نقشه‌ای که از قبیل شناخته شده است، استفاده می‌کند. این روش زمانی مفید است که ربات اطلاعات قبلی از محیط داشته باشد.

• مسیریابی محلی: این نوع مسیریابی بر روی محیط‌های بالافصل ربات تمرکز دارد و برای زمانی که محیط تنها به طور جزئی شناخته شده است یا موانع پویا هستند، ضروری است. برنامه‌ریزان محلی اغلب با برنامه‌ریزان سراسری همکاری می‌کنند تا مسیر را به هنگام حرکت ربات بهبود دهند.

۶. مسیریابی چند رباتی: هنگامی که چندین ربات در یک محیط واحد کار می‌کنند، مسیریابی پیچیده‌تر می‌شود. ربات‌ها نه تنها باید از موانع اجتناب کنند، بلکه باید از برخورد با یکدیگر نیز جلوگیری کنند. استراتژی‌های هماهنگی مانند برنامه‌ریزی مبتنی بر اولویت یا الگوریتم‌های تعاملی برای اطمینان از عدم تداخل مسیرهای ربات‌ها استفاده می‌شود.

۱ - ۲ - تاریخچه پیشرفت علم مسیریابی ربات

مسیریابی ربات از نظر تاریخی، به عنوان یکی از مهم‌ترین مسائل در رباتیک، توسعه یافته است. این فرآیند به ربات‌ها امکان می‌دهد تا به صورت خودکار و بدون دخالت انسان، در محیط‌های مختلف حرکت کنند. مسیریابی ربات به طور پیوسته با پیشرفت‌های الگوریتمی، افزایش قدرت محاسباتی و بهبود تکنولوژی حسگرهای تکامل یافته است.

۱. آغاز اولیه (دهه ۱۹۵۰-۱۹۷۰): مفهوم مسیریابی ربات در دهه ۱۹۵۰ و ۱۹۶۰، هم‌زمان با ظهرور هوش مصنوعی (AI) و سایبرنیت یک، شروع شد. کارهای اولیه بیشتر نظری بودند و بر اصول هدایت ربات‌ها در محیط‌های ساده تمرکز داشتند. با این حال، این مدل‌های اولیه به دلیل محدودیت‌های قدرت محاسباتی آن زمان محدود بودند.

۲. روش‌های مبتنی بر شبکه: یکی از نخستین روش‌های مسیریابی، استفاده از روش‌های مبتنی بر شبکه^۱ بود که در آن

^۱ Grid-Based-method

محیط به یک شبکه تقسیم می شد. هر خانه شبکه یا آزاد بود یا توسط یک مانع اشغال شده بود. ریات می توانست از یک خانه به خانه مجاور و آزاد حرکت کند. الگوریتم هایی مانند A*, که در سال ۱۹۶۸ توسط پیترهارت، نیلز نیلسون و برترام رافائل توسعه یافت، به عنوان یکی از روش های پایه ای در این حوزه شناخته شد.^۱ یک الگوریتم جستجوی اکتشافی است که کوتاه ترین مسیر را از یک گره شروع به یک گره هدف پیدا می کند و در این مسیر موانع را در نظر می گیرد.

^۲. دهه ۱۹۸۰ - رویکرد احتمالاتی: دهه ۱۹۸۰ شاهد پیشرفت های قابل توجهی در مسیر یابی با معرفی روش های احتمالاتی بود. این روش ها انعطاف پذیر تر بودند و می توانستند به خوبی با فضاهای پیچیده و چند بعدی کار کنند.

^۳. نقشه های جاده ای احتمالی: نقشه های جاده ای احتمالی^۱ که در دهه ۱۹۹۰ توسعه یافتند، یک پیشرفت بزرگ محسوب می شوند. الگوریتم PRM با نمونه برداری تصادفی از نقاط در فضای پیکربندی و اتصال این نقاط با مسیر های ساده، نقشه ای از مسیر های ممکن ایجاد می کند. پس از ایجاد نقشه، مسئله مسیر یابی به جستجوی کوتاه ترین مسیر در این نقشه کاهش می یابد.

^۴. درخت های جستجوی تصادفی سریع^۲: الگوریتم RRT در سال ۱۹۹۸ توسط استیون لاوال و جیمز کافنر معرفی شد. این یک الگوریتم احتمالی است که به تدریج درختی را ایجاد می کند که در گره شروع قرار دارد و به سمت نقاط نمونه برداری شده حرکت می کند RRT. به ویژه برای فضاهای با ابعاد بالا مفید است و در رباتیک به طور گسترده ای استفاده شده است.

^۵. بهینه سازی و روش های ترکیبی: در طول دهه ۱۹۹۰ و ۲۰۰۰، رویکردها به سمت روش های مبتنی بر بهینه سازی تغییر کرد. این روش ها نه تنها به یافتن مسیر های قابل قبول می پرداختند، بلکه آن ها را بر اساس معیار هایی مانند کاهش مصرف انرژی یا افزایش ایمنی بهینه سازی می کردند.

^۶. الگوریتم ژنتیک و هوش گروهی: الهام گرفته از فرآیندهای طبیعی، الگوریتم های ژنتیک و روش های هوش گروهی مانند بهینه سازی دسته جمعی ذرات (PSO)^۳ و بهینه سازی کلونی مورچه ها^۴ (ACO) شروع به کاربرد در مسیر یابی کردند. این الگوریتم ها از اصول تکاملی یا رفتار جمعی در طبیعت برای یافتن مسیر های بهینه استفاده می کنند.

^۷. روش های ترکیبی: روش های ترکیبی شامل ترکیب الگوریتم های مختلف برای بهره گیری از نقاط قوت آن هاست. به عنوان مثال، ترکیب A* با PRM یا RRT با تکنیک های بهینه سازی برای بهبود کیفیت مسیر و کارایی محاسباتی.

^۸. یادگیری ماشین و ادغام هوش مصنوعی: جدید ترین پیشرفت ها در مسیر یابی شامل ادغام تکنیک های یادگیری

^۱ Probabilistic Roadmaps (PRM)

^۲ Rapidly-exploring Random Trees (RRT)

^۳ Particle Swarm Optimization

^۴ Ant Colony Optimization

ماشین و هوش مصنوعی است. این رویکردها به ربات‌ها امکان می‌دهند تا از تجربیات گذشته خود برای بهبود مسیریابی در محیط‌های ناشناخته استفاده کنند. شبکه‌های عصبی عمیق^۱ و یادگیری تقویتی^۲ از جمله تکنیک‌هایی هستند که به طور گسترده در این زمینه مورد استفاده قرار می‌گیرند.

۱ - ۲ - ۳ علت استفاده از مسیریابی ربات

ناویگری خودکار: مهم‌ترین دلیل برای مسیریابی ربات این است که به ربات‌ها اجازه می‌دهد بدون دخالت انسان از یک مکان به مکان دیگر حرکت کنند. این موضوع به ویژه در محیط‌هایی که حضور انسان عملی یا ایمن نیست، مانند مأموریت‌های فضایی، اعماق دریاهای سنتی خطرناک، اهمیت دارد.

اجتناب از موانع: مسیریابی تضمین می‌کند که ربات‌ها می‌توانند به طور ایمن در محیط‌های پر از موانع ثابت و متحرک حرکت کنند. چه در یک انبار با لیفتراک‌های متحرک و چه در خیابانی با عابران پیاده، الگوریتم‌های مسیریابی مؤثر به ربات‌ها امکان می‌دهند از برخوردها جلوگیری کنند و از این طریق ایمنی خود و دیگران را تضمین کنند.

بهینه‌سازی و کارایی: مسیریابی فقط برای رسیدن از نقطه‌ای به نقطه‌ای دیگر نیست، بلکه برای انجام این کار به کارآمدترین شکل ممکن است. این امر ممکن است شامل کاهش زمان سفر، کاهش مصرف انرژی، یا بهینه‌سازی مسیر برای وظایف خاص مانند تحویل کالا یا بازرگانی باشد. برای مثال، در انبارهای خودکار، ربات‌ها از مسیریابی برای بهینه‌سازی مسیرها استفاده می‌کنند، که در نتیجه به افزایش بهره‌وری و کاهش هزینه‌ها منجر می‌شود.

انطباق با محیط‌های پویا: ربات‌ها اغلب در محیط‌های پویایی کار می‌کنند که در آن موقعیت مowanع ممکن است تغییر کند. الگوریتم‌های پیشرفته‌ی مسیریابی به ربات‌ها امکان می‌دهند که به این تغییرات در زمان واقعی (Real-Time) واکنش نشان دهند و مسیرهای خود را در صورت لزوم دوباره برنامه‌ریزی کنند. این انطباق‌پذیری برای کاربردهایی مانند رانندگی خودکار یا مأموریت‌های نجات که محیط‌ها غیرقابل پیش‌بینی هستند، ضروری است.

قابلیت مقایسه‌پذیری برای وظایف پیچیده: با افزایش پیچیدگی وظایفی که به ربات‌ها محول می‌شود، نیاز به

^۱ Deep Neural Networks

^۲ Reinforcement Learning

مسیریابی پیشرفته بیشتر می شود. برای مثال، در ربات‌های گروهی^۱، چندین ربات باید با هم کار کنند و از برنخورد با یکدیگر جلوگیری کنند و رفتار جمعی خود را بهینه کنند. الگوریتم‌های مسیریابی برای هماهنگی این تعاملات پیچیده ضروری هستند.

تعامل انسان و ربات:^۲ در محیط‌هایی که ربات‌ها و انسان‌ها در کنار یکدیگر کار می‌کنند، مسیریابی برای اطمینان از تعاملات ایمن و روان بسیار مهم است. ربات‌ها باید بتوانند حرکات انسان‌ها را پیش‌بینی کنند و مسیرهای خود را به گونه‌ای برنامه‌ریزی کنند که از حوادث جلوگیری کرده و همکاری مؤثری ایجاد کنند.

کاربردهای مأموریتی حساس: در کاربردهایی مانند اکتشاف فضایی، دفاعی و جراحی‌های پژوهشکی، ربات‌ها اغلب موظف به انجام وظایفی با دقت بالا هستند. مسیریابی اطمینان می‌دهد که این وظایف با دقت لازم انجام می‌شوند، خواه این کار شامل پیمایش یک کاوشه‌گر در مریخ باشد یا هدایت یک ابزار جراحی در بدن انسان.

تأثیرات اقتصادی: با خودکارسازی وظایفی که قبلاً به صورت دستی انجام می‌شدند، ربات‌هایی که از مسیریابی کارآمد استفاده می‌کنند، می‌توانند به طور قابل توجهی هزینه‌های عملیاتی را کاهش دهند. این امر به ویژه در صنایعی مانند لجستیک مشهود است، جایی که ربات‌های متحرک خودکار یا AMRs حرکت کالاها را بهینه می‌کنند و منجر به کاهش هزینه‌های نیروی کار و افزایش بهره‌وری می‌شوند.

۱-۳- بجهنه‌سازی

بجهنه‌سازی^۳ فرایندی است که به دنبال یافتن بهترین راه حل ممکن یا نتیجه‌ی مطلوب در چارچوب پارامترها و محدودیت‌های تعریف‌شده است. این مفهوم در بسیاری از حوزه‌ها، از جمله ریاضیات، مهندسی، علوم کامپیوتر و حتی علوم اجتماعی، کاربرد گسترده‌ای دارد. بجهنه‌سازی را می‌توان از چندین دیدگاه مختلف طبقه‌بندی کرد که هر کدام به ویژگی‌های خاص الگوریتم‌ها و استفاده از اطلاعات مختلف در فرآیند جستجو بستگی دارد.

۱-۳-۱- طبقه‌بندی براساس استفاده از مشتق یا گرادیان

یکی از مهم‌ترین تقسیم‌بندی‌های الگوریتم‌های بجهنه‌سازی، بر اساس استفاده یا عدم استفاده از مشتق یا گرادیان تابع

^۱ Swarm Robotics

^۲ Human-Robot Interaction

^۳ Optimization

هدف است.

- **الگوریتم‌های مبتنی بر گرادیان:** این الگوریتم‌ها از اطلاعات گرادیان تابع هدف برای تعیین مسیر بهینه‌سازی استفاده می‌کنند. این روش‌ها برای توابع پیوسته و قابل مشتق مناسب هستند و معمولاً سرعت بالایی در همگرایی به سمت جواب بهینه دارند. مثال‌های رایج شامل الگوریتم‌های نزول گرادیان و روش نیوتون هستند.
- **الگوریتم‌های بدون گرادیان!**: در این نوع الگوریتم‌ها، از اطلاعات حاصل از مشتق گرفتن از منحنی مسیر استفاده نمی‌شود و به جای آن، به مقادیر تابع هدف برای جستجوی بهترین جواب تکیه می‌شود. این الگوریتم‌ها برای مسائل ناپیوسته یا مسائلی که محاسبه گرادیان در آن‌ها دشوار یا غیرممکن است، بسیار مناسب‌اند. از جمله این الگوریتم‌ها می‌توان به جستجوی تصادفی^۱ و بهینه‌سازی ازدحام ذرات^۲ اشاره کرد.

۱- ۳- ۲ طبقه بندي براساس ساختار جستجو

- الگوریتم‌های بهینه‌سازی را می‌توان به دو دسته‌ی مبتنی بر مسیر و مبتنی بر جمعیت تقسیم کرد.
- **الگوریتم‌های مبتنی بر مسیر:** این الگوریتم‌ها از یک یا چند راه حل در هر زمان استفاده می‌کنند و با تکرار مداوم، مسیر جستجوی خود را بهبود می‌بخشند. این روش‌ها معمولاً روی یک مجموعه محدود از راه حل‌ها تمرکز دارند و به تدریج به سمت بهینه حرکت می‌کنند. مثال‌های این دسته شامل الگوریتم تبرید شبیه‌سازی شده^۳ و الگوریتم جستجوی تابو^۴ است.
- **الگوریتم‌های مبتنی بر جمعیت:** در این نوع الگوریتم‌ها، جمعیتی از راه حل‌ها به طور همزمان استفاده می‌شود و از تعامل بین آن‌ها برای بهبود راه حل‌ها بهره گرفته می‌شود. این رویکرد باعث افزایش تنوع در فضای جستجو و کاهش احتمال گرفتار شدن در بهینه‌های محلی می‌شود. از جمله این الگوریتم‌ها می‌توان به الگوریتم ژنتیک^۵ و بهینه‌سازی ازدحام ذرات اشاره کرد.

^۱ Gradient-free Algorithms

^۲ Random Search

^۳ Particle Swarm Optimization

^۴ Simulated Annealing

^۵ Tabu Search

^۶ Genetic Algorithm

۱ - ۳ - ۳ طبقه بندی براساس قطعیت یا تصادفی بودن

الگوریتم‌های بهینه‌سازی را می‌توان به دو دسته‌ی قطعی^۱ و تصادفی^۲ تقسیم کرد.

- **الگوریتم‌های قطعی:** این الگوریتم‌ها بدون استفاده از عناصر تصادفی عمل می‌کنند و هر بار که از یک نقطه اولیه شروع شوند، به همان جواب نهایی خواهند رسید. این روش‌ها برای مسائلی که نیاز به ثبات و قابلیت تکرار دارند، بسیار مناسب هستند. الگوریتم نزول گرادیان یک مثال از این نوع الگوریتم‌ها است.
- **الگوریتم‌های تصادفی:** در این الگوریتم‌ها، از برخی عناصر تصادفی در فرآیند جستجو استفاده می‌شود. این تصادفی بودن به الگوریتم‌ها کمک می‌کند تا از بهینه‌های محلی فرار کنند و فضای جستجوی بیشتری را پوشش دهند. این ویژگی باعث می‌شود که هر بار اجرای الگوریتم، حتی از همان نقطه شروع، ممکن است به نتایج متفاوتی برسد. الگوریتم ژنتیک و بهینه‌سازی ازدحام ذرات از این دسته هستند.

۱ - ۳ - ۴ طبقه بندی براساس جستجوی محلی یا سراسری

یکی دیگر از دسته‌بندی‌های مهم در بهینه‌سازی، بر اساس توانایی الگوریتم‌ها در جستجوی محلی^۳ یا سراسری است.

- **الگوریتم جستجوی محلی:** این الگوریتم‌ها عمدتاً به سمت بهینه‌های محلی همگرا می‌شوند و معمولاً توانایی فرار از بهینه‌های محلی را ندارند. این ویژگی‌ها باعث می‌شود که این الگوریتم‌ها برای بهینه‌سازی جهانی مناسب نباشند. الگوریتم تبرید شبیه‌سازی شده از این دسته است.
- **الگوریتم جستجوی سراسری:** الگوریتم‌های جستجوی سراسری قادر به جستجوی کل فضای راه حل هستند و می‌توانند از بهینه‌های محلی فرار کرده و به سمت بهینه‌های جهانی حرکت کنند. بسیاری از الگوریتم‌های فرالبتکاری مدرن از این نوع هستند. الگوریتم ژنتیک و بهینه‌سازی ازدحام ذرات مثال‌هایی از این دسته هستند.

۱ - ۴ الگوریتم‌های فرالبتکاری

الگوریتم‌های با مولفه‌های تصادفی اغلب در گذشته به عنوان اکتشافی شناخته می‌شدند. اگرچه ادبیات اخیر تمایل دارد به

^۱ Deterministic

^۲ Stochastic

^۳ Local Search

آن‌ها به عنوان فرآبتكاری اشاره کند. ما از قرارداد گلاور پیروی می‌کنیم و همه الگوریتم‌های الهام گرفته از طبیعت مدرن را فرآبتكاری می‌نامیم. واژه "ابتکاری" به معنای یافتن یا کشف با آزمون و خطاست. "فرا" در اینجا به معنی بهبود اکتشافی می‌باشد. واژه فرآبتكاری توسط گلاور در مقاله اصلی خود ابداع شد. فرآبتكاری را می‌توان به عنوان دو چیز توصیف کرد:

- استراتژی اصلی: برای یافتن یا بهبود راه حل‌های فراتر از راه حل‌های محلی، با هدف بهینه‌سازی جهانی.
- استراتژی مکمل: برای بهبود روش‌های بهینه‌سازی محلی و ترکیب آن‌ها با تکنیک‌های دیگر.

الگوریتم‌های فرآبتكاری معمولاً از طریق ترکیب جستجوی محلی با جستجوی سراسری به دست می‌آیند. این به معنای وجود تعادلی میان بهینه‌سازی محلی و جهانی است. تقریباً همه الگوریتم‌های فرآبتكاری برای بهینه‌سازی جهانی مناسب هستند.

تعریف دیگری که می‌توان برای الگوریتم‌های فرآبتكاری داشت عبارت است از "الگوریتم‌های فرآبتكاری یک دسته از تکنیک‌های بهینه‌سازی سطح بالا و مستقل از مسئله هستند که راه حل‌های تقریبی برای مسائل پیچیده بهینه‌سازی ارائه می‌دهند. این الگوریتم‌ها به ویژه برای حل مسائلی مفید هستند که در آن‌ها روش‌های بهینه‌سازی سنتی ممکن است به دلیل پیچیدگی، اندازه‌یا غیرخطی بودن مسئله غیرعملی یا ناکارآمد باشند."

۱ - ۴ - ۱ ویژگی اصلی ساختار الگوریتم‌های فرآبتكاری

دو ویژگی اصلی در الگوریتم‌های فرآبتكاری عبارت‌اند از تشدید^۱ و تنوع^۲. تشدید به معنای جستجوی فشرده در فضای راه حل‌ها برای بهبود مداوم است، در حالی که تنوع به معنای ایجاد راه حل‌های متنوع برای جلوگیری از گرفتار شدن در بهینه‌های محلی است. تعادل خوبی بین تشدید و تنوع باید در طول انتخاب بهترین راه حل‌ها برای بهینه‌سازی فراهم شود. تشدید به معنی تمرکز بر روی بهینه‌سازی یک ناحیه خاص از فضای جستجو است، در حالی که تنوع به جستجوی مناطق مختلف از فضای راه حل کمک می‌کند.

- تشدید: الگوریتم‌هایی که از تشدید استفاده می‌کنند، به بهبود مداوم راه حل‌های فعلی می‌پردازن. این به معنی تمرکز بر روی مناطق خاصی از فضای جستجو است که بهینه‌سازی بیشتری می‌توان در آن‌ها انجام داد. الگوریتم‌های فرآبتكاری که از تشدید بهره می‌برند، معمولاً بهینه‌های محلی را با سرعت بیشتری پیدا می‌کنند.

¹ exploitation

² exploration

- **تنوع:** تنوع به معنی جستجوی گسترده‌تر در فضای راه حل‌ها است. پارامتر تنوع در الگوریتم به جلوگیری از گرفتار شدن در بهینه‌های محلی کمک می‌کند و شанс یافتن بهینه‌های جهانی را افزایش می‌دهد. الگوریتم‌های فرالبتکاری‌ای که از تنوع استفاده می‌کنند، معمولاً بهینه‌های جهانی را با اطمینان بیشتری پیدا می‌کنند. ترکیب خوبی از تشدید و تنوع معمولاً منجر به یافتن بهینه‌های جهانی می‌شود. الگوریتم‌های فرالبتکاری با استفاده از این دو ویژگی می‌توانند به طور مؤثری بهینه‌سازی‌های پیچیده را انجام دهند.

۱ - ۴ - ۲ تاریخچه استفاده از الگوریتم‌های فرالبتکاری

الگوریتم‌های فرالبتکاری به عنوان یکی از پایه‌های اصلی در حل مسائل پیچیده بهینه‌سازی در زمینه‌های مختلفی از جمله مهندسی، اقتصاد و هوش مصنوعی شناخته شده‌اند. توسعه و استفاده از این الگوریتم‌ها به عنوان ابزارهای بهینه‌سازی دارای تاریخچه‌ای غنی است که با نوآوری و تطبیق مستمر همراه بوده است. در اینجا به بررسی روند تکامل الگوریتم‌های فرالبتکاری به عنوان تکنیک‌های بهینه‌سازی می‌پردازیم.

مبانی اولیه و ظهور الگوریتم‌های فرالبتکاری: مفهوم روش‌های ابتکاری^۱ در بهینه‌سازی به اواسط قرن بیستم بازمی‌گردد. اصطلاح "ابتکاری" از کلمه یونانی "heuriskein" به معنای "یافتن" یا "کشف کردن" مشتق شده‌است. الگوریتم‌های ابتکاری در ابتدا به منظور ارائه راه حل‌های مناسب برای مسائل پیچیده‌ای طراحی شدند که در آنها روش‌های دقیق سنتی (مانند جستجوی کامل یا جامع) به دلیل محدودیت‌های محاسباتی امکان‌پذیر نبودند. در دهه‌های ۱۹۵۰ و ۱۹۶۰، روش‌های ابتکاری مانند صعود تپه‌ای^۲ و تبریز شبهیه‌سازی شده^۳ توسعه یافته‌اند. این روش‌ها با بهبود تدریجی یک راه حل پیشنهادی بر اساس مجموعه‌ای از قواعد یا معیارها، سعی در یافتن راه حل بهینه داشتند. با این حال، این الگوریتم‌های ابتدایی غالباً در بهینه‌های محلی گیر می‌کردند و در یافتن راه حل‌های جهانی کمتر موفق بودند.

تولد الگوریتم‌های فرالبتکاری: دهه ۱۹۸۰ یک نقطه عطف مهم بود که در آن الگوریتم‌های فرالبتکاری به طور رسمی معرفی شدند. اصطلاح "فرالبتکاری" توسط فرد گلاور^۴ در سال ۱۹۸۶ ابداع شد که الگوریتم جستجوی تابو را معرفی کرد. الگوریتم‌های فرالبتکاری با الگوریتم‌های ابتکاری متفاوت هستند زیرا مکانیزم‌هایی برای فرار از بهینه‌های محلی و جستجوی گسترده‌تر فضای راه حل‌ها را دارا می‌باشند، که آنها را در یافتن بهینه‌های جهانی قوی‌تر می‌کند. تحولات

^۱Heuristic

^۲Hill Climbing

^۳Simulated Annealing

^۴Fred Glover

- کلیدی در این دوره شامل موارد زیر است:
- تبرید شبیه‌سازی شده: این الگوریتم که توسط کرک‌پاتریک، گلت و وچی معرفی شد، از فرآیند انیلینگ در متالورژی الهام گرفته شده است. این الگوریتم برای جلوگیری از گیر افتادن در بهینه‌های محلی، گاهی اجازه می‌دهد که حرکت‌های "رو به بالا" انجام شود که به طور موقت راه حل را بدتر می‌کند اما ممکن است در آینده به راه حل‌های بهتر منجر شود.
 - الگوریتم ژنتیک: جان هلند و دانشجویانش در دانشگاه میشیگان الگوریتم‌های ژنتیک را در اوخر دهه ۱۹۷۰ توسعه دادند، اما این الگوریتم‌ها در دهه ۱۹۸۰ با انتشار کتاب دیوید ای. گلدبرگ با عنوان "الگوریتم‌های ژنتیک در جستجو، بهینه‌سازی و یادگیری ماشین"^۱ به شهرت گسترده‌ای دست یافتند. الگوریتم‌های ژنتیک از فرآیند انتخاب طبیعی و ژنتیک الهام گرفته و از مکانیزم‌هایی مانند انتخاب، ترکیب^۲ و جهش برای بهبود راه حل‌ها استفاده می‌کنند.
 - جستجوی تابو: جستجوی تابو که توسط فرد گلاور معرفی شد، مفهوم استفاده از ساختارهای حافظه برای پیگیری راه حل‌های قبلی و جلوگیری از بازگشت به آنها را مطرح کرد. این رویکرد امکان جستجوی گسترده‌تر فضای راه حل‌ها را فراهم کرد.
 - گسترش و تنوع: دهه‌های ۱۹۹۰ و ۲۰۰۰ شاهد گسترش سریع توسعه الگوریتم‌های فرالبتکاری بود، به طوری که رویکردهای جدیدی پدید آمدند که از طبیعت، زیست‌شناسی و فیزیک الهام گرفته بودند. این الگوریتم‌ها به دلیل تطبیق‌پذیری و توانایی در حل مسائل بسیار پیچیده بهینه‌سازی، به طور فزاینده‌ای محبوب شدند.
 - مشارکت‌های قابل توجه در این دوره شامل موارد زیر است:
 - بهینه‌سازی کلونی مورچگان^۳: مارکو دوریگو^۴ الگوریتم بهینه‌سازی کلونی مورچگان (ACO) را معرفی کرد که از رفتار جستجوی غذا توسط مورچگان الهام گرفته شده است ACO. از مکانیزم ارتباط بر اساس فرومون برای یافتن مسیرهای بهینه استفاده می‌کند و در حل مسائل بهینه‌سازی ترکیبی مانند مسئله فروشندۀ دوره‌گرد (TSP) بسیار مؤثر بوده است.
 - بهینه‌سازی ازدحام ذرات^۵: جیمز کنדי و راسل ابرهارت الگوریتم بهینه‌سازی ازدحام ذرات (PSO) را توسعه دادند که از رفتار اجتماعی پرندگان یا ماهیان الهام گرفته شده است PSO. به طور گسترده در مسائل بهینه‌سازی پیوسته مورد استفاده قرار گرفته و به دلیل سادگی و کارایی آن شناخته شده است.
 - تکامل تفاضلی^۶: راینر استورن و کنت پرایس الگوریتم تکامل تفاضلی (DE) را معرفی کردند که یک الگوریتم مبتنی بر جمعیت برای بهینه‌سازی توابع با مقادیر واقعی است. این الگوریتم به خاطر مقاومت و کارایی آن در یافتن

^۱ Cross Over

^۲ Marco Dorigo

^۳ Differential Evolution

بهینه‌های جهانی مورد تقدیر قرار گرفته است.

تحولات مدرن و هیبریداسیون: در قرن ۲۱، پیشرفت‌های بیشتری در الگوریتم‌های فرالبتکاری صورت گرفته است، با یک روند رو به رشد به سوی هیبریداسیون – ترکیب الگوریتم‌های مختلف فرالبتکاری یا ادغام آنها با سایر تکنیک‌های بهینه‌سازی (مانند روش‌های دقیق، یادگیری ماشین) برای بهبود عملکرد.

- **الگوریتم‌های هیبرید فرالبتکاری:** این الگوریتم‌ها نقاط قوت رویکردهای فرالبتکاری مختلف را ترکیب می‌کنند تا کیفیت راه حل‌ها و سرعت همگرایی را بهبود بخشنند. به عنوان مثال، ترکیب الگوریتم‌های ژنتیک با تبرید شبیه‌سازی شده یا ادغام جستجوی تابو در بهینه‌سازی ازدحام ذرات منجر به ابزارهای بهینه‌سازی قدرتمندتری شده است.

- **بهینه‌سازی چندهدفه:** ظهور الگوریتم‌های فرالبتکاری چندهدفه، مانند الگوریتم ژنتیک مرتب‌سازی غیرچیره (NSGA-II)، امکان بهینه‌سازی چندین هدف متضاد به طور هم‌زمان را فراهم کرده است که در کاربردهای واقعی بسیار حیاتی است.

- **فرالبتکاری‌ها در داده‌های بزرگ و هوش مصنوعی:** با ظهور داده‌های بزرگ و افزایش هوش مصنوعی، الگوریتم‌های فرالبتکاری کاربردهای جدیدی در زمینه‌هایی مانند انتخاب ویژگی‌ها، تنظیم ابرپارامترها و یادگیری عمیق پیدا کرده‌اند. توانایی آنها در مدیریت فضای جستجوی بزرگ و پیچیده باعث شده که برای این وظایف ایده‌آل باشند. تاریخچه الگوریتم‌های فرالبتکاری نشان‌دهنده‌یک حوزه پویا و در حال تکامل است که با نوآوری و تطبیق مستمر همراه بوده است. از ریشه‌های ابتکاری اولیه تا الگوریتم‌های هیبریدی و الهام‌گرفته از طبیعت مدرن امروزی، فرالبتکاری‌ها به ابزارهای ضروری در حل مسائل پیچیده بهینه‌سازی در طیف گسترده‌ای از رشته‌ها تبدیل شده‌اند. با ادامه رشد قدرت محاسباتی و ظهور چالش‌های جدید، توسعه الگوریتم‌های فرالبتکاری احتمالاً ادامه خواهد یافت و به پیشرفت‌های بیشتری در نظریه و عمل بهینه‌سازی منجر خواهد شد.

۱ - ۴ - ۳ - انواع الگوریتم‌های فرالبتکاری

عموماً الگوریتم‌های فرالبتکاری براساس اینکه بدنیال بهینه‌سازی محلی یا سراسری هستند تقسیم می‌شوند. در نتیجه، الگوریتم‌های فرالبتکاری به طور کلی به دو دسته اصلی تقسیم می‌شوند الگوریتم‌های مبتنی بر جمعیت و مبتنی بر جستجوی محلی.

۱ - ۴ - ۳ - ۱ - الگوریتم‌های مبتنی بر جمعیت

این الگوریتم‌ها با استفاده از یک جمعیت از راه حل‌ها به طور همزمان به جستجوی فضای راه حل‌ها می‌پردازند. هر عضو جمعیت یک راه حل ممکن است که با سایر اعضاء تعامل دارد و با تبادل اطلاعات، به بهبود راه حل‌ها کمک می‌کند. برخی از الگوریتم‌های معروف مبتنی بر جمعیت عبارتند از:

- الگوریتم زنتیک
- الگوریتم ازدحام ذرات
- الگوریتم کرم شب تاب^۱
- الگوریتم بهینه‌سازی کلونی مورچگان

۱-۴-۳-۲ الگوریتم‌های مبتنی بر جستجوی محلی

این الگوریتم‌ها با شروع از یک یا چند راه حل اولیه و با بهبود تدریجی آنها از طریق جستجوی محلی، به دنبال یافتن بهینه سراسری هستند. این الگوریتم‌ها به طور معمول از تکنیک‌های مختلف برای فرار از بهینه‌های محلی استفاده می‌کنند. برخی از الگوریتم‌های معروف مبتنی بر جستجوی محلی عبارتند از:

- الگوریتم تبرید شبیه‌سازی شده
- الگوریتم جستجوی تابو
- الگوریتم جستجوی هارمونی^۲

این دو دسته اصلی، هر یک شامل انواع مختلفی از الگوریتم‌ها هستند که هر کدام با استفاده از روش‌ها و تکنیک‌های خاص خود، به حل مسائل بهینه‌سازی پیچیده می‌پردازند.

۱-۴-۴ چرا باید از الگوریتم‌های فرآبتكاری استفاده کنیم.

در محاسبات کلاسیک تلاش بر مدل‌سازی و حل مستقیم یک مسئله بوده است، هر چند این نحوه محاسبه برای بسیاری از مسائل دنیای واقعی به دلیل پیچیدگی زیاد و دخیل بودن فاکتورهای فراوان، برای رسیدن به جواب مطلوب کاربردی ندارد و عملاً به جوابی نخواهیم رسید زیرا بسیاری از مسائل به این صورت هستند که نمی‌توان گفت یک

^۱ Firefly Algorithm

^۲ Harmony Search

جواب دارند یا خیر و جوابشان چیست. تنها می‌توان با مقایسه ویژگی‌های هر کدام، بهترین جواب را انتخاب کرد. در این موارد، استفاده از روش‌های عددی کاربرد بسیار زیادی دارد. این روش‌ها عموماً با تکرار یک فرآیند و انجام محاسبات پیچیده ولی ساده‌تر، می‌توانند به جوابی بهینه (نه قطعی) برسند. از زیرمجموعه‌های روش‌های عددی، می‌توان به مسائل بهینه‌سازی اشاره کرد. برخی از این الگوریتم‌های بهینه‌سازی که از طبیعت الهام گرفته شده‌اند، الگوریتم‌های فرالبتکاری یا متاهیوریستیک نامیده می‌شوند. الگوریتم‌های فرالبتکاری ابزارهای قدرتمندی هستند که برای حل مسائل پیچیده بهینه‌سازی استفاده می‌شوند، مسائلی که حل آن‌ها با استفاده از روش‌های سنتی دشوار یا غیرممکن است. در ادامه به دلایل اصلی استفاده از این الگوریتم‌ها می‌پردازیم:

- **مدیریت مسائل پیچیده و غیرخطی:** بسیاری از مسائل بهینه‌سازی در دنیای واقعی دارای مناظر غیرمحدب، غیرخطی و چندحالته^۱ هستند، جایی که روش‌های سنتی مانند نزول گرادیان قادر به یافتن بهینه‌ی جهانی نیستند. الگوریتم‌های فرالبتکاری مانند الگوریتم‌های ژنتیک، شبیه‌سازی تبرید و بهینه‌سازی دسته‌جمعی ذرات برای کاوش در این فضاهای پیچیده طراحی شده‌اند. مسائلی که شامل متغیرهای گستره یا ساختارهای ترکیبی هستند، مانند مسئله فروشنده دوره‌گرد یا مسائل زمان‌بندی، اغلب با استفاده از فرالبتکاری‌ها بهتر حل می‌شوند.
- **جلوگیری از گیر افتادن در بهینه‌های محلی:** الگوریتم‌های فرالبتکاری برای جلوگیری از گیر افتادن در بهینه‌های محلی طراحی شده‌اند، که این مشکل رایج در روش‌های بهینه‌سازی سنتی است. این الگوریتم‌ها از استراتژی‌هایی مانند تصادفی بودن، جستجوی جمعیتی و مکانیزم‌های تطبیقی برای کاوش کامل‌تر فضای حل استفاده می‌کنند.
- **انعطاف‌پذیری و سازگاری:** فرالبتکاری‌ها را می‌توان به طیف گسترده‌ای از مسائل بهینه‌سازی تطبیق داد بدون اینکه نیاز به دانش عمیق و خاصی از مسئله باشد. این الگوریتم‌ها را می‌توان به راحتی سفارشی کرد یا با الگوریتم‌های دیگر ترکیب کرد تا عملکرد آن‌ها بهبودیابد. این الگوریتم‌ها می‌توانند مسائل با اندازه‌های مختلف، از مسائل کوچک تا مسائل بزرگ را مدیریت کنند.
- **توانایی جستجوی جهانی:** فرالبتکاری‌ها از یک استراتژی جستجوی جهانی استفاده می‌کنند که به آن‌ها اجازه می‌دهد تمام فضای حل را کاوش کنند، نه فقط یک همسایگی محلی. این امر آن‌ها را برای یافتن بهینه‌ی جهانی در مسائل با مناظر پیچیده مناسب می‌سازد.
- **ساده بودن پیاده‌سازی:** در مقایسه با الگوریتم‌های بهینه‌سازی دقیق که ممکن است نیاز به فرموله‌سازی‌های ریاضی دقیق و الگوریتم‌های خاص مسئله داشته باشند، فرالبتکاری‌ها اغلب ساده‌تر پیاده‌سازی می‌شوند و فرضیات کمتری درباره ساختار مسئله دارند.

^۱ Multi-Modal

- **پتانسیل موازی‌سازی^۱:** بسیاری از الگوریتم‌های فرابتکاری، به ویژه آن‌هایی که بر پایه جمعیت هستند مانند الگوریتم‌های ژنتیک و بهینه‌سازی دسته‌جمعی ذرات، به صورت ذاتی موازی‌پذیر هستند. این ویژگی آن‌ها را برای پیاده‌سازی بر روی معماری‌های محاسبات موازی مدرن مناسب می‌سازد و منجر به افزایش سرعت محاسبات می‌شود.
 - کاربرد در بهینه‌سازی چندهدفه^۲: در بسیاری از سناریوهای دنیای واقعی، چندین هدف متضاد وجود دارد که باید در نظر گرفته شوند. فرابتکاری‌ها را می‌توان برای حل مسائل بهینه‌سازی چندهدفه توسعه داد و مجموعه‌ای از راه حل‌های بهینه معروف به جبهه پارتو^۳ را ارائه داد.
 - استحکام در محیط‌های نامطمئن و پویا: الگوریتم‌های فرابتکاری در برابر محیط‌های نویزی، نامطمئن و پویا که در آن‌ها پارامترهای مسئله ممکن است با گذشت زمان تغییر کنند، مقاوم هستند. این ویژگی آن‌ها را برای کاربردهای Real-Time و مسائل با محدودیت‌های متغیر مناسب می‌سازد.
 - عدم نیاز به اطلاعات گرادیان: برخلاف روش‌های بهینه‌سازی مبنی بر گرادیان، فرابتکاری‌ها نیازی به اطلاعات گرادیان ندارند، که این ویژگی آن‌ها را برای بهینه‌سازی توابع هدف غیرقابل تفکیک، ناپیوسته یا نویزی مناسب می‌کند.
 - قابلیت کاربرد گسترده در حوزه‌های مختلف: الگوریتم‌های فرابتکاری در حوزه‌های مختلف از جمله طراحی مهندسی، لجستیک، مالی، یادگیری ماشین و بیوانفورماتیک به طور موقوفیت‌آمیزی به کار گرفته شده‌اند که این نشان‌دهنده اثربخشی و همه‌کاره بودن آن‌هاست.
- الگوریتم‌های فرابتکاری یک رویکرد قدرتمند و انعطاف‌پذیر برای حل مسائل پیچیده بهینه‌سازی ارائه می‌دهند، به ویژه زمانی که روش‌های سنتی ناکافی باشند. توانایی آن‌ها در مدیریت انواع مختلف مسائل، اجتناب از بهینه‌های محلی و تطبیق با شرایط متغیر، آن‌ها را به ابزارهای ضروری در زمینه‌ی بهینه‌سازی تبدیل کرده است.

^۱ Parallelization Potential

^۲ Multi-Objective Optimization

^۳ Pareto Front

فصل ۲:

الگوریتم‌های فرآابتکاری در بهینه‌سازی

۲-۱-۱- تئوری و روند الگوریتم‌ها

در این فصل، به بررسی تئوری الگوریتم‌های فرالبتکاری، فرمول‌ها و روابط استفاده شده در این الگوریتم‌ها، نحوه مدل‌سازی این الگوریتم‌ها در نرم افزار متلب، بررسی نحوه عملکرد این الگوریتم‌ها بر روی تابع تست شانه تخم مرغی با ورودی متغیر و ثابت، و در نهایت مقایسه عملکرد این الگوریتم‌ها می‌پردازیم.

۲-۱-۱- الگوریتم بهینه‌سازی غذایابی باکتریایی^۱

برای غذایابی اجتماعی، یک حیوان به قابلیت‌های ارتباطی نیاز دارد و در طی یک دوره زمانی، مزایایی به دست می‌آورد که می‌تواند از قابلیت‌های حسگری گروه بهره‌برداری کند. این به گروه کمک می‌کند تا طعمه‌های بزرگ‌تر را شکار کند یا در عوض، افراد می‌توانند هنگام حضور در یک گروه، محافظت بهتری در برابر شکارچیان داشته باشند. ما رفتار جستجوی باکتری رایج، ای‌کولای^۲ را در نظر گرفتیم. رفتار و حرکت آن از مجموعه ای از شش تاژک چرخان صلب(۱۰۰ تا ۲۰۰ دور در ثانیه) ناشی می‌شود که هر کدام به عنوان یک موتور بیولوژیکی هدایت می‌شوند. سرعت دویدن $20\mu ms^{-1}$ است، اما آن‌ها نمی‌توانند بطور مستقیم شنا کنند. اقدامات کموتاتیک باکتری به شرح زیر مدل‌سازی می‌شود:

- در یک محیط خشی، اگر باکتری بطور مداوم بدود یا غلت بخورد، حرکت باکتری مشابه با جستجو می‌باشد.
- اگر در جهت شدت ماده مغذی شنا کند(یا در جهت خروج از شدت مواد مضر) و یا مدت طولانی تری شنا کند) نزدیکی به شدت مواد مغذی یا دوری از شدت مواد مضر)، رفتارش همانند جستجو برای محیط‌های مطلوب‌تر می‌باشد.

در نتیجه، می‌توان بیان کرد که باکتری تمایل به حرکت به سمت مواد مغذی موجود در محیط و دوری از مواد مضر موجود در محیط دارد. پروتئین‌های گیرنده از جمله حسگرهایی هستند که برای تشخیص دقیق به وضوح نیاز و دقت بالایی نیاز دارند، این موضوع به این معنی است که یک تغییر کوچک در غلظت مواد مغذی می‌تواند تغییر قابل توجهی در رفتار ایجاد کند. این احتمالاً بهترین سیستم حسی و تصمیم‌گیری در زیست‌شناسی است. جهش در باکتری ای‌کولای

^۱ Bacterial foraging algorithm
^۲ E-Coli

بر بازدهی تولیدمثُل در دمای‌های گوناگون، تاثیر می‌گذارد و با نرخی حدود 10^{-7} در تمام زن‌ها در هر نسل رخ می‌دهد. ای کولای گهگاه در گیر یک جفت‌گیری می‌شود که بر ویژگی‌های جمعیت تاثیر می‌گذارد. انواع مختلفی از آرایش‌ها وجود دارند که در باکتری‌ها استفاده می‌شوند مانند آئروتاکسی^۱ (جذب اکسیژن)، فوتوتاکسی^۲ (نور)، ترموتاکسی^۳ (دما)، مگنوتاکسی^۴ (خطوط مغناطیسی شار). برخی از باکتری‌ها می‌توانند شکل و تعداد تاژک‌های خود را برای پیکربندی مجدد، به منظور اطمینان از جستجوی کارآمد براساس محیط، تغییر دهند. باکتری‌ها قادرند در محیط‌های نیمه جامد، الگوهای مکانی-زمانی پیچیده و پایداری ایجاد کنند. اگر این باکتری‌ها ابتدا در مرکز مواد مغذی قرار گیرند، می‌توانند از طریق این محیط به بقا ادامه دهند. همچنین، در شرایط خاص، با ترشح سیگنال‌هایی از یک سلول به سلول دیگر، یکدیگر را جذب کرده، گروه‌بندی می‌کنند و از هم‌دیگر محافظت می‌نمایند. مراحل اساسی که در شبیه‌سازی الگوریتم جستجوی باکتری‌ها استفاده می‌شود، به شرح زیر است:

• کموتاکسی^۵

این مرحله حرکت ای کولای را از طریق شنا و غلت زدن، با استفاده از تاس‌گری شبیه‌سازی می‌کند. باکتری‌ها برای تمام طول عمر بین این دو حالت عملیاتی، متناوب هستند. حرکت باکتری‌ها طبق معادله ۱_۲ نشان داده شده است:

$$\theta^i(j+1, k, l) = \theta^i(j, k, l) + c^i \frac{\Delta(i)}{\sqrt{\Delta(i)^T \Delta(i)}} \quad (1_2)$$

• تولیدمثُل^۶

ای کولای می‌تواند در محیط مطلوب دمایی (دمای بدن) و با دریافت منابع غذایی کافی، همه چیز را تکثیر و سنتز کند. سرعت ساختن یک کلون از باکتری ای کولای در حدود ۲۰-۲۲ دقیقه است، در نتیجه یک سلول ای کولای در مدت ۲۴ ساعت می‌تواند ۲۷۲ سلول از خود را کلون کند که برای بسته بندی یک مکعب ۱۷ متری کافی است.

• حذف پراکندگی^۷

از آنجایی که باکتری ای کولای به دما بسیار حساس می‌باشد، با افزایش دما در یک منطقه تمام باکتری‌های آن منطقه از

^۱ aero taxis

^۲ phototaxis

^۳ thermotaxis

^۴ magneto taxis

^۵ Chemotaxis

^۶ reproduction

^۷ Elimination and dispersal

بین می‌روند یا گروهی در یک مکان جدید پراکنده می‌شوند.

۲- ۱- ۱- ۱- الگوریتم

الگوریتم غذایابی باکتریایی براساس چهار مکانیزم اصلی مشاهده شده در یک سیستم باکتریایی واقعی است: کموتاكسی، ازدحام، تولیدمثل و حذف پراکنده‌گی. در این قسمت یک شبکه کد برای BFOA گام به گام توضیح داده شده است. در زیر مخفف‌های مختلف استفاده شده در این شبکه کد آمده است:

L: نمایه گام کموتاتیک

k: نمایه گام تولیدمثل

l: نمایه گام حذف پراکنده‌گی

p: ابعاد فضای جستجو

S: تعداد باکتری‌های مجموعه

N_c: تعداد گام‌های کموتاتیک

N_s: طول شنا

N_{re}: تعداد گام‌های تولیدمثل

N_{ed}: تعداد رویدادهای حذف پراکنده‌گی

P_{ed}: احتمال حذف پراکنده‌گی

C(i): اندازه گام برداشته شده در جهت تصادفی مشخص شده توسط غلتک

$P(j, k, l) = \theta^i(j, k, l) | i = 1, 2, \dots, S$ نشان‌دهنده موقعیت هر عضو جمعیت S باکتری‌ها در گام زام کموتاتیک، گام kام تولیدمثل و گام lام رویداد حذف پراکنده‌گی است. $J(j, k, l)$ نشانگر مقدار تابع هزینه برای باکتری آن در موقعیت (j, k, l) می‌باشد.

مراحل الگوریتم که در شبکه کد (Δ) خلاصه شده در ادامه آورده شده است:

۱. مقداردهی اولیه پارامترهای $p, S, N_c, N_s, N_{re}, N_{ed}, P_{ed}, C(i), (i = 1, 2, \dots, S)$, θ^i

۲. حلقه حذف پراکنده‌گی

۳. حلقه تولیدمثل:

۴. حلقه کموتاكسی:

۵. برای $i = 1, 2, \dots, S$ یک گام کموتاتیک برای باکتری آن انجام دهید.

ب. تابع هزینه $J(i, j, k, l)$ را محاسبه کنید.

ج. $J_{Last} = J(i, j, k, l)$ را ایجاد می‌کنیم تا این مقادیر را ذخیره کنیم زیرا ممکن است هزینه کمتر و بهتری را از طریق اجرا پیدا کنیم.

د. غلت زدن: یک بردار تصادفی $\Delta(i)$ ایجاد کنید که هر عضو $(m = 1, 2, \dots, p)$ آن یک عدد تصادفی در بازه $[1, -1]$ باشد.

ه. حرکت $\theta^i(j+1, k, l) = \theta^i(j, k, l) + c^i \frac{\Delta(i)}{\sqrt{\Delta(i)^T \Delta(i)}}$ را انجام دهید.

و. $J(i, j+1, k, l)$ را محاسبه کنید.

ز. شنا کردن: m (شمارنده طول شنا) را برابر صفر قرار دهید. تا زمانی که $N_s < m$ (اگر برای مدت طولانی به پایین صعود نکرده باشد) $J(i, j+1, k, l) < J_{Last}$ قرار دهید. اگر $J(i, j+1, k, l) = J_{Last}$ را قرار دهید و از این

سپس $\theta^i(j+1, k, l) = \theta^i(j, k, l) + c^i \frac{\Delta(i)}{\sqrt{\Delta(i)^T \Delta(i)}}$ را قرار دهید و از این برای محاسبه $J(i, j+1, k, l)$ استفاده کنید. در غیر اینصورت m را برابر N_s قرار دهید و حلقه و شرط را تمام کنید.

ح. به باکتری بعدی $i+1$ بروید (اگر i برابر S نبود) یعنی به حالت (۴ب) برگردید تا باکتری بعدی را بررسی کنید.

۵. اگر $N_c < j$ به مرحله (۴) بروید. در این مورد کموتاکسی را ادامه دهید زیرا عمر باکتری به پایان نرسیده است.

۶. تولیدمثل:

ا. برای k و l برای هر $i = 1, 2, \dots, S$ را سلامت باکتری i ام قرار دهید (معیاری از تعداد مواد مغذی دریافتی در طول عمر خود و میزان موفقیت باکتری در اجتناب از مواد مضر). باکتری‌ها و پارامتر کموتاکسی $C(i)$ را به ترتیب صعودی J_{health} دسته بندی کنید (هزینه بالاتر به معنی سلامت کمتر است).

ب. باکتری‌های S با بالاترین مقدار J_{health} می‌برند و باکتری‌های S باقی‌مانده با بهترین مقادیر تقسیم می‌شوند. (این فرآیند توسط کپی‌هایی که ساخته می‌شوند و در همان مکان والدینشان قرار می‌گیرند، انجام می‌شود).

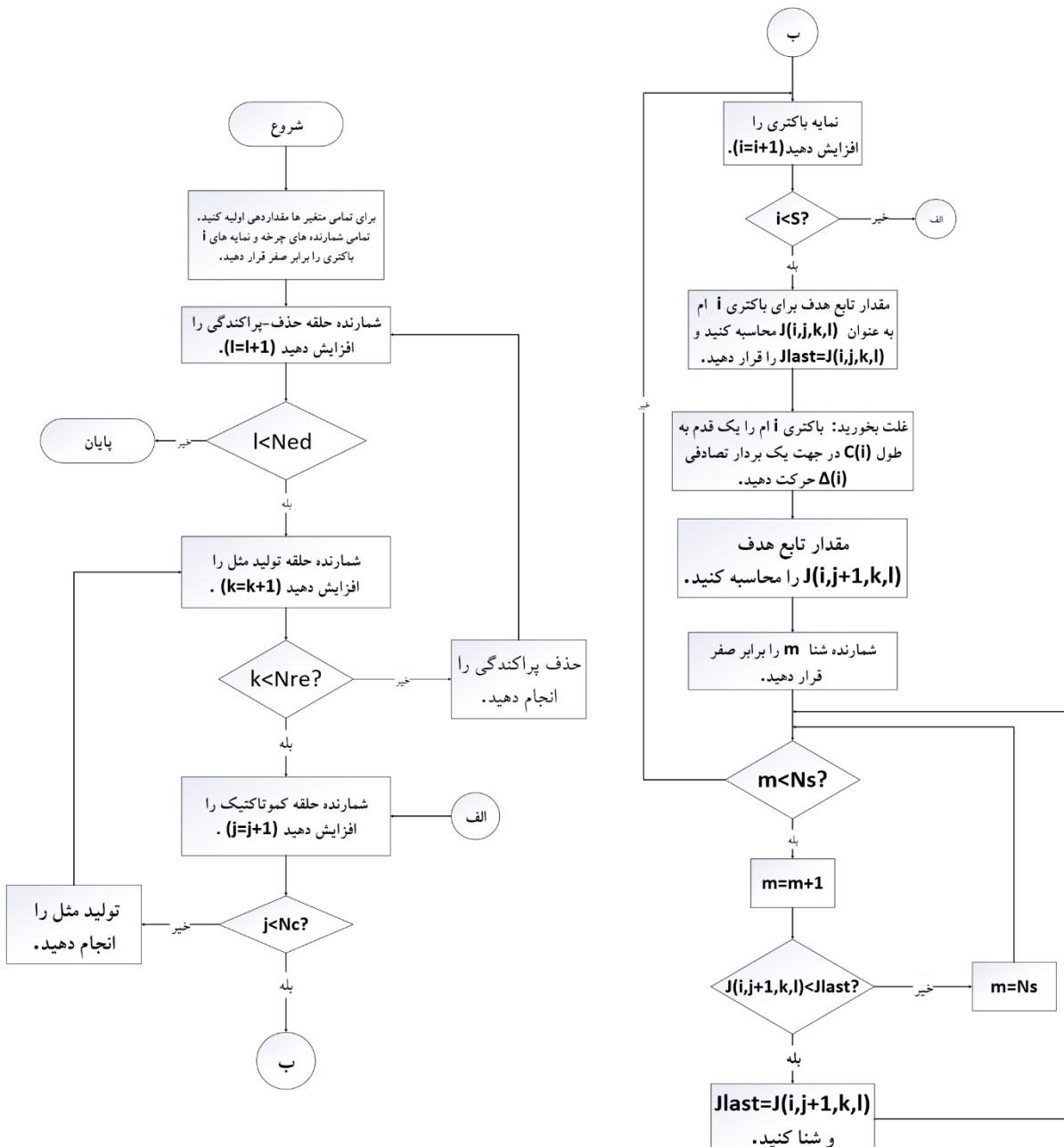
۷. اگر $k < N_{re}$ باشد، به مرحله (۳) بروید. در این حالت به تعداد مراحل تولیدمثل مشخص شده نرسیده‌ایم، بنابراین

نسل بعدی حلقه کمotaتیک را شروع می‌کنیم.

۸. حذف پراکندگی: برای $i=1, 2, \dots, S$ با احتمال P_{ed} ، هر باکتری را از بین بپرید و پراکنده کنید (با این کار تعداد باکتری‌ها در جمعیت ثابت می‌ماند). برای انجام این کار، اگر یک باکتری از بین رفت، به سادگی یک باکتری دیگر را در یک مکان تصادفی در دامنه بهینه‌سازی پراکنده کنید.

اگر $I < N_{ed}$ به مرحله ۲ بروید. در غیر اینصورت چرخه را پایان دهید.

در ادامه فلوچارت این الگوریتم در شکل ۲ - ۱ Error! Reference source not found. نشان داده شده است.



شکل ۲ - ۱: فلوچارت غذایابی باکتریایی

۲-۱-۲ مدل‌سازی الگوریتم با ورودی تصادفی

برای مدل‌سازی عیناً مانند الگوریتم و طبق مرحله (۱) عمل می‌کنیم. پارامترهای تعیین شده در بخش اول کد در جدول ۲ - ۱ آورده شده است. بعد از آن ابتدا یک جمعیت تصادفی ایجاد می‌شود.

جدول ۲ - ۱: پارامترهای الگوریتم غذایابی باکتریایی

مقدار	پارامتر
-۵۱۲ ۵۱۲	<i>Range</i>
۶۰	<i>population</i>
۳۰	N_c
۲	N_s
۴	N_{re}
۴	N_{ed}
۳۰	$S_r(S/2)$
۰.۵	P_{ed}
۰.۸	C

برنامه ۲ - ۱: ایجاد جمعیت اولیه باکتری‌ها

```
%% initialization

x=rand(Np,D)*(ub-lb)+lb;

for i=1:Np %for plotting
    x11(i,1)=x(i,1);
    y11(i,1)=x(i,2);
end

for k=1:Np
    v= x(k,:); %initial population
    J(k)=b(v); % initial fitness calculation
end
```

با استفاده از حلقه‌ی while برای ایجاد شرط خروج با تلرانس دقیق ۰.۰۱، تمامی مراحل توضیح داده شده الگوریتم را عیناً مرحله به مرحله با استفاده از حلقه‌های for و شرط if مدل‌سازی و اجرا می‌شود.

برنامه ۲ - ۲: حلقه‌ی اصلی الگوریتم بهینه‌سازی غذایابی باکتریایی

```

for iter=1:Ne %elimination
    for k=1:Nr %reproduction
        Jchem=J; %for calculating Jhealth
        for j=1:Nc
            % Chemotaxis Loop %

                for i=1:Np
                    del=(rand(1,D)-0.5)*2; %random vector
                    x(i,:)=x(i,:)+(C/sqrt(del*del'))*del; %step of size in
                    the direction of the tumble for bacterium i.
                    v= x(i,:);
                    v(v>ub)=ub;v(v<lb)=lb; %boundary check
                    x(i,:)=v;
                    J(i)=b(v);
                    m=0;
                    while m<Ns %counter for swim length
                        m=m+1;

                            if J(i)<Jlast(i) %choosing the better
value
                                Jlast(i)=J(i);
                                x(i,:)=x(i,:)+C*(del/sqrt(del*del'));
                                v= x(i,:);
                                v(v>ub)=ub;v(v<lb)=lb;
                                x(i,:)=v;
                                J(i)=b(v);
                            else
                                m=Ns; %end of swimming
                            end
                        end

                    end %end of Np

                    Jchem=[Jchem J];
    end % End of Chemotaxis % end of Nc

    for i=1:Np
        Jhealth(i)=sum(Jchem(i,:)); % sum of cost function of all
chemotactic loops for a given k & l
    end
    %Sort bacteria and chemotactic parameters
    [Jhealth,I]=sort(Jhealth, 'ascend'); %higher cost means lower health
    %The S bacteria with the highest health values die and the
    % remaining S bacteria with the best values split
    x=[x(I(1:Np/2),:);x(I(1:Np/2),:)];
    J=[J(I(1:Np/2),:);J(I(1:Np/2),:)];
    xmin=x(I(1),:); %best soloution so far

end %end of Reproduction
if iter>1
    if min(J)<Jmin(iter-1)
        [Jmin(iter),loc]=min(J);
        sol=x(loc,:);
    end
end

```

```

        else
            Jmin(iter)=Jmin(iter-1);
        end
    else
        [Jmin,loc]=min(J);
        sol=x(loc,:);
    end
    % random elimination dispersion
    % keeping the number of bacteria in the population constant
    for i=1:Np
        r=rand;
        if r>=Ped %elimination probability
            x(i,:)=rand(1,D)*(ub-lb)+lb; %random new position
            v= x(i,:);
            J(i)=b(v);
        end
    end
end %end of elimination Ne

```

در خط چهارم، طبق مرحله ۴ الگوریتم غذایانی باکتریایی، مرحله کمotaکسی آغاز شده است و در خط دوم طبق مرحله ۳ تولید مثل آغاز شده است و در خط اول طبق مرحله ۲ حذف پراکندگی آغاز می‌شود. پس از بروزرسانی موقعیت‌ها در هر مرحله بررسی می‌شود تا جواب‌های بدست‌آمده در بازه تعیین شده برای تابع هدف باشند و پس از آن نیز جواب بهینه را نمایش داده و طبق کد آ-۱ نمودارها را رسم می‌کنیم.

۲-۱-۳- مدل‌سازی الگوریتم با ورودی از قبل تعیین شده

برای بررسی بیشتر قدرت الگوریتم، داده‌های ورودی برای بهینه‌سازی را به صورت غیر تصادفی در ۸ حالت مختلف تقسیم نمودیم. موقعیت اولیه نقاط به ترتیب به دسته‌های مقابل تقسیم شدند: بالا_راست، قطر_فرعی، بالا_چپ، راست، قطر_اصلی، پایین_راست، پایین_راست و چپ به این معنی که از آنجایی که بازه‌ی فضای مورد بررسی در مسئله [-512, 512] می‌باشد، پس نقاطی که در دسته‌ی بالا راست قرار می‌گیرند در بازه‌ی [0, 512] چه در جهت x و چه در جهت y قرار دارند. سپس بدون تغییر پارامترهای مسئله، کد مربوط به این نوع از گرفتن داده را به شکل زیر می‌نویسیم.

برنامه ۲ - ۳: وارد کردن ورودی‌های برنامه و ایجاد حلقه‌ی اصلی

```

InitialVariables = {'top_right', 'top_left', 'Secondary_Axis', 'right',
'Main_Axis', 'left', 'down_right', 'down_left' };

for h = 1:numel(InitialVariables)
    % Load data from .mat file
    data = load(InitialVariables{h});
    %% initialization
    x=data.x;
    for i=1:Np %for plotting
        x11(i,1)=x(i,1);
        y11(i,1)=x(i,2);
    end
    for k=1:Np
        v= x(k,:); %initial population
        J(k)=CostFunction(v); % initial fitness calculation
    end
end

```

end

واضحاً از آنجایی که الگوریتم تغییری نکرده است و فقط در ورودی‌های الگوریتم تغییر ایجاد شده پس باقی قسمت‌های برنامه همانند بخش مدل‌سازی الگوریتم با ورودی تصادفی می‌باشد. تمام توضیحاتی که در گذشته داده شده است برروی این نحوه نگارش هم صدق می‌کند.

۲ - ۱ - ۲ الگوریتم بهینه‌سازی کرم شبتاب

نور چشمک زن کرم شبتاب منظره‌ای شگفت‌انگیز در آسمان تابستان در مناطق گرمسیری و معتدل است. حدود دو هزار گونه کرم شبتاب وجود دارد و بسیاری از این کرم‌ها پرتوهای کوتاه و ریتمیک تولید می‌کنند. الگوی چشمک‌ها اغلب برای هر گونه خاص منحصر به فرد است. نور چشمک زن توسط فرآیند بیولومینسانس^۱ تولید می‌شود و عملکرد واقعی سیستم‌های سیگنال‌دهی این چنینی هنوز در حال بحث است. با این حال، دو کارکرد اصلی شامل جذب ماده‌های جفت‌گیری و جذب طعمه بالقوه است. علاوه بر این، چشمک زدن ممکن است به عنوان یک مکانیزم هشدار محافظتی نیز عمل کند. چشمک زدن کرم شبتاب به سرعت هر دو جنس را به هم نزدیک می‌کند. ماده‌ها به الگویی منحصر به فرد پرواز گونه‌ی خود پاسخ می‌دهند. در حالی که در برخی از گونه‌ها مانند فوتوریس^۲، کرم شبتاب ماده می‌تواند الگوی چشمک زن دیگر گونه‌ها را تقلید کند تا کرم‌های شبتاب نر را که ممکن است به اشتباه به چشمک زدن به عنوان یک جفت بالقوه اشتباه بگیرند را فریب داده و بخورند. می‌دانیم که شدت نور^۳ در یک فاصله خاص از منبع نور (r) از

قانون مربع معکوس تبعیت می‌کند. یعنی با افزایش فاصله از منبع نور (r)، بر حسب $\frac{1}{r^2} I\alpha$ شدت نور (I) کاهش می‌یابد. علاوه بر این هوا نور را جذب می‌کند که با افزایش فاصله نور ضعیفتر می‌شود. این دو عامل ترکیبی باعث می‌شود که اکثر کرم‌های شبتاب فقط در فاصله‌ای محدود، معمولاً چند صد متر در شب، قابل مشاهده باشند. که معمولاً برای ارتباط کرم‌های شبتاب کافی است. نور چشمک زن را می‌توان به گونه‌ای فرمول‌بندی کرد که با تابع هدف بهینه‌سازی شده مرتبط باشد که امکان فرمول‌بندی الگوریتم‌های بهینه‌سازی جدید را فراهم می‌کند. برای سادگی در توصیف الگوریتم، اکنون فرض می‌کنیم که جذابیت را از قانون زیر استفاده می‌کنیم:

۱. جذب کرم‌های شبتاب

جذابیت هر کرم شبتاب با روشنایی آن متناسب است. بنابراین برای هر دو کرم شبتاب چشمک زن، کرم با

^۱ bioluminescence

^۲ photuris

^۳ Light Intensity

روشنایی کمتر به سمت روشن‌تر حرکت می‌کند. با افزایش فاصله کرم شدت نور تابیده شده به کرم دیگر کاهش می‌باید و در نتیجه جذابیت کرم شبتاب با دور شدن از کرم دیگر کاهش می‌باید. باید توجه داشت که اگر کرم شبتاب، کرم دیگری باشد نور بیشتری در اطراف خود نیابد بطور تصادفی شروع به حرکت در محیط می‌کند.

۲. روشانایی و هدف

روشنایی یک کرم شبتاب تحت تأثیر چشم انداز هدف قرار می‌گیرد یا تعیین می‌شود. برای یک مشکل بهینه‌سازی، روشانایی می‌تواند به سادگی با مقدار توابع هدف مناسب باشد.

۳. تغییر شدت نور و فرمول‌بندی جذابیت

در الگوریتم کرم شبتاب دو موضوع مهم وجود دارد: تغییر شدت نور و فرمول‌بندی جذابیت. برای سادگی، ما همیشه می‌توانیم فرض کنیم که جذابیت^۱ یک کرم شبتاب با روشانایی آن تعیین می‌شود که به نوبه خود با تابع هدف گذاری شده مرتبط است.

۲-۱-۱-۱ الگوریتم

در ساده‌ترین حالت برای حداقل مسائل، روشانایی یک کرم شبتاب(I) در یک مکان خاص(x) می‌تواند به صورت $I(x) \propto f(x)$ انتخاب شود. با اینحال، جذابیت نسبی است و باید در چشم بیننده دیده شود یا توسط سایر کرم‌های شبتاب قضاوت شود. بنابراین با فاصله r_{ij} بین کرم شبتاب i و کرم شبتاب j مقدارش تغییر می‌باید. علاوه بر این، شدت نور با فاصله از منبع کاهش می‌باید و نور نیز در رسانه جذب می‌شود، بنابراین باید اجازه دهیم جذابیت با درجه جذب متفاوت باشد. در ساده‌ترین شکل، شدت نور $I(r)$ بر اساس قانون مربع معکوس، $I(r) = \frac{I_s}{r^2}$ تغییر می‌کند که در آن I_s شدت در منع نور است. برای یک محیط با ضریب جذب نور ثابت، γ ، شدت نور I با فاصله r تغییر می‌کند. یعنی $I = I_o e^{-\gamma r}$ که در آن شدت نور منع، I_o می‌باشد. به منظور اجتناب از تکیدگی در $r = 0$ در عبارت I_s / r^2 اثر ترکیبی قانون مربع معکوس و جذب را می‌توان با استفاده از شکل گاوسی زیر تقریب زد.

$$I = I_o e^{-\gamma r^2} \quad (2-2)$$

از آنجایی که جذابیت کرم شبتاب مناسب با شدت نوری است که توسط کرم‌های شبتاب دیگر مشاهده می‌شود، اکنون می‌توانیم جذابیت کرم شبتاب را به صورت زیر در چارچوب ریاضیات نشان دهیم:

^۱ attractiveness

$$\beta(r) = \beta_0 e^{-\gamma r^2} \quad (3-2)$$

در این فرمول β_0 جاذبه در r_0 می‌باشد. اگر بسط تیلور تابع نمایی $e^{-\gamma r^2}$ را بنویسیم، متوجه می‌شویم که می‌توانیم بطور تقریبی در بسیاری از مسائل بجای عبارت $e^{-\gamma r^2}$ از جمله‌ی اول بسط تیلور یعنی $\frac{1}{1+r^2}$ استفاده کنیم. در این صورت فرمول محاسبه‌ی جذابیت با عبارت $\beta(r) = \frac{\beta_0}{1+\gamma r^2}$ (3-2) یک فاصله مشخصه $\Gamma = 1/\sqrt{\gamma}$ را تعریف می‌کند که جذابیت از $B_0 e^{-1}$ به B_0 تبدیل می‌کند. همچنین در فرمول ساده شده‌ی $\beta(\Gamma) = 0.36\beta_0$ اگر مقدار $\Gamma = r$ قرار دهیم آنگاه $\Gamma(\Gamma)$ نقطه‌ای است که جذابیت به نصف خود رسیده است و نقطه‌ی بحرانی $\beta(\Gamma) = 0.36\beta_0$ در نظر گرفته می‌شود. در پیاده‌سازی، شکل واقعی تابع جذابیت $\beta(r)$ می‌تواند هر تابع کاهشی یکنواخت مانند شکل تعیین یافته زیر باشد:

$$\beta(r) = \beta_0 e^{-\gamma r^m} \quad (m \geq 1) \quad (4-2)$$

برای γ ثابت، طول مشخصه هنگامی که $\Gamma = \gamma^{-1/m} \rightarrow 1$ می‌شود. در مقابل برای مقیاس طول معین، Γ در یک مسئله، پارامتر γ به عنوان یک مقدار اولیه معمولی انتخاب می‌شود. به این معنا که $\gamma = 1/\Gamma^m$ همچنین فاصله بین هر دو کرم شبتاب i و j در x_i و x_j با فرمول (5-2) محاسبه می‌شود.

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^d (x_{i,k} - x_{j,k})^2} \quad (5-2)$$

که در آن $x_{i,k}$ جمله k ام مختصات فضایی x_i ، برای کرم شبتاب i است.

$$r_{ij} = \|x_i - x_j\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

در حالت دو بعدی داریم: حرکت یک کرم شبتاب i که به کرم شبتاب درخشنان j و جذاب‌تر j جذب شده‌است، به صورت زیر مشخص می‌شود:

$$\begin{aligned} y_i &= y_i + \beta_0 e^{-\gamma r^2} (y_j - y_i) + \alpha (\text{rand} - \frac{1}{2}) \\ x_i &= x_i + \beta_0 e^{-\gamma r^2} (x_j - x_i) + \alpha (\text{rand} - \frac{1}{2}) \end{aligned} \quad (6-2)$$

این عبارت از ۳ قسمت تشکیل شده‌است: (از آنجایی که فرمول‌ها برای x و y یکسان هستند قسمت‌ها را در یکی از این

فرمول‌ها بررسی می‌کنیم و قابل تعمیم به فرمول دیگر هستند)

• x_i و y_j : این قسمت مشخصاً نقطه‌ی شروع یا موقعیت اولیه کرم شبتاب n ام می‌باشد.

• $(y_j - y_i)^{\beta_0 e^{-\gamma r^2}}$: این بخش فرمول تعیین کننده پارامتر شدت^۱ می‌باشد. به این معنا که این قسمت در تلاش

است تا موقعیت فعلی کرم شبتاب را با تمام سرعت به بهترین نقطه‌ی ممکن برحسب تابع هزینه نزدیک کند. در

نتیجه با افزایش γ می‌توانیم شدت الگوریتم را افزایش دهیم.

• $\alpha(rand - \frac{1}{2})$: این بخش فرمول حرکت، تعیین کننده پارامتر تنوع الگوریتم می‌باشد. به این معنا که در تلاش

است تا کرم شبتاب n ام تا جایی که امکان دارد در محیط مشخص شده جستجو کند. این بخش الگوریتم برای

جلوگیری از گیرافتادن الگوریتم در بهینه‌ی محلی است. در نتیجه این بخش در تلاش است تا بجای اینکه کرم

شبتاب n ام از مسیر مستقیم به موقعیت کرم شبتاب z برسد، با انحرافی از مسیر مستقیم حرکت کند تا نقاط

بیشتری یا بررسی و اکتشاف کند. با این تفاسیر می‌توان در نظر داشت که با افزایش المان α می‌توانیم مقدار تنوع

الگوریتم فرآیندهای کرم شبتاب را افزایش دهیم.

۲-۱-۲-۲ مدل‌سازی الگوریتم با ورودی تصادفی

در بخش مدل‌سازی مانند گذشته شروع به نوشتن کد الگوریتم در متلب می‌کنیم. در قسمت اول پارامترهای مورد نیاز را

طبق جدول ۲-۲ وارد می‌کنیم.

جدول ۲-۲: جدول پارامترهای الگوریتم بهینه‌سازی کرم شبتاب

پارامتر	مقدار
Range	-۵۱۲ _ ۵۱۲
population	۶۰
γ	۰.۰۱
β_0	۱
α	۰.۹
ضریب افت	۰.۹۷

برای انتخاب پارامترها از مقالات کمک گرفته شده و با آزمون خطا تلاش شده است تا این مقادیر بهینه شوند، همچنین

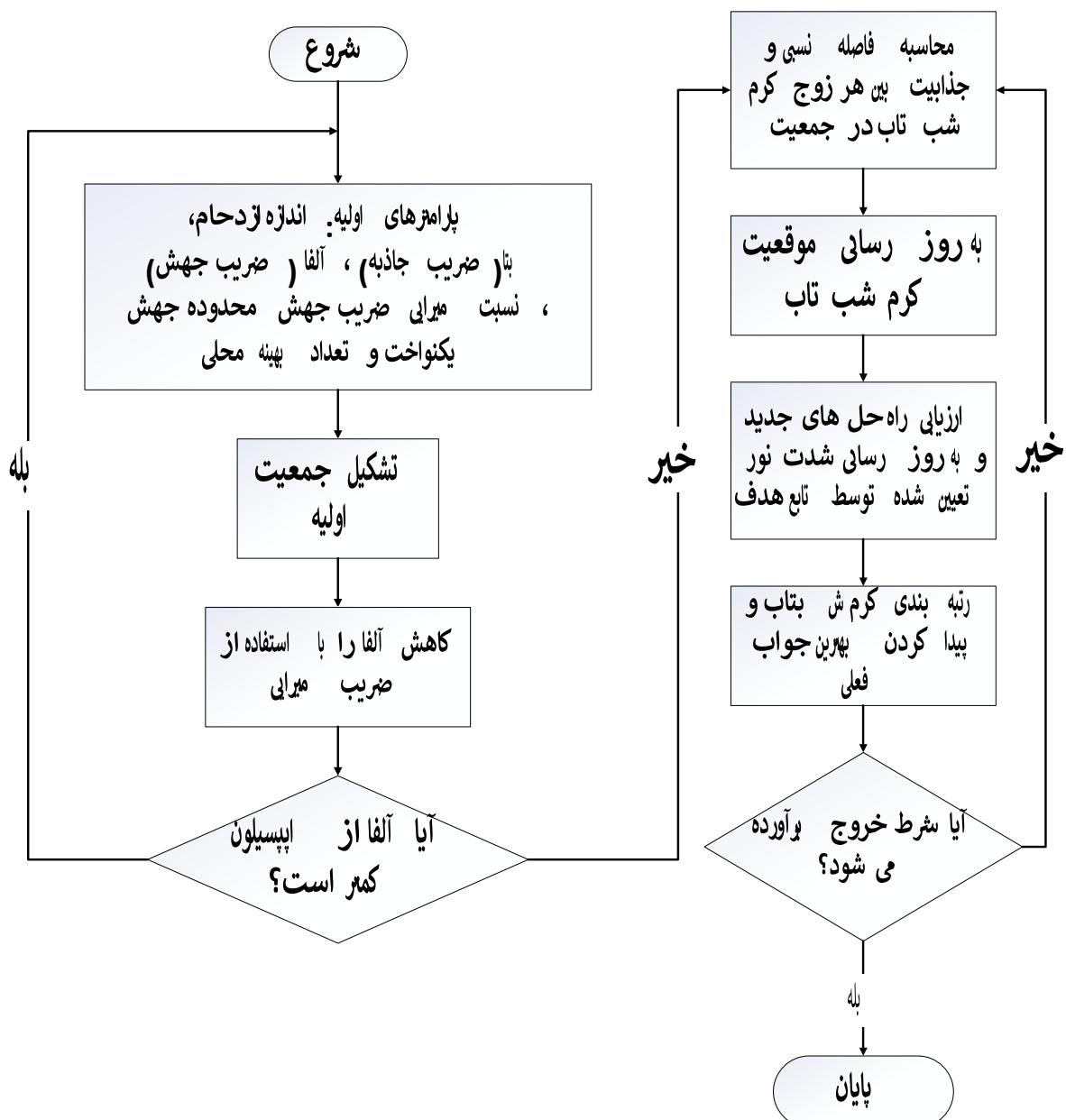
جمعیت تمام الگوریتم‌ها یکسان و به مقدار ۶۰ در نظر گرفته شده است تا بتوانیم مقایسه عادلانه‌ای داشته باشیم. حال در

بازه‌ی تعیین شده برای تابع هدف، جامعه‌ای با موقعیت تصادفی در فضای جستجو ایجاد می‌کنیم.

^۱ exploitation

برنامه ۲ - ۴: ایجاد جمعیت اولیه کرم‌های شب‌تاب

```
% initializing
% Generating the initial locations of n fireflies
for i=1:n
    ns(i,:)=Lb+(Ub-Lb).*rand(1,d);
    Lightn(i)=egg_holder(ns(i,1),ns(i,2));
end
% Randomization
% Evaluate objectives
```



شکل ۲ - ۲: فلوچارت الگوریتم بهینه‌سازی کرم شب‌تاب

در حلقه‌ی اصلی مربوط به الگوریتم از حلقه‌ی while ای برای رسیدن به دقت ۹۹٪ در نقطه مینیمم سراسری تابع شانه تخم مرغی استفاده می‌کنیم. در قسمت بعد یک مقیاس برای مقدار حرکت تصادفی و یک آلفا جدید با استفاده از ضریب

افت آلفا(برای کاهش پارامتر تونع در طول عملکرد الگوریتم و تکرارهای الگوریتم) ایجاد می‌کنیم. در ادامه قبل از شروع بهینه‌سازی، شرطی ایجاد می‌کنیم که در صورتی که همگرایی و بهینه‌سازی کد زمان بسیاری برداشته باشد و حلقه بیش از حد تکرار شده و α قابل چشم پوشی شده، جامعه اولیه دوباره به صورت تصادفی انتخاب شود و پارامترها به حالت اولیه خود بازگردند و پروسه از سر گرفته شود زیرا اگر α بیش از حد کوچک شود، کد وارد چرخه باطل شده و به نتیجه مطلوب نمی‌رسد.

برنامه ۲ - ۵: تنظیم دوباره پارامترها و جمعیت جدید

```
alpha=alpha*theta; % Reduce alpha by a factor theta
scale=abs(Ub-Lb); % Scale of the optimization problem
%% reset the whole initial swarm and alpha factor
if alpha<1e-4 %To avoid getting stuck in the loop
    alpha=0.9;
    for i=1:n
        ns(i,:)=Lb+(Ub-Lb).*rand(1,d); % Randomization
        Lightn(i)=egg_holder(ns(i,1),ns(i,2)); % Evaluate objectives
    end
end
```

سپس در حلقه اصلی الگوریتم با دستور for به تعداد جمعیت کرم‌های شبتاب ایجاد می‌شوند. در این الگوریتم جذابیت کرم شبتاب α و زبا یکدیگر مقایسه می‌شوند و اگر کرم شبتاب ز جذاب‌تر از α باشد، فاصله دو کرم شبتاب را با استفاده از معادله (۴-۲) و β را با استفاده از معادله (۵-۲) و همچنین گام حرکت را با استفاده از آلفا تغییر یافته که در برنامه (۴-۲) به آن اشاره شد و اعداد رندوم در بازه [۰.۵, ۰.۵] محاسبه می‌کنیم.

برنامه ۲ - ۶: حلقه اصلی الگوریتم

```
for i=1:n
    for j=1:n
        % Evaluate the objective values of current solutions
        Lightn(i)=egg_holder(ns(i,1),ns(i,2)); % Call the objective
        % Update moves
        if Lightn(i)>=Lightn(j) % Brighter/more attractive
            r=sqrt(sum((ns(i,:)-ns(j,:)).^2));
            beta=beta0*exp(-gamma*r.^2); % Attractiveness
            steps=alpha.*((rand(1,d)-0.5).*scale);
            % The FA equation for updating position vectors
            ns(i,:)=ns(i,:)+beta*(ns(j,:)-ns(i,:))+steps;

        end
    end % end for j
end % end for i
```

سپس باید پاسخ‌های حاصل از الگوریتم را بررسی کنیم که آیا در محیط تعیین شده [۵۱۲, ۵۱۲] هستند. در صورت اینکه جواب بدست آمده بالاتر از مراکزیم باشد مقدار مراکزیم بازه برای آن تعیین می‌شود و اگر پایین‌تر از بازه تعیین شده باشد، مقدار مینیم برای آن تعیین می‌شود. در مرحله آخر نیز موقعیت‌های جدید را توسط معادله (۶-۲) تعیین

کرده و نتیجه تابع هدف یا هزینه را بدست می‌آوریم.

برنامه ۲ - ۷: چک کردن مرزها و بدست آوردن نتایج

```

for i=1:n
    nsol_tmp=ns(i,:);
    % Apply the lower and upper bounds
    I=nsol_tmp<Lb;  nsol_tmp(I)=Lb(I);
    J=nsol_tmp>Ub; nsol_tmp(J)=Ub(J);
    % Update this new move
    ns(i,:)=nsol_tmp;
    Lightn(i)=egg_holder(ns(i,1),ns(i,2));
end

```

در نهایت نیز جواب‌های بدست آمده را با توجه به روشنایی آن‌ها که در واقع تابع هزینه برای این مساله می‌باشد مرتب سازی می‌کیم (از کم به زیاد) و از میان جواب‌ها به اندازه‌ی جمعیت (۶۰ عدد) انتخاب می‌کنیم و باقی را حذف می‌کنیم و بهترین جواب را نیز از میان این ۶۰ مورد انتخاب می‌کنیم. بعد از خروج از چرخه while با استفاده از نمودارهای تغییرات بهینه در طول اجرای کد و موقعیت اولیه و نهایی ذرات در فضای دو بعدی، نتایج نمایش داده می‌شوند.

برنامه ۲ - ۸: مرتب سازی جواب‌ها و نمایش جواب بهینه

```

%% Rank fireflies by their light intensity/objectives

[Lightn,Index]=sort(Lightn);
nsol_tmp=ns;
for i=1:n
    ns(i,:)=nsol_tmp(Index(i),:);
end
%% Find the current best solution and display outputs
fbest=Lightn(1); nbest=ns(1,:);
e=e+1;
mins=[mins fbest];

```

نحوه رسم نمودارها در پیوست آ-۱ است.

۲-۱-۳- مدل‌سازی الگوریتم با ورودی از قبل تعیین شده

برای بررسی بیشتر قدرت الگوریتم، داده‌های ورودی برای بهینه‌سازی را به صورت غیر تصادفی در ۸ حالت مختلف تقسیم نمودیم. موقعیت اولیه نقاط به ترتیب به دسته‌های مقابله‌ی تقسیم شدند: بالا_راست، قطر_فرعی، بالا_چپ، راست، قطر_اصلی، پایین_راست، پایین_راست_چپ به این معنی که از آنجایی که بازه‌ی فضای مورد بررسی در مسئله [۵12, ۵12] می‌باشد، پس نقاطی که در دسته‌ی بالا راست قرار می‌گیرند در بازه‌ی [۰, ۵12] چه در جهت x و چه در جهت y قرار دارند. سپس بدون تغییر پارامترهای مسئله، کد مربوط به این نوع از گرفتن داده را به شکل زیر می‌نویسیم.

برنامه ۲ - ۹: وارد کردن ورودی‌های برنامه و ایجاد حلقه‌ی اصلی

```
InitialVariables = {'top_right', 'top_left', 'Secondary_Axis', 'right',
```

```
'Main_Axis', 'left', 'down_right', 'down_left'};

for i = 1:numel(InitialVariables)
    %% Load data from .mat file
    data = load(InitialVariables{i});
    ns = double(data.x); % Assuming the variable 'x' contains the data you need
    Lightn(i)=egg_holder(ns(i,:)); % Evaluate objectives

    %% Initial swarm
    for j = 1:n
        x11(j, 1) = ns(j, 1);
        y11(j, 1) = ns(j, 2);
    end

```

واضحاً از آنجایی که الگوریتم تغییری نکرده است و فقط در ورودی‌های الگوریتم تغییر ایجاد شده پس باقی قسمت‌های برنامه همانند بخش مدل‌سازی الگوریتم با ورودی تصادفی می‌باشد و تمام توضیحاتی که در گذشته داده شده است بروی این نحوه نگارش هم صدق می‌کند.

۲ - ۱- ۳- الگوریتم جستجوی فاخته

در مقاله جستجوی فاخته^۱ توسط پرواز لوی^۲ از الگویتیمی دیگر برای بهینه‌سازی استفاده می‌شود که نام آن جستجوی فاخته است. هدف این گزارش فرموله کردن الگوریتم جدیدی به نام جستجوی فاخته است که براساس رفتارهای پرورشی جالب مانند انگلی نوزادان گونه‌های خاصی از فاخته‌ها است. ابتدا رفتار پرورش فاخته و ویژگی‌های پرواز لوی برخی از پرندگان و مگس میوه را معرفی می‌کنیم و سپس جستجوی فاخته جدید را فرموله می‌کنیم و به دنبال آن پیاده‌سازی می‌کنیم.

۲ - ۱- ۳- رفتار فاخته

فاخته‌ها، نه تنها به خاطر صدای زیبایی که می‌توانند تولید کنند، بلکه به دلیل استراتژی تولید مدل تهاجمی‌شان پرندگان جذابی هستند. برخی از گونه‌ها مانند فاخته‌های آنی^۳ و گویراء^۴ تخم‌های خود را در لانه‌های اشتراکی می‌گذارند، اگرچه ممکن است تخم‌های دیگر را برای افزایش احتمال جوچه شدن تخم‌های خود حذف کنند. تعداد زیادی از گونه‌ها با

^۱ Cuckoo search

^۲ Levy flight

^۳ ani

^۴ guira

تخمگذاری انگلی^۱ اجباری مبتلا می‌شوند. سه نوع اصلی تخمگذاری انگلی وجود دارد: انگل درون گونه‌ای، پرورش تعافی^۲ و تسخیر لانه^۳. برخی از پرندگان میزبان می‌توانند درگیری مستقیم با فاخته‌های مزاحم داشته باشند. اگر پرنده میزبان متوجه شود که تخم‌ها متعلق به او نیستند، یا این تخم‌های بیگانه را دور می‌اندازد یا به سادگی لانه خود را رها می‌کند و لانه جدیدی در جای دیگری می‌سازد. برخی از گونه‌های فاخته مانند تخم گذار انگلی تاپرا^۴ به گونه‌ای تکامل یافته‌اند که فاخته‌های انگلی ماده اغلب در تقلید رنگ و الگوی تخم‌های چندگونه میزبان منتخب بسیار حرفه‌ای هستند. این امر احتمال رها شدن تخم‌های آن‌ها را کاهش می‌دهد و در نتیجه باروری آنها را افزایش می‌دهد.

۲-۱-۳ پرواز لوی

از سوی دیگر، مطالعات مختلف نشان داده‌اند که رفتار پروازی بسیاری از حیوانات و حشرات، ویژگی‌های معمولی پروازهای لوی را نشان داده است. مطالعه اخیر توسط رینولدز و فرای نشان می‌دهد که مگس‌های میوه یا مگس‌های سرکه با استفاده از یک سری مسیرهای پرواز مستقیم که با یک پیچ ناگهانی ۹۰ درجه مشخص شده‌اند، مناظر خود را کاوش می‌کنند که منجر به الگوی جستجوی آزاد متناوب در مقیاس پرواز لوی می‌شود. مطالعات بروی رفتار انسان مانند الگوهای جستجوگر شکارچی-جمع آور نیز ویژگی معمول پروازهای لوی را نشان می‌دهد. حتی نور نیز می‌تواند به پروازهای لوی مربوط شود. متعاقباً، چنین رفتاری برای بهینه‌سازی و جستجوی بهینه اعمال شده‌است و نتایج اولیه قابلیت امیدوار کننده را نشان می‌دهد. نظریه و توزیع لوی (Levy Distribution) مفاهیم ریاضی و آماری هستند که برای مدل‌سازی پدیده‌هایی استفاده می‌شوند که دارای گام‌های بلند و تغییرات ناگهانی هستند. در این توزیع، برخلاف توزیع نرمال که بیشتر در اطراف میانگین تمرکز دارد، توزیع لوی دارای دُم‌های سنگین و بلند است که نشان‌دهنده احتمال بالای رخدادهای نادر و شدید است. این ویژگی‌ها توزیع لوی را برای مدل‌سازی بسیاری از فرآیندهای طبیعی و انسانی که شامل نوسانات شدید و غیرمنتظره هستند، مناسب می‌سازد.

۲-۱-۳-۳ ویژگی‌های توزیع لوی

• **دُم‌های بلند:** توزیع لوی دارای دم‌های بلند است که نشان‌دهنده احتمال وقوع بالای رویدادهای نادر و بسیار بزرگ است. این ویژگی برای مدل‌سازی پدیده‌هایی مانند تغییرات شدید قیمت در بازارهای مالی یا نوسانات شدید در

^۱ Brood parasitism

^۲ Intraspecific brood parasitism

^۳ Nest takeover

^۴ Tapera

داده‌های محیطی بسیار مفید است.

- بدون میانگین و واریانس محدود: برخلاف توزیع نرمال، توزیع لوی دارای میانگین و واریانس نامحدود است. این بدان معنی است که میانگین و واریانس برای این توزیع تعریف نمی‌شوند، که نشان‌دهنده تغییرات شدید و غیرقابل پیش‌بینی است.
- خود شبیه به خود: توزیع لوی دارای ویژگی «خود شبیه به خود» است، به این معنی که اگر مقیاس زمانی یا مکانی تغییر کند، توزیع کلی همچنان شبیه به خود باقی می‌ماند. این ویژگی در مدل‌سازی رفتارهای فرکتالی (Fractal) نیز دیده می‌شود.

۱-۳-۴ الگوریتم

برای سادگی در توصیف جستجوی فاخته جدید از سه قانون ایده آل زیر استفاده می‌کنیم:

- هر فاخته هر بار یک تخم می‌گذارد و آن را در لانه‌ای که بطور تصادفی انتخاب شده، می‌ریزد.
- بهترین لانه‌ها با کیفیت بالای تخم (جواب) به نسل‌های بعدی منتقل می‌شود.
- تعداد لانه‌های میزبان موجود ثابت است و یک میزبان می‌تواند یک تخم بیگانه را با احتمال $P_a \in [0,1]$ کشف کند. در این حالت، پرنده میزبان می‌تواند تخم‌ها را دور بیندازد یا لانه را رها کند تا یک لانه کاملاً جدید در مکانی جدید بسازد.

برای سادگی، این فرض اخر را می‌توان با کسر P_a از n لانه که با لانه‌های جدید جایگزین می‌شود (با راه حل‌های تصادفی جدید در مکان‌های جدید) تقریب زد.

هنگام ایجاد راه حل‌های جدید $x^{(t+1)}$ برای فاخته‌نام، پرواز لوی مطابق فرمول زیر انجام می‌شود:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \otimes Le'vy(\lambda) \quad (7-2)$$

که در آن $\alpha > 0$ اندازه گام است که باید با مقیاس‌های مسئله مورد نظر مرتبط باشد. در بیشتر موارد، می‌توانیم از $\alpha = 0$ استفاده کنیم. حاصل ضرب \otimes به معنای ضرب‌های ورودی است. پروازهای لوی اساساً یک پیاده‌روی تصادفی ارائه می‌دهند در حالی که مراحل تصادفی آنها از توزیع لوی برای گام‌های بزرگ‌ترسیم می‌شود که دارای واریانس نامتناهی یا بی‌نهایت است.

$$Le'vy \sim u = t^{-\lambda} \quad (1 < \lambda \leq 3) \quad (8-2)$$

در اینجا جهش‌ها/ گام‌های متوالی فاخته اساساً فرآیند راه رفتمن تصادفی را تشکیل می‌دهند که از توزیع طول پله توانی

دم بلند^۱ تبعیت می‌کند. شایان ذکر است که در دنیا واقعی، اگر تخم فاخته بسیار شبیه تخم‌های میزبان باشد، احتمال کشف تخم فاخته کمتر است، بنابراین شایستگی باید به تفاوت در جواب‌ها مرتبط باشد. بنابراین، ایده خوبی است که یک پیاده روی تصادفی را به روشنی مغرضانه با اندازه‌های تصادفی گام انجام داد.

۲-۱-۳-۵ پرواز لوی

در این بخش با استفاده از کتاب الگوریتم‌های فرآیند کاری الهام گرفته از طبیعت، روش محاسبه پرواز لوی توضیح داده شده است. به طور کلی، پروازهای لوی یک پیاده روی تصادفی هستند که طول گام آن از توزیع لوی، اغلب بر حسب یک فرمول ساده قانون توان $L(s) \sim |s|^{-1-\beta}$ که در آن $0 < \beta \leq 2$ یک شاخص است، برداشت می‌شود. از نظر ریاضی، یک نسخه ساده از توزیع لوی را می‌توان به این صورت تعریف کرد:

$$L(s, \gamma, \mu) = \begin{cases} \sqrt{\frac{\gamma}{2\pi}} \exp\left[-\frac{\gamma}{2(s-\mu)}\right] \frac{1}{(s-\mu)^{3/2}} & 0 < \mu < s < \infty \\ 0 & \text{otherwise} \end{cases} \quad (9-2)$$

که در آن $\mu < 0$ یک گام حداقل و γ پارامتر مقیاس است. همانطور که $\gamma \rightarrow \infty$ داریم:

$$L(s, \gamma, \mu) \approx \sqrt{\frac{\gamma}{2\pi}} \frac{1}{s^{3/2}} \quad (2-10)$$

این یک مورد خاص از توزیع لوی تعمیم یافته است. به طور کلی، توزیع لوی باید بر اساس تبدیل فوریه تعریف شود:

$$F(k) = \exp[-\alpha |k|^\beta], \quad 0 < \beta \leq 2 \quad (11-2)$$

جایی که α یک پارامتر مقیاس است. معکوس این انتگرال آسان نیست، زیرا به جز چند مورد خاص، شکل تحلیلی ندارد. برای حالت $\beta = 2$ داریم:

$$F(k) = \exp[-\alpha k^2] \quad (12-2)$$

که تبدیل فوریه معکوس آن با توزیع گاوی مطابقت دارد. مورد خاص دیگر $\beta = 1$ است و ما داریم:

$$F(k) = \exp[-\alpha |k|] \quad (13-2)$$

که مربوط به توزیع کوشا است.

$$p(x, \gamma, \mu) = \frac{1}{\pi} \frac{\gamma}{\gamma^2 + (x - \mu)^2} \quad (14-2)$$

که μ پارامتر مکان است، در حالی که γ مقیاس این توزیع را کنترل می‌کند. برای حالت کلی، انتگرال معکوس زیر را

^۱ Power-law step-length distribution with a heavy tail

داریم:

$$L(s) = \frac{1}{\pi} \int_0^{\infty} \cos(ks) \exp[-\alpha |k|^{\beta}] dk \quad (15-2)$$

اگر فرض کنیم که $s \rightarrow \infty$ آنگاه می‌توانیم معادله حرکت را به فرم زیر بنویسیم.

$$L(s) \rightarrow \frac{\alpha \beta \Gamma(\beta) \sin(\pi \beta / 2)}{\pi |s|^{1+\beta}}, s \rightarrow \infty \quad (16-2)$$

در اینجا $\Gamma(z)$ تابع گاما است:

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt \quad (17-2)$$

در صورتی که $z = n$ یک عدد صحیح باشد، $\Gamma(n) = (n-1)!$ داریم. پروازهای لوی در کاوش فضای جستجوی ناشناخته و در مقیاس بزرگ کارامدتر از پیاده روی تصادفی هستند. دلایل زیادی برای توضیح این کارایی وجود دارد یکی از آنها این است که واریانس پروازهای لوی ذیل بسیار سریع‌تر از رابطه خطی (یعنی $\sim t$) ($\sigma^2(t)$) پیاده روی تصادفی براونی افزایش می‌یابد.

$$\sigma^2(t) \sim t^{3-\beta}, 1 \leq \beta \leq 2 \quad (18-2)$$

از نقطه نظر اجرا، تولید اعداد تصادفی با پروازهای لوی شامل دو مرحله است: انتخاب جهت تصادفی و تولید مراحلی که از توزیع انتخابی لوی تبعیت می‌کنند. تولید یک جهت باید از یک توزیع یکنواخت گرفته شود، در حالی که تولید مراحل بسیار مشکل است. چند راه برای دستیابی به این امر وجود دارد، اما یکی از کارآمدترین و در عین حال ساده‌ترین راه‌ها استفاده از الگوریتم موسوم به متگنا^۱ برای توزیع پایدار لوی متقارن است. در اینجا "متقارن" به این معنی است که مراحل می‌توانند مثبت و منفی باشند. یک متغیر تصادفی U و توزیع احتمال آن را می‌توان پایدار نامید اگر ترکیب خطی دو نسخه یکسان آن (یا U_1 و U_2) از توزیع یکسانی پیروی کند. یعنی $aU_1 + bU_2$ توزیعی مشابه با $cU + d$ دارد که در آن $a, b > 0$ و $c, d \in \mathbb{R}$ اگر $d = 0$ باشد، کاملاً پایدار نامیده می‌شود. توزیع‌های گوسی، کوشی و لوی همگی توزیع‌های پایدار هستند. در الگوریتم متگنا، طول گام s را می‌توان با استفاده از معادله زیر به دست آورد:

$$s = \frac{u}{|v|^{1/\beta}} \quad (19-2)$$

جایی که u و v از توزیع‌های نرمال گرفته شده‌اند. به این معنا که:

$$u \sim N(0, \sigma_u^2), \quad v \sim N(0, \sigma_v^2) \quad (20-2)$$

که در آن

¹ Mantegna

$$\sigma_u = \left\{ \frac{\Gamma(1+\beta) \sin(\pi\beta/2)}{\Gamma[(1+\beta)/2] \beta 2^{(\beta-1)/2}} \right\}^{\frac{1}{\beta}}, \sigma_v = 1 \quad (21-2)$$

این توزیع (برای s) از توزیع لوی مورد انتظار برای s_0 پیروی می‌کند که در آن s_0 کوچکترین گام است. در اصل s_0 ، اما در واقعیت s_0 را می‌توان به عنوان یک مقدار معقول مانند $s_0 = 0.1$ تا 1 در نظر گرفت. مطالعات نشان می‌دهد که پروازهای لوی می‌توانند کارایی جستجوی منابع در محیط‌های نامشخص را به حداقل برسانند. در واقع، پروازهای لوی در میان الگوهای جستجوگر غذای آلباتروس و مگس میوه و میمون عنکبوتی مشاهده شده‌است. علاوه بر این، پروازهای لوی کاربردهای زیادی دارند. بسیاری از پدیده‌های فیزیکی مانند انتشار مولکول‌های فلورسنت، رفتار خنک کننده و نویز می‌توانند ویژگی‌های پرواز لوی را در شرایط مناسب نشان دهند.

۱-۲-۳-۶ مدل‌سازی با ورودی تصادفی

برای صحه‌گذاری عملکرد این الگوریتم توسط تابع هزینه شانه تخم مرغی ابتدا الگوریتم را در مطلب، مدل‌سازی می‌کنیم. در ادامه توضیحی مختصر از کد مدل‌سازی آورده شده است. پارامترهای تعیین شده در بخش اول کد در جدول ۲ - ۳ آورده شده است.

جدول ۲ - ۳: پارامترهای الگوریتم غذایابی باکتریایی

مقدار	پارامتر
-۵۱۲ _ ۵۱۲	Range
۶۰	population
۳۰	P_a

همانند الگوریتم‌های غذایابی باکتریایی و الگوریتم کرم شبتاب در ابتدا باید، لانه‌های اولیه را به صورت تصادفی موقعیت‌دهی کنیم و شایستگی آن‌ها را محاسبه نمائیم.

برنامه ۲ - ۱۰: ایجاد جمعیت اولیه لانه‌ها در الگوریتم جستجوی فاخته

```
%>>> %% initialization
for i = 1:n
    % Generate random solutions within the lower half of the search space
    lower_half_bounds = (Ub - Lb) / 2; % Define the bounds for the lower half
    nest(i,:) = Lb + lower_half_bounds .* rand(size(Lb));
end

% Get the current best
fitness=10^10*ones(n,1); %preallocating and making sure the initial state will
be changed
[fmin,bestnest,nest,fitness]=get_best_nest(nest,nest,fitness);
```

در اینجا مشاهده می‌شود که برای محاسبه‌ی تابع هزینه (شایستگی) هر لانه (جواب) جدید و جایگزینی لانه قدیمی در صورت نیاز از تابع `get_best_nest` استفاده می‌شود.

برنامه ۲ - ۱۱: تابع محاسبه بهترین لانه

```
%% Find the current best nest
function [fmin,best,nest,fitness]=get_best_nest(nest,newnest,fitness)
% Evaluating all new solutions
for j=1:size(nest,1)
    fnew=egg_holder(newnest(j,:));
    if fnew<=fitness(j)
        fitness(j)=fnew;
        nest(j,:)=newnest(j,:);
    end
end
% Find the current best
[fmin,K]=min(fitness) ;
best=nest(K,:);
```

در این تابع با وارد کردن موقعیت‌های لانه‌های جدید و قدیم و محاسبه‌ی تابع هزینه بر حسب موقعیت‌های این لانه‌ها می‌توانیم به مقایسه‌ی مقدار تابع هزینه این موقعیت‌ها بپردازم و اگر لانه‌های جدید، جواب بهتری برای مسئله ما باشند جایگزین لانه‌های قدیمی می‌شوند. در نهایت نیز بهترین جواب را به عنوان best به کد اصلی منتقل می‌کنیم. در قسمت بعد پرواز لوی را طبق معادلات [\(۱۹-۲\)](#) و [\(۲۰-۲\)](#) اجرا کرده و لانه‌های جدید را بعد از پرواز لوی ایجاد می‌کنیم و در

لانه‌ها قرار می‌دهیم.

برنامه ۲ - ۱۲: تابع پرواز لوی

```
%% Get cuckoos by random walk
function nest=get_cuckoos(nest,best,Lb,Ub)
n=size(nest,1);
beta=3/2;
sigma=(gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-1)/2)))^(1/beta);
for j=1:n
    s=nest(j,:);

    u=randn(size(s))*sigma;
    v=randn(size(s));
    step=u./abs(v).^(1/beta);
    stepsize=0.01*step.* (s-best);
    s=s+stepsize.*randn(size(s));
    nest(j,:)=simplebounds(s,Lb,Ub);
end
```

در تابع پرواز لوی، همانطور که مشاهده می‌شود ابتدا سیگما محاسبه شده و از روی آن u و v و گام برای هر لانه محاسبه می‌شود. لانه جدید پس از پرواز لوی ایجاد شده در آخر بررسی می‌شود تا بین مزهای تابع هزینه باشند. در قسمت بعد نیز مانند کد [\(۹-۲\)](#) شایستگی را بررسی کرده و در صورتی که وضعیت لانه جدید بهتر بود جایگزین لانه قبل می‌شود. در مرحله آخر بهینه‌سازی حرکتی تصادفی با احتمال اکتشاف P_e صورت می‌گیرد زیرا در دنیای واقعی، اگر تخم فاخته

بسیار شبیه به تخم یک میزبان باشد، احتمال کشف این تخم فاخته کمتر است، بنابراین شایستگی باید به تفاوت جواب‌ها مرتبط باشد. به بیان دیگر، الگوریتم‌های بهینه‌سازی فراباتکاری باید از تنوع خوبی در کنار شدت برخوردار باشند تا در مینیمم‌های محلی گیر نکنند و بتوانند کل محیط و پاسخ‌های محتمل را به خوبی بررسی کنند. در نتیجه، ایده خوبی است که یک پیاده‌روی تصادفی را به روشنی معرفی کنیم.

برنامه ۲ - ۱۳: لانه‌های خالی کشف شده

```
function new_nest=empty_nests(nest,Lb,Ub,pa)
n=size(nest,1);
K=rand(size(nest))>pa;
stepsize=rand*(nest(randperm(n),:)-nest(randperm(n),:));
new_nest=nest+stepsize.*K;
for j=1:size(new_nest,1)
    s=new_nest(j,:);
    new_nest(j,:)=simplebounds(s,Lb,Ub);
end
```

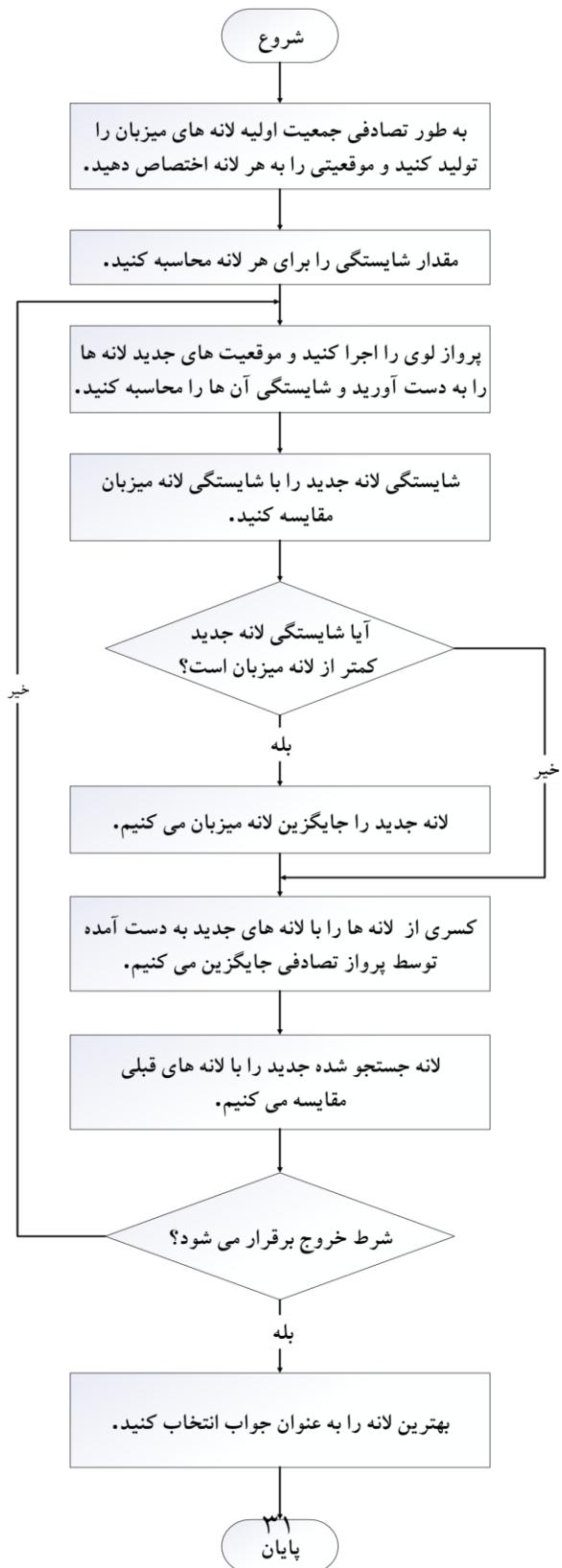
حال دیگر به سراغ حلقه‌ی اصلی الگوریتم می‌رویم. در این حلقه از تمام توابع توضیح داده شدن و لانه‌های اولیه استفاده می‌شود. این حلقه با دستور while برای تکرار تا جایی که به دقت ۹۹٪ در پیدا کردن نقطه‌ی مینیمم سراسری تابع شانه تخم مرغی بررسیم، نوشته شده است.

برنامه ۲ - ۱۴: حلقه اصلی الگوریتم جستجوی فاخته

```
% Generate new solutions (but keep the current best)
new_nest=get_cuckoos(nest,bestnest,Lb,Ub);
[~,~,nest,fitness]=get_best_nest(nest,new_nest,fitness);
% Discovery and randomization
new_nest=empty_nests(nest,Lb,Ub,pa) ;

% Evaluate this set of solutions
[fnew,best,nest,fitness]=get_best_nest(nest,new_nest,fitness);
% Find the best objective so far
if fnew<fmin
    fmin=fnew;
    bestnest=best;
end
mins1=[mins1 fmin];
e=e+1;
```

در انتها نیز مشابه سایر الگوریتم‌ها برای بررسی نتایج اجرای کد، نمودارهای لانه‌های اولیه و نهایی در فضای سه بعدی و نمودار تغییر کمینه هر چرخه رسم می‌شود که روش رسم آن در پیوست آ_۱ توضیح داده شده است.



شکل ۲ - ۳: فلوچارت الگوریتم بهینه‌سازی جستجوی فاخته

۲-۱-۳-۷ مدل‌سازی الگوریتم با ورودی از قبل تعیین شده

برای بررسی بیشتر قدرت الگوریتم، داده‌های ورودی برای بهینه‌سازی را به صورت غیر تصادفی در ۸ حالت مختلف تقسیم نمودیم. موقعیت اولیه نقاط به ترتیب به دسته‌های مقابل تقسیم شدند: بالا_راست، قطر_فرعی، بالا_چپ، راست، قطر_اصلی، پایین_راست، پایین_راست، چپ به این معنی که از آنجایی که بازه‌ی فضای مورد بررسی در مسئله $[512, 512]$ می‌باشد، پس نقطی که در دسته‌ی بالا راست قرار می‌گیرند در بازه‌ی $[0, 512]$ چه در جهت x و چه در جهت y قرار دارند. سپس بدون تغییر پارامترهای مسئله، کد مربوط به این نوع از گرفتن داده را به شکل زیر می‌نویسیم.

برنامه ۲ - ۱۵: وارد کردن ورودی‌های برنامه و ایجاد حلقه‌ی اصلی

```
InitialVariables = {'top_right', 'top_left', 'Secondary_Axis', 'right',
'Main_Axis', 'left', 'down_right', 'down_left'};
for i = 1:numel(InitialVariables)
    % Load data from .mat file
    data = load(InitialVariables{i});
    nest = double(data.x); % Assuming the variable 'x' contains the data you
need

function [best, fmin, elapsed_time, x11, y11, mins1] = cuckoo_optimization(nest,
n, pa, Lb, Ub, Tol, mins1, e)
% Get the current best
fitness = 10^10 * ones(n, 1); %preallocating and making sure the initial state
will be changed
[fmin, bestnest, nest, fitness] = get_best_nest(nest, nest, fitness);
for i = 1:n
    x11(i, 1) = nest(i, 1);
    y11(i, 1) = nest(i, 2);
end %for plotting the initial nests
```

واضحاً از آنجایی که الگوریتم تغییری نکرده است و فقط در ورودی‌های الگوریتم تغییر ایجاد شده پس باقی قسمت‌های برنامه همانند بخش مدل‌سازی الگوریتم با ورودی تصادفی می‌باشد و تمام توضیحاتی که در گذشته داده شده است بر روی این نحوه نگارش هم صدق می‌کند.

۲-۱-۴ الگوریتم امپریالیست رقابتی

در این بخش به بررسی استفاده از الگوریتم امپریالیست رقابتی برای حل سیستم‌های معادلات غیر خطی می‌پردازیم. الگوریتم امپریالیست رقابتی یک الگوریتم تکاملی جدید برای بهینه‌سازی است که از رقابت امپریالیستی الهام گرفته شده است. بد نیست اشاره کنیم که این الگوریتم روشی مستحکم مبتنی بر امپریالیسم است که سیاست گسترش قدرت و

حاکمیت یک دولت به خارج از مرزهای خود می‌باشد. در این الگوریتم با یک جمعیت اولیه به عنوان کشورهای اولیه شروع می‌کنیم. برخی از بهترین کشورها در میان جمعیت به عنوان امپریالیست انتخاب شده‌اند. بقیه جمعیت بین امپریالیست‌های ذکر شده به عنوان مستعمره تقسیم می‌شوند. سپس رقابت امپریالیستی در میان همه امپراطوری‌ها آغاز می‌شود. ضعیفترین امپراطوری که نتواند قدرت خود را افزایش دهد و در این رقابت موفق شود، از رقابت حذف می‌شود. روند حذف این امپراطوری به این صورت است که در هر دوری که الگوریتم عمل می‌کند یکی از مستعمره‌های خود را از دست می‌دهد، این فرآیند تا جایی ادامه پیدا می‌کند که دیگر مستعمره‌ای نداشته باشد و خود نیز به عنوان یک مستعمره به امپراطوری دیگری ملحق شود. همچنین در هین رقابت بروز امپراطوری، رقابتی درون امپراطوری بین امپراطور و مستعمره‌های خود وجود دارد. این رقابت به این شکل است که هر کدام از مستعمرات که بتواند قدرت بیشتری نسبت به امپراطور خود پیدا کند دست به انقلاب می‌زند و خود امپراطور جدید می‌شود. در نهایت امیدواریم که مکانیزم فروپاشی باعث شود همه کشورها به حالتی همگرا شوند که در آن فقط یک امپراطوری در جهان وجود داشته باشد و همه کشورهای دیگر، مستعمره آن امپراطوری واحد باشند. این امپراطوری یکپارچه و قدرتمند، امپراطوری دارد که جواب مسئله ما را مشخص می‌کند.

۲-۱-۴-۱ ایجاد امپراطوری‌های اولیه

یافتن راه حل بهینه هدف بهینه‌سازی است. ما کشورهای خود را تولید می‌کنیم که راه حل‌های تصادفی شده به عنوان جمعیت هستند. این کشورهای اولیه (جواب‌های اولیه) با توجه به بعد مسئله دچار تغییر می‌شوند. در مسئله مورد نظر ما از آنجایی که دو بعدی است، کشور نیز یک ارایه $N \times 2$ می‌باشد، که N نیز تعیین کننده جمعیت پاسخ‌ها می‌باشد.

$$country = (x_1, x_2, \dots, x_n) \quad x_i \in \mathbb{R} \quad 1 \leq i \leq N \quad (22-2)$$

باید به مقدار N_{pop} از آنها تولید کنیم. سپس با استفاده ازتابع هزینه، مقدار هزینه‌ی هر کدام از پاسخ‌ها را محاسبه می‌کنیم.

$$cost = f(country) = f(x_1, x_2, \dots, x_n) \quad (23-2)$$

حال به سراغ تعیین امپراطورها می‌رویم. تعداد امپراطورها را با N_{imp} نشان می‌دهیم. این جمعیت در واقع بهترین جواب‌ها از میان کشورهای اولیه هستند که انتخاب می‌شوند. اگر جمعیت مستعمرات را با N_{col} نشان دهیم، آنگاه جمعیت امپراطوری‌ها به شکل مقابل محاسبه می‌شوند، سپس باید جمعیت مستعمرات را بین امپراطورها تقسیم کنیم تا امپراطوری‌ها را شکل دهیم. توجه داشته باشید که هزینه‌ی یک امپریالیست را باید نرمالایز کنیم و از طریق معادله زیر آن را محاسبه می‌کنیم.

$$C_n = c_n - \max_i \{c_i\} \quad (24-2)$$

که در آن c_n هزینه‌ی n امین امپریالیست و C_n هزینه‌ی عادی شده‌ی آن می‌باشد. قدرت (احتمال) هر امپریالیست با

معادله ذیل تعریف می‌شود:

$$P_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right| \quad (25-2)$$

با محاسبه‌ی قدرت هر امپریالیست می‌توانیم تعداد مستعمره‌هایی که به آن تعلق می‌گیرد را با فرمول زیر محاسبه کنیم.

$$No.C_n = round(p_n.N_{col}) \quad (26-2)$$

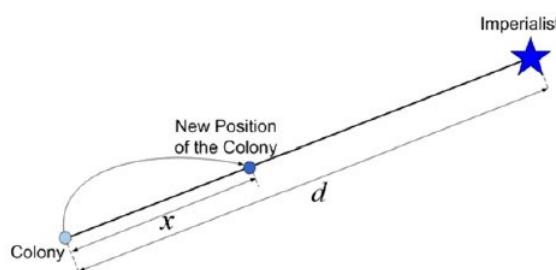
که در آن $No.C_n$ تعداد اولیه مستعمرات امپراطوری n و N_{col} تعداد کل کشورهای مستعمره می‌باشد. برای تقسیم مستعمرات بین امپریالیست‌ها، بطور تصادفی به مقدار $No.C_n$ از مستعمرات را انتخاب می‌کنیم و آن‌ها را به امپراطوری n ام نسبت می‌دهیم.

۲-۱-۴-۲ حرکت مستعمرات به سمت امپراطور خود

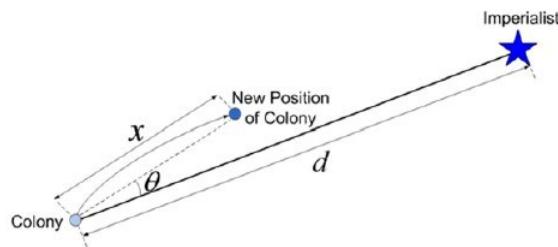
از آنجایی که امپریالیست هر مستعمره کوچک‌ترین مقدار هزینه را دارد، مستعمره‌ها به سمت او حرکت می‌کنند تا بتوانند به موقعیت بهتر (موقعیت امپراطور) برسند. هر چند این حرکت نباید بطور مستقیم و در یک گام صورت گیرد زیرا اصل تنوع از بین می‌رود پس هر مستعمره به مقدار x واحد در جهت امپراطور حرکت می‌کند.

$$x \sim U(0, \beta \times d), \quad \beta > 1 \quad (27-2)$$

در معادله‌ی (۲۷-۲)، d فاصله‌ی بین امپراطور و مستعمره مورد بررسی است، β باعث نزدیک شدن مستعمره به امپریالیست می‌شود. در پروژه‌ی مورد بررسی $\beta = 2$ ثابت است. در شکل ۲ - ۴ و شکل ۲ - ۵ نشان داده‌ایم که برای جستجوی اصولی اطراف امپریالیست، باید مقدار تصادفی انحراف را به جهت حرکت اضافه کنیم. مقدار انحراف را نیز با θ نشان می‌دهیم.



شکل ۲ - ۴: حرکت مستعمرات به سمت امپریالیست مربوطه خود



شکل ۲ - ۵: حرکت مستعمرات به سمت امپریالیست مربوطه خود در جهت تصادفی انحرافی

۲-۱-۴-۳- انقلاب^۱

در هر تکرار، تعدادی از مستعمرات در یک امپراطوری با همان تعداد کشورهای جدید ایجاد شده جایگزین می‌شوند. این کار برای ایجاد کشور جدید و جایگزینی آنها با برخی مستعمرات امپراطوری به صورت تصادفی صورت می‌گیرد تا پارامتر تنوع الگوریتم حفظ شود. این اقدام انقلاب نامیده می‌شود که در این الگوریتم نقش حساسی دارد. تعداد مستعمرات امپراطوری که قرار است با همین تعداد کشورهای جدید ایجاد شود، با فرمول زیر محاسبه می‌شود.

$$N.R.C = \text{round} \{ \text{RevolutionRate} \times \text{No.}(\text{the colonies of empire}_n) \} \quad (28-2)$$

$N.R.C$ تعداد مستعمرات انتخاب شده برای اعمال فرآیند انقلاب بر روی آنها می‌باشد. این امر همگرایی جهانی الگوریتم را بهبود می‌بخشد و از گیر کردن آن به کمینه محلی جلوگیری می‌کند.

۲-۱-۴-۴- رقابت درون امپراطوری

با جابجایی، یک مستعمره ممکن است مستعمره به موقعیت بهتری نسبت به موقعیت امپریالیست‌ها دسترسی داشته باشد. پس این مستعمره نقش امپریالیست امپراطوری را بدست می‌آورد و امپریالیست قبلی به مستعمره تبدیل می‌شود. در نتیجه، تمام مستعمرات حرکت خود را به سمت امپریالیست جدید آغاز می‌کنند.

۲-۱-۴-۵- قدرت کل یک امپراطوری

^۱ Revolution

هر امپراطوری تشکیل شده از یک امپریال و چند مستعمره می‌باشد که تعداد آن نیز در تمام امپراطوری‌ها یکسان نیست. برای محاسبه قدرت این امپراطوری باید مجموع قدرت امپریال و مستعمراتش را در نظر گرفت، هرچند که امپریال نشان‌دهنده قدرتمندترین کشور موجود در امپراطوری است، پس باید به مستعمرات دیگر ضریبی داد تا تاثیر آنها کمتر از امپریال باشد.

$$T.C_n = \cos t(imperialist_n) + \zeta \cdot mean(\cos t(colonies_of_empire_n)) \quad (29-2)$$

که ضریب موقعیت کلونی‌ها می‌باشد که تاثیر قدرت کلونی‌ها بر قدرت امپراطوری را تعیین می‌کند.

۲-۱-۴-۶ رقابت امپریالیستی

در رقابت بین امپراطوری‌ها، امپراطوری‌ها در تلاش هستند که تا جایی که می‌شود قدرت خود را افزایش دهند. هر چند در این رقابت همواره برخی از امپراطوری‌ها در حال ضعیفتر شدن و برخی دیگر در حال قدرتمندتر شدن هستند. این قدرت تعیین کننده احتمال هر امپراطوری برای تصاحب کلونی جدا شده از ضعیفترین امپراطوری است. در زیر فرمول نرمال شده قدرت هر امپراطوری را نشان داده‌ایم.

$$N.T.C_n = T.C_n - \max\{T.C_i\} \quad (30-2)$$

حال با محاسبه قدرت نرمالایز شده می‌توانیم احتمال تصاحب کلونی جدا شده از ضعیفترین امپراطوری را برای تمام امپراطوری‌ها به فرم زیر محاسبه کنیم.

$$P_{pn} = \left| \frac{N.T.C_n}{\sum_{i=1}^{N_{imp}} N.T.C_n} \right| \quad (31-2)$$

مستعمرات مذکور را براساس احتمال تصاحب آنها بین امپراطوری‌ها تقسیم می‌کنیم. پس بردار احتمالات به‌شکل زیر نشان داده می‌شود.

$$P_{pn} = |P_{p_1}, P_{p_2}, P_{p_3}, \dots, P_{p_{N_{imp}}} | \quad (32-2)$$

و همچنین بردار R با عناصر توزیع یکنواخت:

$$R = [r_1, r_2, r_3, \dots, r_{N_{imp}}] \quad r_1, r_2, r_3, \dots, r_{N_{imp}} \sim U(0,1) \quad (33-2)$$

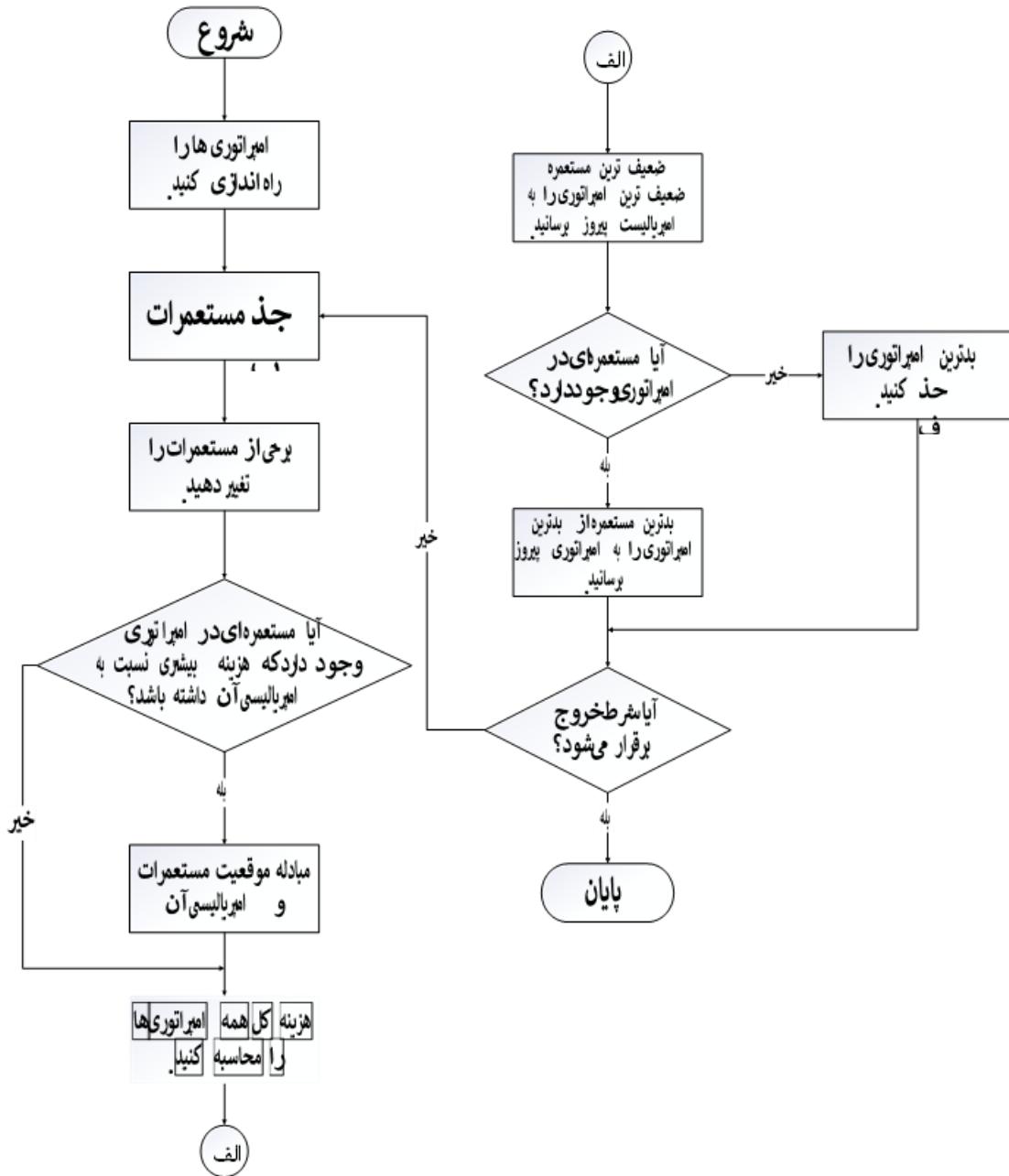
و در نهایت بردار D برابر است با:

$$D = P - R = [p_{p1} - r_1, p_{p2} - r_2, p_{p3} - r_3, \dots, p_{p_{N_{imp}}} - r_{N_{imp}}] \quad (34-2)$$

عناصر D مستعمرات ذکر شده را به امپراطوری تحویل می‌دهند که شاخص مربوطه در D حداکثر است.

۲-۱-۴-۷ همگرایی

در پایان، یک امپراطوری باقی ماند و همه مستعمرات تحت کنترل این امپراطوری خواهد بود. بنابراین، تمام مستعمرات همان هزینه‌هایی را خواهند داشت که امپراطوری منحصر به فرد خواهد داشت. این بدان معنی است که هیچ تفاوتی بین مستعمرات و امپراطوری‌ها باقی نمی‌ماند. به معنای دیگر کشورهای مستعمره را دیگر نیاز نداریم و جواب نهایی ما امپریالیست تنها امپراطوری باقی‌مانده می‌شود. در این حالت الگوریتم پایان یافته و به مقداری مشخص همگرا شده است.



شکل ۲ - ۶: فلوچارت الگوریتم امپریالیست رقابتی طبق الگوریتم^(۳)

۱-۴-۸- مدل‌سازی با ورودی تصادفی

برای بررسی قابلیت این الگوریتم برای بهینه‌سازی تابع تست شانه تخم مرغی ابتدا باید الگوریتم امپریالیست رقابتی را در مطلب، مدل‌سازی کنیم. در ادامه توضیح مختصری از کد مدل‌سازی آورده شده است. پارامترهای تعیین شده در بخش اول کد در جدول ۲ - ۴ آورده شده است. در این برنامه، تعداد امپریالیست‌های اولیه، $nEmp$ ، فشار انتخاب، α ، ضریب جذب، β ، نرخ انقلاب، μ ، احتمال انقلاب، $pRevolution$ ، و ζ میانگین ضریب هزینه مستعمرات است.

جدول ۲ - ۴: جدول پارامترهای الگوریتم بهینه امپریالیست رقابتی

پارامتر	مقدار
<i>Range</i>	-۵۱۲_۵۱۲
<i>population</i>	۶۰
<i>nEmp</i>	۱۰
<i>alpha</i>	۱
<i>beta</i>	۲
<i>pRevolution</i>	۰.۵
<i>mu</i>	۰.۵
<i>zeta</i>	۰.۱

در قسمت بعدی مانند تمامی الگوریتم‌های دیگر در این گزارش، یک جمعیت تصادفی اولیه در فضای جستجو ایجاد می‌کنیم و هزینه هر عضو این جمعیت را محاسبه می‌کنیم. سپس تمامی کشورها را بر حسب هزینه مرتب کرده و به تعداد $nEmp$ از میان کشورها امپریالیست انتخاب می‌کنیم که بهترین هزینه را در میان تمام کشورها دارند. بعد از آن طبق فرمول (۲۵-۲)، p را حساب کرده و با استفاده از انتخاب با چرخ رولت، سایر کشورهای باقی‌مانده یا کلونی‌ها را به هر امپریالیست نسبت می‌دهیم و تعداد کلونی‌های هر امپریالیست را می‌شماریم. به طور کلی در این قسمت امپراتوری اولیه ایجاد می‌شود.

برنامه ۲ - ۱۶: ایجاد جمعیت اولیه برای الگوریتم امپریالیست رقابتی

```
function [emp,x11,y11]=CreateInitialEmpires()
    global ProblemSettings;
    global ICASettings;
    CostFunction=ProblemSettings.CostFunction;
    nVar=ProblemSettings.nVar;
    VarSize=ProblemSettings.VarSize;
    VarMin=ProblemSettings.VarMin;
    VarMax=ProblemSettings.VarMax;

    nPop=ICASettings.nPop;
    nEmp=ICASettings.nEmp;
    nCol=nPop-nEmp;
    alpha=ICASettings.alpha;

    empty_country.Position=[];
    empty_country.Cost=[];
```

```

country=repmat(empty_country,nPop,1);

for i=1:nPop
    country(i).Position=unifrnd(VarMin,VarMax,VarSize);

    country(i).Cost=CostFunction(country(i).Position);
    x11(i)=country(i).Position(1);
    y11(i)=country(i).Position(2);
end

costs=[country.Cost];
[~, SortOrder]=sort(costs);
country=country(SortOrder);

imp=country(1:nEmp);

col=country(nEmp+1:end);

empty_empire.Imp=[];
empty_empire.Col=repmat(empty_country,0,1);
empty_empire.nCol=0;
empty_empire.TotalCost=[];

emp=repmat(empty_empire,nEmp,1);

% Assign Imperialists
for k=1:nEmp
    emp(k).Imp=imp(k);
end

% Assign Colonies
P=exp(-alpha*[imp.Cost]/max([imp.Cost]));
P=P/sum(P);
for j=1:nCol
    k=RouletteWheelSelection(P);
    emp(k).Col=[emp(k).Col
                col(j)];

    emp(k).nCol=emp(k).nCol+1;
end
emp=UpdateTotalCost(emp);
end

```

برنامه ۲ - ۱۷: انتخاب با روش چرخ رولت

```

function i=RouletteWheelSelection(P)
r=rand;
C=cumsum(P);
i=find(r<=C,1,'first');
end
end

```

سپس بهینه‌سازی را با استفاده از چرخهای با شرط خروج، اختلاف 0.01 با کمینه جهانی، شروع می‌کنیم. در قسمت بعد، بخش جذب آغاز می‌شود. مطابق معادله‌ی [\(۲۷-۲\)](#) موقعیت کلونی‌ها را تغییر می‌دهیم و بعد از بررسی حضور جواب‌ها در مرزهای فضای جستجو، هزینه کلونی را محاسبه می‌کنیم.

برنامه ۲ - ۱۸: جذب

```

nEmp=numel(emp);
for k=1:nEmp
for i=1:emp(k).nCol

    emp(k).Col(i).Position = emp(k).Col(i).Position ...
+ beta*rand(VarSize).*(emp(k).Imp.Position-emp(k).Col(i).Position);

    emp(k).Col(i).Position = max(emp(k).Col(i).Position,VarMin);
    emp(k).Col(i).Position = min(emp(k).Col(i).Position,VarMax);

    emp(k).Col(i).Cost = CostFunction(emp(k).Col(i).Position);
end
end

```

حال به بررسی تابع مربوط به انقلاب می‌پردازیم. ابتدا طبق معادله (۲۸-۴) تعداد کلونی‌های هر امپریالیست که انقلاب می‌کنند را محاسبه و سپس موقعیت جدیدی برای این کشورها به صورت رندوم تولید می‌کنیم. سپس بررسی می‌کنیم اگر موقعیت جدید کلونی بهتر بود، آن را با موقعیت قبلی جایگزین کرده و در آخر به بررسی قرار گرفتن کلونی‌های جدید در مرزهای تعیین شده می‌پردازیم.

برنامه ۲ - ۱۹: انقلاب

```

sigma=0.1*(VarMax-VarMin);
nEmp=numel(emp);

for i=1:nEmp
    nmu(i)=ceil(mu*emp(i).nCol);
    rands(i).rands=randperm(emp(i).nCol);
end
for k=1:nEmp
    for i=1:nmu(k)
        NewPos = emp(k).Col(rands(k).rands(i)).Position + sigma*randn(VarSize);
        if rand<=pRevolution
            emp(k).Col(rands(k).rands(i)).Position(:) = NewPos(:);
            for j=1:emp(k).nCol
                emp(k).Col(j).Position = max(emp(k).Col(j).Position,VarMin);
                emp(k).Col(j).Position = min(emp(k).Col(j).Position,VarMax);
                emp(k).Col(j).Cost = CostFunction(emp(k).Col(j).Position);
            end
        end
    end
end

```

در قسمت بعد برای رقابت درون امپراطوری، هزینه کلونی‌های هر امپریالیست را با هزینه امپریالیست مقایسه کرده و اگر بهتر بود، جای آنها را عوض می‌کنیم.

برنامه ۲ - ۲۰: رقابت درون امپراطوری

```

for k=1:nEmp
    for i=1:emp(k).nCol
        if emp(k).Col(i).Cost<emp(k).Imp.Cost
            imp=emp(k).Imp;
            col=emp(k).Col(i);

            emp(k).Imp=col;
            emp(k).Col(i)=imp;

        end
    end
end

```

سپس هزینه کل هر امپراطوری را حساب می‌کنیم و اگر امپریالیستی کلونی نداشت، هزینه کل آن امپراطوری برابر هزینه امپریالیست می‌شود.

برنامه ۲ - ۲۱: به روزرسانی هزینه کل

```

nEmp=numel(emp);

for k=1:nEmp
    if emp(k).nCol>0
        emp(k).TotalCost=emp(k).Imp.Cost+zeta*mean([emp(k).Col.Cost]);
    else
        emp(k).TotalCost=emp(k).Imp.Cost;
    end
end

```

و در آخر بهینه‌سازی، با استفاده از هزینه کل، رقابت بین امپریالیست‌ها آغاز می‌شود. به اینصورت که ضعیفترین امپریالیست با حداقل هزینه را پیدا می‌کنیم و طبق ادامه بخش رقابت امپریالیستی، p را محاسبه می‌کنیم. در ادامه اگر ضعیفترین امپریالیست کلونی مربوطه داشت، ضعیفترین کلونی آن را محاسبه می‌کنیم و با استفاده از چرخه رولت، یک امپریالیست را به عنوان امپریالیستی که کلونی را تصرف می‌کند انتخاب می‌کنیم. اینگونه ضعیفترین امپراطوری را حذف کرده و کشورهای مربوطه را به سایر امپراطوری‌ها منتقل می‌کنیم. هر چند اگر امپریالیست دیگر کلونی‌ای نداشت، خود امپریالیست را به امپریالیست برنده به عنوان کلونی دیگر اضافه می‌کنیم.

برنامه ۲ - ۲۲: رقابت بین امپراطوری

```

TotalCost=[emp.TotalCost];
[~, WeakestEmpIndex]=max(TotalCost);
WeakestEmp=emp(WeakestEmpIndex);

P=exp(-alpha*TotalCost/max(TotalCost));
P(WeakestEmpIndex)=0;
P=P/sum(P);
if any(isnan(P))
    P(isnan(P))=0;
    if all(P==0)
        P(:)=1;
    end
    P=P/sum(P);
end
if WeakestEmp.nCol>0

```

```
[~, WeakestColIndex]=max([WeakestEmp.Col.Cost]);
WeakestCol=WeakestEmp.Col(WeakestColIndex);
WinnerEmpIndex=RouletteWheelSelection(P);
WinnerEmp=emp(WinnerEmpIndex);
WinnerEmp.Col(end+1)=WeakestCol;
WinnerEmp.nCol=WinnerEmp.nCol+1;
emp(WinnerEmpIndex)=WinnerEmp;
WeakestEmp.Col(WeakestColIndex)=[];
WeakestEmp.nCol=WeakestEmp.nCol-1;
emp(WeakestEmpIndex)=WeakestEmp;
end
if WeakestEmp.nCol==0

    WinnerEmpIndex2=RouletteWheelSelection(P);
    WinnerEmp2=emp(WinnerEmpIndex2);

    WinnerEmp2.Col(end+1)=WeakestEmp.Imp;
    WinnerEmp2.nCol=WinnerEmp2.nCol+1;
    emp(WinnerEmpIndex2)=WinnerEmp2;

    emp(WeakestEmpIndex)=[];
end
```

در پایان کد نیز بررسی می‌شود تا تمام کشورها بین مرزهای تعیین شده قرار داشته باشند و بهترین کشور انتخاب شده را با شرط خروج تطبیق می‌دهیم. اگر شرط برقرار شد، جواب بهینه طبق پیوست آ_۱ نمایش داده می‌شود. در غیر این صورت چرخه را دوباره تکرار می‌کنیم تا شرط خروج برقرار شود.

۲-۱-۴-۹- مدل‌سازی الگوریتم با ورودی از قبل تعیین شده

برای بررسی بیشتر قدرت الگوریتم، داده‌های ورودی برای بهینه‌سازی را به صورت غیر تصادفی در ۸ حالت مختلف تقسیم نمودیم. موقعیت اولیه نقاط به ترتیب به دسته‌های مقابله تقسیم شدن: بالا_راست، قطر_فرعی، بالا_چپ، راست، قطر_اصلی، پایین_راست، پایین_راست_چپ به این معنی که از آنجایی که بازه‌ی فضای مورد بررسی در مسئله [512, 512] می‌باشد، پس نقاطی که در دسته‌ی بالا راست قرار می‌گیرند در بازه‌ی [0, 512] چه در جهت x و چه در جهت y قرار دارند. سپس بدون تغییر پارامترهای مسئله، کد مربوط به این نوع از گرفتن داده را به شکل زیر می‌نویسیم.

برنامه ۲ - ۲۳: وارد کردن ورودی‌های برنامه

```
InitialVariables = {'top_right', 'Secondary_Axis','top_left',
'right','Main_Axis', 'down_right' , 'down_left', 'left' };
%it just work on the first two elemnt of InitialVariables
for f = 1:numel(InitialVariables)
    % Load data from .mat file
    data = load(InitialVariables{f}); % Load data from .mat file
```

برنامه ۲ - ۲۴: نسبت دادن ورودی‌ها به کشورها و امپریالیست‌ها

```
function [emp,x11,y11] = CreateInitialEmpire_WithKnownInitialCountry(data)
% Global Variables
```

```

global ProblemSettings;
global ICASettings;
CostFunction=ProblemSettings.CostFunction;
nVar=ProblemSettings.nVar;
VarSize=ProblemSettings.VarSize;
VarMin=ProblemSettings.VarMin;
VarMax=ProblemSettings.VarMax;

nPop=ICASettings.nPop;
nEmp=ICASettings.nEmp;
nCol=nPop-nEmp;
alpha=ICASettings.alpha;

%% Initialize Countries
% Initialize empty country structure
empty_country = struct('Position', [], 'Cost', []);

% Initialize country structure
country = repmat(empty_country, nPop, 1);

for i=1:nPop

    country(i).Position = data.x(i,:);
    country(i).Cost=CostFunction(country(i).Position);
    x11(i)=country(i).Position(1);
    y11(i)=country(i).Position(2);
end

costs=[country.Cost];
[~, SortOrder]=sort(costs);
country=country(SortOrder);

imp=country(1:nEmp);

col=country(nEmp+1:end);

empty_empire.Imp=[];
empty_empire.Col=repmat(empty_country,0,1);
empty_empire.nCol=0;
empty_empire.TotalCost=[];

emp=repmat(empty_empire,nEmp,1);

% Assign Imperialists
for k=1:nEmp
    emp(k).Imp=imp(k);
end

% Assign Colonies
P=exp(-alpha*[imp.Cost]/max([imp.Cost]));
P=P/sum(P);
for j=1:nCol

    k=RouletteWheelSelection(P);

```

```

emp(k).Col=[emp(k).Col
            col(j)];
emp(k).nCol=emp(k).nCol+1;
end

emp=UpdateTotalCost(emp);

end

```

واضح از آنجایی که الگوریتم تغییری نکرده است و فقط در ورودی‌های الگوریتم تغییر ایجاد شده پس باقی قسمت‌های برنامه همانند بخش مدل‌سازی الگوریتم با ورودی تصادفی می‌باشد.

۲ - ۵ بهینه‌سازی ماهی الکتریکی

تمامی این الگوریتم‌های فرآیند کاری مبتنی بر جمعیت، از یک چارچوب مشخص و تقریباً ثابت استفاده می‌کنند. با این حال استفاده از عملگرهای جستجو به مدل بستگی دارد و این قضیه الگوریتم‌ها را از هم تمیز می‌دهد. ابتدا، مفروضاتی که رفتار ماهی الکتریکی را مطابق با ماهیت واقعی آنها به الگوریتم اکتشافی تبدیل می‌کند، معرفی می‌شوند. فرض اول این است که یک منبع غذایی بی نهایت در فضای جستجو وجود دارد، جایی که یک منبع غذایی به عنوان بهترین منبع شناخته می‌شود. ماهی‌های الکتریکی، مربوط به افراد موجود در الگوریتم، در جایی در فضا قرار دارند و اطلاعات مکان خود را حمل می‌کنند. کیفیت هر فرد با فاصله آن تا محل بهترین منبع تعیین می‌شود و از این رو مشکل موجود به مشکل یافتن بهترین منبع غذایی با کیفیت تبدیل می‌شود. فرض دیگری که از تکامل طبیعی الهام گرفته شده است، این است که ماهی‌هایی که دارای منابع با کیفیت بالا در مدت زمان طولانی هستند، سریع‌تر رشد می‌کنند و بنابراین سیگنال‌های الکتریکی با دامنه بالاتری نسبت به بقیه تولید می‌کنند.

رفتارهای هوشمند ماهی الکتریکی به صورت زیر مدل‌سازی می‌شود:

۱. **مکان‌یابی فعال**:^۱ ماهی‌های الکتریکی هر چه به بهترین منبع غذایی نزدیک‌تر شوند، سیگنال‌های الکتریکی پیوسته‌تری را با استفاده از اندام‌های الکتریکی خود تولید می‌کنند. مکان‌یابی الکتریکی فعال دارای برد بسیار محدودی است و بنابراین برای اطمینان از جستجوی محلی از افراد با مقادیر تناسب بهتر استفاده شده است، به طوری که این افراد می‌توانند در مجاورت خود جستجو کنند.

^۱ Active electrolocation

۲. مکان‌یابی الکتریکی غیر فعال^۱: ماهی‌های دیگر میدان الکتریکی ایجاد نمی‌کنند، اما به سیگنال‌های الکتریکی متکی هستند که از اندام الکتریکی همنوع خود یا از موجودات زنده منتشر می‌شود. مکان‌یابی الکتریکی غیرفعال دامنه وسیع‌تری نسبت به مکان‌یابی الکتریکی فعال دارد لذا در الگوریتم، از آن برای ایجاد تعادل در مکان‌یابی الکتریکی فعال و اطمینان از جستجوی جهانی با قادر ساختن افراد، به ویژه آنها‌ی که شایستگی ضعیفی دارند برای کاوش در فضای دور استفاده شده است.

۳. فرکانس: در طبیعت، فرکانس تولید میدان الکتریکی به نزدیکی منبع بستگی دارد. ماهی‌هایی که نزدیک به بهترین منبع هستند، میدان الکتریکی بیشتری نسبت به سایرین تولید می‌کنند. فرکانس در EFO به عنوان کلیدی برای تعیین نقش هر فرد در زمان t استفاده شده است. زیرا نشانگر این است که توسط ماهی‌های برقی استفاده می‌شود تا بفهمند کدام افراد در مجاورت منبع غذایی بهتری هستند. همانند طبیعت، افراد با فرکانس بالاتر از مکان‌یابی الکتریکی فعال و دیگران از مکان‌یابی الکتریکی غیرفعال استفاده می‌کنند.

۴. دامنه: دامنه میدان الکتریکی به رشد ماهی بستگی دارد و محدوده موثر محرک‌های الکتریکی را تعیین می‌کند. در الگوریتم EFO، با توجه به توانایی آن در تعیین تسلط میدان‌های الکتریکی، دامنه برای تعیین محدوده موثر در جستجوی محلی و احتمال حس شدن افراد در جستجوی جهانی استفاده شده است.

۱-۵-۱-۲ الگوریتم

در ابتدا جمعیت ماهی‌های الکتریکی (N) (در توضیحات افراد نامیده می‌شوند) بطور تصادفی در فضای جستجوی تعیین شده تولید می‌شوند.

$$x_{ij} = x_{\min j} + \phi(x_{\max j} - x_{\min j}) \quad (33-2)$$

x موقعیت فرد (راه حل) i را در فضای جستجوی مسئله نشان می‌دهد. لازم به ذکر است که فرد i از میان جمعیت با اندازه $N, i = 1, 2, \dots, N$ انتخاب می‌شود. x و ϕ مرزهای تعیین شده برای جستجو در آن می‌باشد و $[0, 1] \in \phi$ نیز یک مقدار تصادفی است که از یک توزیع یکنواخت استفاده می‌کند. افراد در جمعیت از طریق قابلیت مکان‌یابی فعال یا غیرفعال خود در فضای جستجو، درست پس از مرحله اول مقداردهی اولیه جمعیت، حرکت می‌کنند. فرکانس نقش کلیدی در الگوریتم برای ایجاد تعادل بین اکتشاف و بهره‌برداری ایفا می‌کند و برای تعیین اینکه آیا یک فرد مکان‌یابی الکتریکی فعال یا غیرفعال را انجام می‌دهد استفاده می‌شود. افراد بهتر (افراد حالت فعال) را که به احتمال زیاد در مجاورت مناطق امیدوار کننده هستند، وادر می‌کند تا از همسایگی خود سوء استفاده کنند، و افراد دیگر (افراد حالت غیرفعال) را به کاوش در فضای جستجو هدایت می‌کند تا منطقه جدیدی را کشف کنند، که برای عملکردهای

^۱ Passive electrolocation

چند وجهی ضروری است. در الگوریتم، مانند طبیعت افراد با فرکانس بالاتر از مکانیابی الکتریکی فعال استفاده می‌کنند و دیگران از مکانیابی الکتریکی غیر فعال استفاده می‌کنند. مقدار فرکانس یک فرد از حداقل مقدار f_{\min} تا حداقل مقدار f_{\max} متغیر است. از آنجایی که مقدار فرکانس یک ماهی بر قی در زمان t به شدت با نزدیکی آن به منبع غذایی مرتبط است، مقدار فرکانس یک فرد (f_i^t) از مقدار تناسب آن مشتق می‌شود:

$$f_i^t = f_{\min} + \left(\frac{fit_{i_{worst}}^t - fit_i^t}{fit_{i_{worst}}^t - fit_{i_{best}}^t} \right) (f_{\max} - f_{\min}) \quad (34-2)$$

fit^t ارزش نامین فرد در تکرار t است. در این مطالعه، از آنجایی که از مقدار فرکانس برای محاسبه احتمال استفاده می‌شود، f_{\min} و f_{\max} به ترتیب روی ۰ و ۱ تنظیم می‌شوند. به غیر از فرکانس، ماهی‌های الکتریکی دارای اطلاعات دامنه نیز هستند. این محدوده فعال ماهی را در حین مکانیابی الکتریکی فعال و احتمال درک شدن توسط سایر ماهی‌های مکانیابی الکتریکی غیرفعال را تعیین می‌کند، زیرا قدرت میدان الکتریکی با مکعب معکوس فاصله کاهش می‌یابد. دامنه‌یک فرد به وزن دامنه‌های قبلی فرد بستگی دارد (α در معادله ۳۵-۲)، و بنابراین ممکن است تغییر شدیدی نکند. مقدار دامنه نامین فرد (A_i) به صورت زیر محاسبه می‌شود:

$$A_i^t = \alpha A_i^{t-1} + (1-\alpha) f_i^t \quad (35-2)$$

که در آن α مقدار ثابتی بین ۰ و ۱ است که بزرگی مقدار دامنه قبلی را تعیین می‌کند. در الگوریتم، مقدار دامنه اولیه نامین فرد با توجه به مقدار فرکانس اولیه خودش f تنظیم می‌شود. مقادیر پارامتر فرکانس و دامنه‌یک ماهی (یا فرد) براساس نزدیکی ماهی به بهترین منبع طعمه به روز می‌شود. در هر تکرار الگوریتم، جمعیت بر اساس مقدار فرکانس هر فرد به دو گروه تقسیم می‌شود: افرادی که مکانیابی الکتریکی فعال (N_A) را انجام می‌دهند و افرادی که مکانیابی الکتریکی غیرفعال (N_p) را انجام می‌دهند (یعنی $N_A \cup N_p = N$). از آنجایی که فرکانس یک فرد با یک مقدار تصادفی توزیع شده یکواخت مقایسه می‌شود، هر چه مقدار فرکانس یک فرد بیشتر باشد، احتمال بیشتری برای انجام مکانیابی الکتریکی فعال خواهد داشت. سپس جستجو توسط افراد در N_A و N_p به صورت موازی انجام می‌شود.

۲-۱-۵-۲ مکانیابی الکتریکی فعال

محدوده موثر بیولوژیکی مکانیابی الکتریکی فعال تقریباً به نیمی از اندازه جمعیت ماهی‌ها محدود شده است و ماهی قادر به درک شکار خارج از این محدوده نیست. این بدان معنی است که این قابلیت به ماهی‌ها اجازه می‌دهد تا هر منبع

غذایی را در مجاورت خود پیدا کنند. قابلیت جستجو یا بهره‌برداری محلی بر اساس این ویژگی‌های مکانیابی الکترونیکی فعال است. در بهینه‌سازی همه افراد در حالت فعال (انجام مکانیابی الکترونیکی فعال) با اصلاح صفات خود در فضای جستجو حرکت می‌کنند. با این حال، تنها یک پارامتر که به طور تصادفی انتخاب شده مجاز به اصلاح است، تا از منطقه امیدوارکننده خیلی دور نشوند. حرکت فرد i ام ممکن است بسته به وجود همسایگان در محدوده فعال آن متفاوت باشد. اگر همسایه‌ای وجود نداشته باشد (شکل ۷-۲ بخش A)، یک پیاده روی تصادفی در سراسر محدوده خود انجام می‌دهد، در غیر اینصورت یک همسایه را به طور تصادفی انتخاب می‌کند و بسته به این همسایه مکان آن را تغییر می‌دهد (شکل ۷-۲ بخش B). جستجوی اختیاری با رویکرد "اول کاوش، سپس بهره‌برداری" در الگوریتم کمک می‌کند. از آنجایی که افراد در ابتدا از یکدیگر دور هستند، احتمال کمتری دارد که به محدوده فعال یکدیگر تعلق داشته باشند. بنابراین، بسیار محتمل است که افراد حالت فعال در ابتدا به صورت تصادفی در همسایگی خود جستجو کنند و سپس با ادامه تکرار شروع به بهره‌برداری از یکی از نزدیک‌ترین همسایگان کنند. محدوده فعال i امین فرد (r_i) مانند طبیعت با مقدار دامنه (A_i) خود تعیین می‌شود. محاسبه محدوده فعال طبق معادله (۳۶-۲) صورت می‌گیرد. برای یافتن افراد همسایه ($S | S \subset N$) در محدوده حسگر/فعال، باید فاصله بین آن فرد و بقیه جمعیت را اندازه‌گیری کرد. فاصله بین افراد i و k با از روش دکارتی تعیین می‌شود:

$$d_{ik} = \|x_i - x_k\| = \sqrt{\sum_{j=1}^d (x_{ij} - x_{kj})^2} \quad (37-2)$$

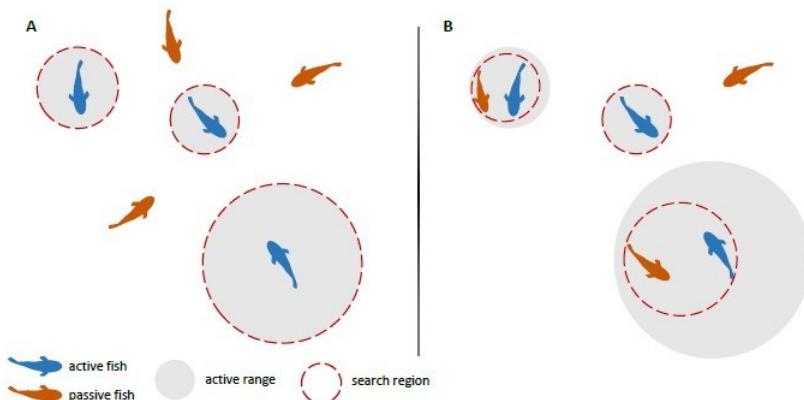
در موردی که حداقل یک همسایه در ناحیه سنجش فعال وجود داشته باشد، از معادله (۳۸-۲) و در غیر اینصورت (یعنی $S = \phi$) از معادله (۳۹-۲) استفاده می‌کنیم.

$$x_{ij}^{cond} = x_{ij} + \phi(x_{kj} - x_{ij}) \quad (38-2)$$

که در آن K نشان‌دهنده فردی است که از مجموعه همسایه فرد i ام (یعنی $d_{ik} \leq r_i$ و $k \in S$) به طور تصادفی انتخاب شده است.

$$x_{ij}^{cond} = x_{ij} + \phi r_i \quad (39-2)$$

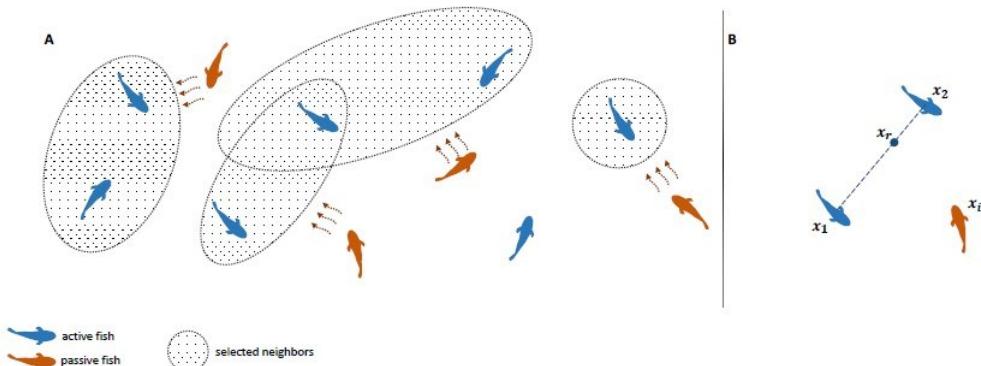
جایی که $\phi \in [-1, 1]$ در معادله (۳۸-۲) و (۳۹-۲) یک عدد تصادفی است که از یک توزیع یکنواخت تولید می‌شود و x_{ij}^{cond} نشان‌دهنده مکان منتخب فرد i است.



شکل ۲ - ۷: نمایش جستجوی محلی EFO. فردی که در حالت فعال است بسته به وجود همسایه در نزدیکی خود

۲-۱-۵-۳ مکانیابی الکتریکی غیر فعال

برخلاف مکانیابی الکتریکی فعال، فاصله حسی به فرد بستگی ندارد و مانند طبیعت از محدوده مکانیابی الکتریکی فعال فراتر می‌رود. به همین دلیل است که قابلیت مکانیابی الکتریکی غیرفعال الزامات مکانیزم جستجو یا اکتشاف جهانی الگوریتم EFO پیشنهادی را برآورده می‌کند. همانطور که قبل ذکر شد، احتمال درک یک سیگنال مستقیماً با مقدار دامنه خود و فاصله تا فرد مورد نظر تناسب دارد. افراد در حالت غیرفعال، افراد دیگری را در حالت فعال انتخاب می‌کنند که بسته به احتمال، سیگنال‌های الکتریکی را منتشر می‌کنند و سپس مکان آن را تغییر می‌دهند.



شکل ۲ - ۸: نمایش جستجوی جهانی EFO

احتمال درک k امین فرد در حالت فعال ($k \in N_A$) توسط فرد i در حالت غیرفعال ($i \in N_P$) با استفاده از معادله (۴۰-۲) محاسبه می‌شود:

$$P_k = \frac{A_k/d_{ik}}{\sum_{j \in N_A} A_j/d_{ij}} \quad (40-2)$$

در اینجا ذکر این نکته مهم است که انتخاب همسایه احتمالی، افراد حالت غیرفعال را مجبور می‌کند قبل از بهره‌برداری کاوش را انجام دهنند. بنابراین، افراد حالت غیرفعال در تکرارهای اولیه افراد همسایه را انتخاب می‌کنند، زیرا فاصله عامل غالب است که باعث انتخاب افراد فقیر می‌شود و به کشف مناطق جدید فضای جستجو کمک می‌کند. با ادامه تکرار، دامنه به عامل غالب تبدیل می‌شود (زیرا فاصله بین افراد EFO به صفر نزدیک می‌شود)، که باعث انتخاب و جستجوی محلی بهترین افراد می‌شود. با استفاده از استراتژی‌های مختلف، مانند انتخاب چرخ رولت که در اینجا به کار می‌رود، k فرد از N_A بر اساس معادله (40-2) انتخاب می‌شوند و سپس یک مکان مرجع (x_{rj}) براساس معادله (41-2) تعیین می‌شود. مکان جدید سپس از معادله (42-2) ایجاد می‌شود.

$$x_{rj}^{new} = \frac{\sum_{k=1}^K A_k x_{kj}}{\sum_{k=1}^K A_k} \quad (41-2)$$

$$x_{ij}^{new} = x_{ij} + \phi(x_{rj} - x_{ij}) \quad (42-2)$$

برخلاف جستجو در مکان‌یابی فعال، بیش از یک پارامتر را می‌توان تغییر داد، به طوری که افراد فضای جستجو را بسیار سریع‌تر کشف می‌کنند. با این حال، هرچند به ندرت، ممکن است موردی وجود داشته باشد که در آن فردی با فرکانس بالاتر، مکان‌یابی غیرفعال را انجام دهد. در چنین حالتی، آن فرد اطلاعات مکان خود را به طور کامل از دست می‌دهد، که به دلیل منطقه امیدوار کننده‌ای که در آن قرار دارد، انتظار نمی‌رود. برای جلوگیری از این امر، EFO معادله (43-2) را در نظر می‌گیرد، برای تعیین اینکه کدام پارامترها تغییر خواهند کرد. در شرایط پذیرش، احتمال این که چنین فردی کل صفت خود را تغییر دهد به میزان قابل توجهی کاهش می‌یابد.

$$x_{ij}^{cond} = \begin{cases} x_{ij}^{new} & rand_j(0,1) > f_i \\ x_{ij} & \text{else} \end{cases} \quad (43-2)$$

که در آن $rand_j(0,1)$ یک عدد تصادفی یکنواخت تولید شده برای پارامتر j است. مرحله نهایی مکان‌یابی الکترونیکی غیرفعال، اصلاح یک پارامتر از فرد i برای افزایش احتمال تغییرات یک صفت بر حسب معادله (44-2) است.

$$x_{ij}^{cond} = x_{\min j} + \phi(x_{\max j} - x_{\min j}) \quad rand(0,1) \leq rand(0,1) \quad (44-2)$$

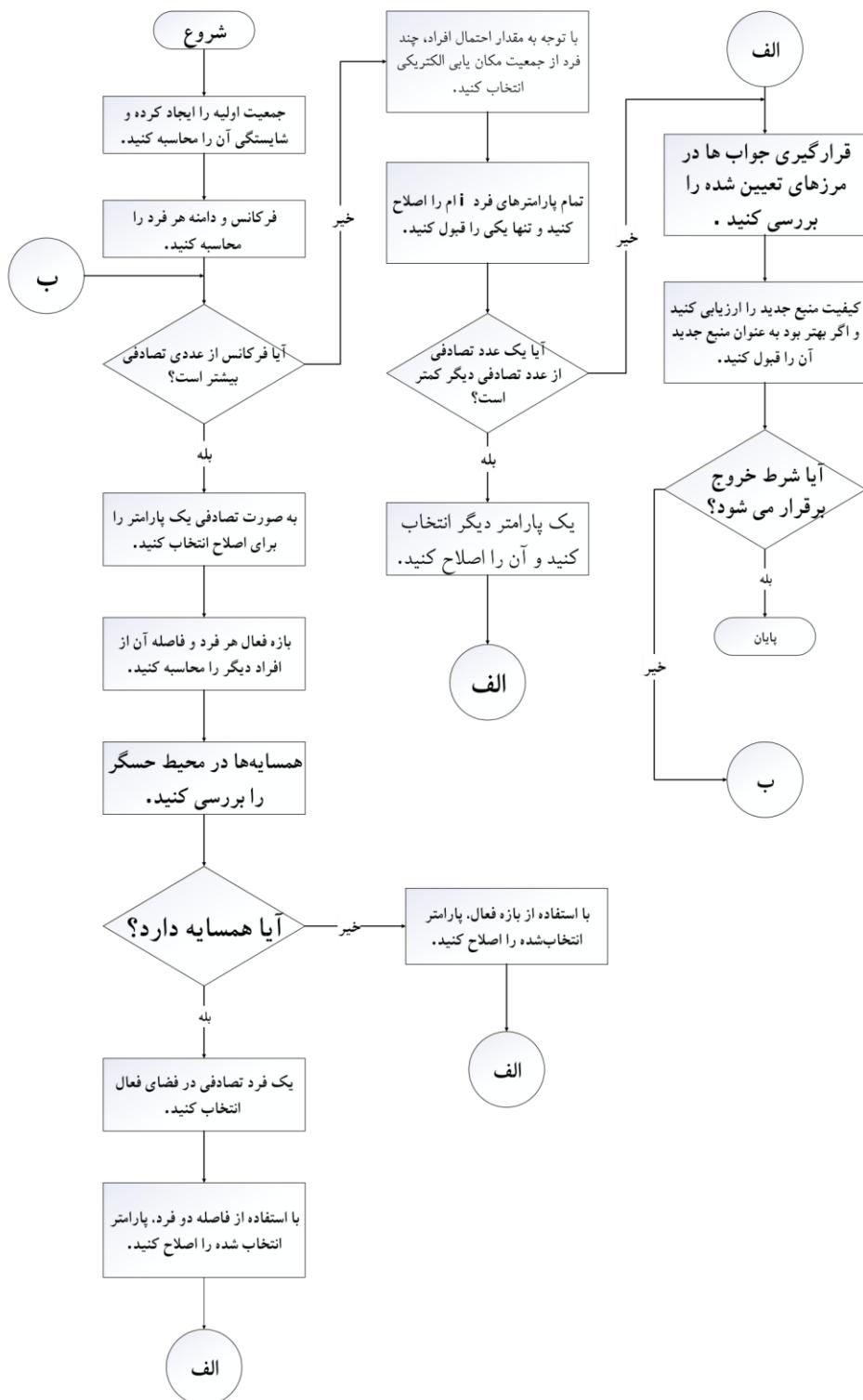
که در آن $rand(0,1)$ یک عدد تصادفی یکنواخت تولید شده است. اگر مقدار z امین پارامتر از i امین فرد از مرزهای

فضای جستجو تجاوز کند، به مرز فضایی که از آن فراتر می‌رود منتقل می‌شود:

$$x_{ij}^{cond} = \begin{cases} x_{\min j} & x_{ij}^{cond} < x_{\min j} \\ x_{ij}^{cond} & x_{\min j} \leq x_{ij}^{cond} \leq x_{\max j} \\ x_{\max j} & x_{ij}^{cond} > x_{\max j} \end{cases} \quad (45-2)$$

۲-۱-۵-۴ مدل‌سازی با ورودی تصادفی

در بخش مدل‌سازی در قسمت اول، پارامترهای مورد نیاز الگوریتم را طبق جدول ۵-۲ تعیین می‌کنیم.



شکل ۲ - ۹: فلوچارت الگوریتم بهینه‌سازی ماهی الکترونیکی طبق پیوست(ج-۴)

جدول ۲ - ۵: جدول پارامترهای الگوریتم بهینه‌سازی ماهی الکتریکی

پارامتر	مقدار
Range	-۵۱۲_۵۱۲
population	۶۰
x	۰.۹۹۹۹
k	۲۱
thres	e^{-5}

در بخش پارامترها، x ضریبی برای ایجاد توزان بین فرکانس کنونی و دامنه قدیمی، k همانطور که توضیح داده شده، تعداد ذرات جمعیت فعال و $thres$ یک مقدار کوچک برای محاسبه انحراف فرکانس‌ها است. در بخش بعدی یک جمعیت تصادفی اولیه در بازه تعیین شده برایتابع محک، برحسب معادله (۳۴-۲) ایجاد می‌شود. در ادامه نیز پس از تعیین موقعیت بهترین و بدترین ذره با استفاده از معادله (۳۴-۲) فرکانس و دامنه اولیه ذرات تعیین می‌شود. اما در این قسمت شرطی موجود است که اگر انحراف نتایج ذرات از مقدار $thres$ کمتر بود، فرکانس ذرات تصادفی ایجاد شود.

برنامه ۲ - ۲۵: فرکانس و دامنه اولیه

```
for p = 1: populationSize
    if std([population(:).fitness]) < thres
        population(p).frequency = rand;
    else
        population(p).frequency = (worstCost - population(p).fitness) / (worstCost - bestCost);
    end
    population(p).amplitude = population(p).frequency;
end
```

پس از تعیین شرایط اولیه، عملیات بهینه‌سازی آغاز می‌شود. چرخه‌ای ایجاد می‌شود که تا زمان برقراری شرط خروج یعنی اختلاف ۰.۰۱ نتایج با نتیجه بهینه جهانی، مرحله بهینه‌سازی تکرار شود. بعد از آن برای تمامی ذرات، موقعیت ماهی‌های همسایه را تعیین می‌کنیم و وارد فاز مکان‌یابی الکتریکی فعال می‌شویم. ابتدا هر ماهی که فرکانس آن از یک عدد تصادفی بیشتر باشد را در دسته ماهی‌های فعال قرار می‌دهیم. سپس از تابع جستجو فعال استفاده کرده و بازه جستجو فعال را برای ذرات فعال تعیین می‌کنیم. سپس انتخاب می‌کنیم که کدام بعد مسئله تغییر کند (موقعیت x یا y). آغاز مرحله مکان‌یابی الکتریکی فعال

```
function dist = getDistance(newIndividualSol, solutionSet)
    % Distance Calculator
    parameterSize = size(newIndividualSol, 2);
    popSize = size(solutionSet, 2) / parameterSize;
    solutionSet = reshape(solutionSet, parameterSize, popSize);
```

```
%sorting the solutionset for dimensions
dist = sqrt(sum((solutionSet - repmat(newIndividualSol, popSize, 1)) .^2,
2));
end
```

همانطور که در کد ۲۰-۲ مشخص است برای محاسبه فاصله ذرات از یکدیگر از معادله (۲۷-۳۷) استفاده شده است.

برنامه ۲ - ۲۷: محاسبه فاصله ذرات با ذرات همسایه

```
function dist = getDistance(newIndividualSol, solutionSet)
% Distance Calculator
parameterSize = size(newIndividualSol, 2);
popSize = size(solutionSet, 2) / parameterSize;
solutionSet = reshape(solutionSet, parameterSize, popSize)';
%sorting the solutionset for dimensions
dist = sqrt(sum((solutionSet - repmat(newIndividualSol, popSize, 1)) .^2,
2));
end
```

سپس بررسی می‌کنیم که آیا حداقل یک ماهی در بازه جستجو فعال قرار دارد یعنی فاصله بین آنها از مقدار بازه جستجو کمتر است. اگر هیچ همسایه‌ای وجود نداشته باشد، طبق معادله (۳۹-۲) و اگر حداقل یک همسایه وجود داشته باشد طبق معادله (۳۸-۲) وضعیت ذره تغییر می‌کند.

برنامه ۲ - ۲۸: تغییر موقعیت ماهی‌ها در فاز فعال

```
index = find(dist < activeRange);

if isempty(index)

newIndividual.solution(parameter) = newIndividual.solution(parameter) + (2 *
rand - 1) * activeRange;
else
selectedNeighbor = index(randi(size(index, 1)));
newIndividual.solution(parameter) = newIndividual.solution(parameter) + ...
(neighbornIndividual(selectedNeighbor).solution(parameter) - ...
newIndividual.solution(parameter)) * (2 * rand - 1);
end
newSolution = newIndividual.solution;
```

در قسمت بعدی حالتی بررسی می‌شود که فرکانس کوچکتر از عددی تصادفی بوده و ماهی در دسته غیر فعال قرار گیرد و به جای فاز فعال از تابع غیر فعال استفاده شود. پس اگر در همسایگی ماهی، ماهی فعالی نباشد تابع غیرفعال شروع می‌شود. در ابتدا اگر تعداد ماهی‌ها کمتر از مقدار از پیش تعیین شده است، مقدار از پیش تعیین شده را برابر جمعیت فعال کنونی قرار می‌دهیم. سپس مانند فاز فعال، فواصل را با تابع getDistance محاسبه می‌کنیم. بعد از آن با استفاده از معادله (۴۰-۲) احتمال درک ذرات را محاسبه می‌کنیم و k ذره را انتخاب می‌کنیم (چرخ رولت).

برنامه ۲ - ۲۹: محاسبه احتمالات و انتخاب ذرات فعال

```
p = cumsum([activePopulation(:).amplitude] ./ (dist' + e));
selectedNeighbor = zeros(1, activeIndSize);
for i = 1: activeIndSize
I = find (rand*p(end) < p);
selectedNeighbor(i) = I(1);
end
```

سپس جامعه همسایه را ایجاد کرده و مانند حالت فعال نتایج را برای هر بعد مرتب می‌کنیم. سپس طبق فرمول (۴۱-۲)، یک مکان مرجع درست کرده و بعد طبق فرمول (۴۲-۲) مکان ذرات جدید را تعیین می‌کنیم. با انتخاب اینکه کدام بعد باید تغییر کند، پارامتر مرتبط را بروزرسانی می‌کنیم.

برنامه ۲ - ۳۰: بروزرسانی موقعیت ذرات در فاز غیرفعال

```
neighbor = activePopulation(selectedNeighbor);
solutionSet = reshape([neighbor.solution], D, activeIndSize)';
xReference = sum(repmat([neighbor.amplitude]', 1, D) .* solutionSet) ./
sum(repmat([neighbor.amplitude]', 1, D));
newSolution = newIndividual.solution + (xReference - newIndividual.solution) *
(2 * rand(1, D) - 1);
I = find(rand(1, D) > newIndividual.frequency);
newIndividual.solution(I) = newSolution(I);
newSolution = newIndividual.solution;
```

سپس مانند معادله (۴۴-۲) اگر شرط برقرار شود، برای متنوع بودن جامعه موقعیت‌های تصادفی جدید ایجاد می‌کنیم.

برنامه ۲ - ۳۱: تغییر پارامتر دیگر به صورت تصادفی

```
if rand < rand % Mutates one or more parameters in a stochastic manner to keep
the population diverse
I = randi(parameterSize);
newIndividual.solution(I) = lowerBound + (upperBound - lowerBound) * rand;
%random new individual
end
```

در آخر بهینه‌سازی با استفاده ازتابع boundaryCheck کنترل می‌کنیم که جواب‌ها در بازه تعیین شده تابع محک باشند و سپس نتیجه محاسبه جواب‌های ماهی‌ها را با جواب‌های اولیه مقایسه می‌کنیم و اگر بودند ذرات را به جای ذرات قبلی قرار می‌دهیم.

برنامه ۲ - ۳۲: تغییر موقعیت ذره‌ها برای رسیدن به جواب بهینه

```
newIndividual.solution = boundaryCheck(newIndividual.solution, lowerBound,
upperBound); % Solution set of candidate individual, (x_cand)
newIndividual.fitness =
egg_holder(newIndividual.solution(1),newIndividual.solution(2)); % %
Fitness value of candidate individual
if newIndividual.fitness < population(p).fitness % Accepts if better
source found
population(p) = newIndividual;
end
```

بعد از آن دوباره مانند برنامه (۲۵-۲) فرکانس و دامنه ذرات را محاسبه و بروزرسانی می‌کنیم. در نهایت بهینه هر چرخه محاسبه شده و با شرط خروج بررسی می‌شود. در صورت ارضای شرط خروج از حلقه خارج می‌شویم و همانطور که در پیوست آ-۱ توضیح داده شده است نمودارها رسم می‌شود.

۲-۱-۵-۵ مدل‌سازی الگوریتم با ورودی از قبل تعیین شده

برای بررسی بیشتر قدرت الگوریتم، داده‌های ورودی برای بهینه‌سازی را به صورت غیر تصادفی در ۸ حالت مختلف تقسیم نمودیم. موقعیت اولیه نقاط به ترتیب به دسته‌های مقابل تقسیم شدن: بالا_راست، قطر_فرعی، بالا_چپ، راست، قطر_اصلی، پایین_راست، پایین_راست، چپ به این معنی که از آنجایی که بازه فضای مورد بررسی در مسئله [512, 512] می‌باشد، پس نقاطی که در دسته‌ی بالا راست قرار می‌گیرند در بازه‌ی [0, 512] چه در جهت x و چه در جهت y قرار دارند. سپس بدون تغییر پارامترهای مسئله، کد مربوط به این نوع از گرفتن داده را به شکل زیر می‌نویسیم.

برنامه ۲ - ۳۳: وارد کردن ورودی‌های برنامه و تخصیص به جمعیت اولیه

```
InitialVariables = {'top_right', 'top_left', 'Secondary_Axis', 'right',
'Main_Axis', 'left', 'down_right', 'down_left' };

for i = 1:numel(InitialVariables)

% Load data from .mat file
data = load(InitialVariables{i});

%% preallocating
population=struct('solution',cell(1,populationSize),'fitness',cell(1,populationSize),
'isActive',cell(1,populationSize), ...
'frequency',cell(1,populationSize),'amplitude',cell(1,populationSize));
%% initializing

for p = 1: populationSize

population(p).solution = data.x(p,:);
population(p).fitness =
egg_holder(population(p).solution(1),population(p).solution(2));
population(p).isActive = true;

end

populationinitial=population; %initial swarm

% The best individual information is stored
[bestCost, index] = min( [population(:).fitness] );
bestIndividual = population(index).solution;

% The worst individual information is also stored for frequency calculation
[worstCost, index] = max( [population(:).fitness] );
worstIndividual = population(index).solution;

globalBest = bestCost; %minimum cost
globalBestSol = bestIndividual; %the solution for minimum cost

% Frequency (f) and amplitude (A) values are also initialized
for p = 1: populationSize
```

```

if std([population(:).fitness]) < thres
    population(p).frequency = rand;
else
    population(p).frequency = (worstCost - population(p).fitness) / (worstCost - bestCost);
end
population(p).amplitude = population(p).frequency;
end

```

واضح از آنجایی که الگوریتم تغییری نکرده است و فقط در ورودی‌های الگوریتم تغییر ایجاد شده پس باقی قسمت‌های برنامه همانند بخش مدل‌سازی الگوریتم با ورودی تصادفی می‌باشد. و تمام توضیحاتی که در گذشته داده شده است بر روی این نحوه نگارش هم صدق می‌کند.

۲-۱-۲ تحلیل نتایج حاصل از استفاده از ورودی‌های تصادفی

در این بخش با استفاده از داده‌ها و نمودارهای به دست آمده، الگوریتم‌ها را برای تابع محک شانه تخم مرغی با یکدیگر مقایسه کرده و ارزیابی می‌کنیم. برای صحه‌گذاری قدرت عملکرد الگوریتم‌ها، با ورودی تصادفی با توجه به آنکه بروی مقدار ورودی‌ها نقشی نداریم، تنها با ثابت نگه داشتن تعداد تکرارها و پارامترهای موجود در الگوریتم باید قدرت الگوریتم را بررسی کنیم. لذا با اجراسازی برنامه برای ده بار و بررسی داده‌های خروجی شامل مدت زمان اجرا، تعداد تکرار برای رسیدن به مینیمم جهانی (خروج از حلقه while)، دقت موقعیت و نتیجه بهینه بدست آمده، می‌توانیم الگوریتم‌ها را با یکدیگر مقایسه کنیم. دقت داشته باشید که نتایج الگوریتم در یک زمان و با یک دستگاه مشخص استخراج شده است و از نظر سخت‌افزاری هیچ الگوریتمی در بستر بهتر یا بدتری اجرا نشده است.

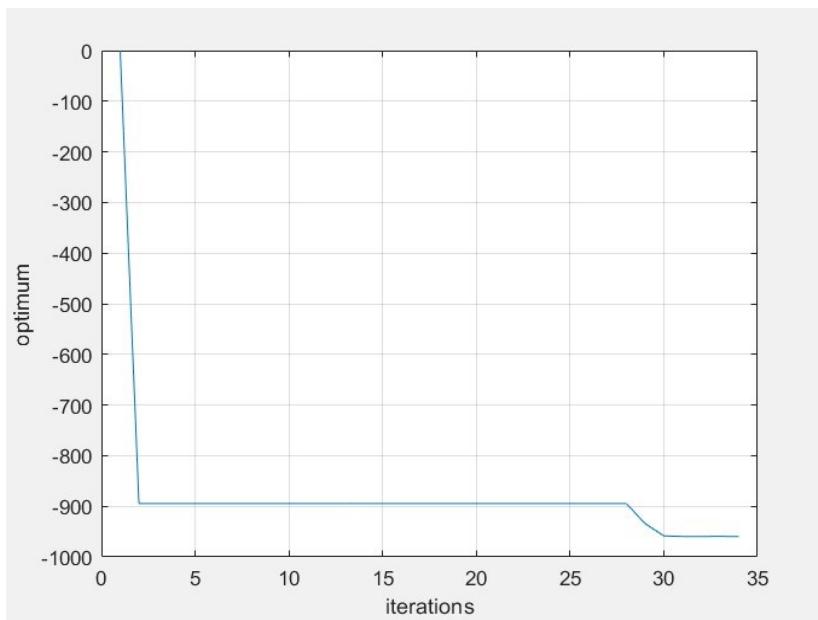
۲-۱-۳ الگوریتم بهینه‌سازی غذایابی باکتریایی

کد مربوط به الگوریتم غذایابی باکتریایی را برای ده بار اجرا می‌کنیم و تعداد تکرار مورد نیاز برای خروج از حلقه while، مدت زمان اجرای کد و موقعیت ذره‌ها را در زمان رسیدن به مینیمم سراسری استخراج می‌کنیم و در جدول (۶-۲) وارد می‌کنیم.

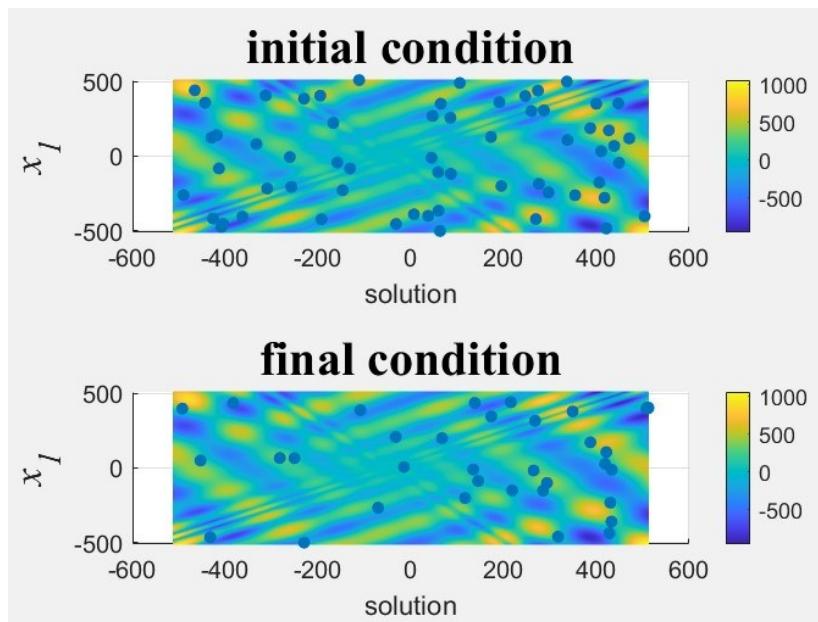
جدول ۲ - ۶: نتایج بدست آمده از الگوریتم بهینه‌سازی غذایابی باکتریایی

شماره فهرست	موقعیت بهینه جهانی	نتیجه بهینه جهانی	زمان اجرا (s)	تعداد تکرار حلقه
۱	(۵۱۲, ۴۰۴, ۳۲۲۳)	-۹۵۹/۶۳۱۱	۱/۲۸۱۹	۱۱
۲	(۵۱۲, ۴۰۴, ۲۹۶۶)	-۹۵۹/۶۳۵۹	۶/۴۱۶۷۴۱	۶۹
۳	(۵۱۲, ۴۰۴, ۲۷۶۱)	-۹۵۹/۶۳۸۴	۲/۳۴۰۸۴۸	۲۲
۴	(۵۱۲, ۴۰۴, ۱۷۸۲)	-۹۵۹/۶۳۷۴	۰/۵۲۶۸۲۲	۷
۵	(۵۱۲, ۴۰۴, ۲۴۶۵)	-۹۵۹/۶۴۰۴	۹/۵۲۶۴۹۷	۱۱۴
۶	(۵۱۲, ۴۰۴, ۲۹۵۱)	-۹۵۹/۶۳۶۱	۰/۸۵۱۳۵۲	۹
۷	(۵۱۲, ۴۰۴, ۲۱۳۴)	-۹۵۹/۶۴۰۳	۰/۲۷۵۸۱۴	۴
۸	(۵۱۲, ۴۰۴, ۱۷۸۴)	-۹۵۹/۶۳۷۴	۰/۶۷۳۱۰۵	۸
۹	(۵۱۲, ۴۰۴, ۲۹۹۴)	-۹۵۹/۶۳۵۵	۰/۲۲۵۱۳۲	۳
۱۰	(۵۱۲, ۴۰۴, ۲۴۵۹)	-۹۵۹/۶۴۰۴	۲/۱۲۲۹۷۹	۲۴

با استفاده از جواب‌های بدست آمده و میانگین‌گیری نتایج، نتیجه بهینه میانگین برابر 959.6373 ، مدت زمان اجرای میانگین برابر 2.424 ثانیه و تعداد میانگین تکرار چرخه برابر 27 است. شکل (۱۰-۲) و (۱۱-۲) نمودارهای یکی از اجراهای کد متلب مربوط به الگوریتم بهینه‌سازی غذایابی باکتریایی می‌باشد، در این اجرا نتیجه بهینه برابر 959.6404 و تعداد تکرار چرخه برابر 24 است. با اجرای مکرر برنامه مدل‌سازی الگوریتم غذایابی باکتریایی و بررسی نتایج به دست آمده، کمینه در هر تکرار دستخوش تغییرات ناگهانی نمی‌شود و اختلاف نتایج با کمینه جهانی معمولاً روند کاهشی دارد و در نهایت به جواب با دقت 0.01 می‌رسد. هر چند در برخی اجراهای مشاهده می‌شود که روند تغییرات کمینه تماماً نزولی نیست و در برخی نقاط کمینه از تکرار قبل بیشتر می‌شود. طبق مشاهدات، زمان بیشتری به سبب وجود حلقه‌های تو در تو برای انجام بهینه‌سازی لازم است، ولی تعداد تکرار به نسبت کمتری داریم.



شکل ۲ - ۱۰: نمودار تغییرات جواب بهینه براساس تعداد تکرار حلقه



شکل ۲ - ۱۱: جایگیری باکتری‌ها در موقعیت اولیه و نهایی

در اجرای مورد بررسی مشاهده می‌شود که الگوریتم توانسته است در تکرار حلقه‌ی سوم به ۹۴٪ دقت برسد که بنظر دقت بسیار خوبی است برای تنها سه مرتبه تکرار. در جدول ۲ - ۱ مشاهده می‌شود که ۳۰ باکتری از ۶۰ باکتری در جایگذاری بسیار نزدیک به هدف قرار دارند. در نتیجه یعنی ۵۰٪ باکتری‌ها توانسته‌اند به نقطه‌ی مطلوب برسند. پس درصد جایگیری صحیح حدود ۵۰ درصد می‌باشد. همانطور که در شکل ۲ - ۱۰ و شکل ۲ - ۱۱ مشاهده می‌شود، همگرایی مطلوبی در این الگوریتم صورت می‌گیرد و با توجه به جایگیری باکتری‌ها در جدول ۲ - ۱ مشاهده می‌کنیم

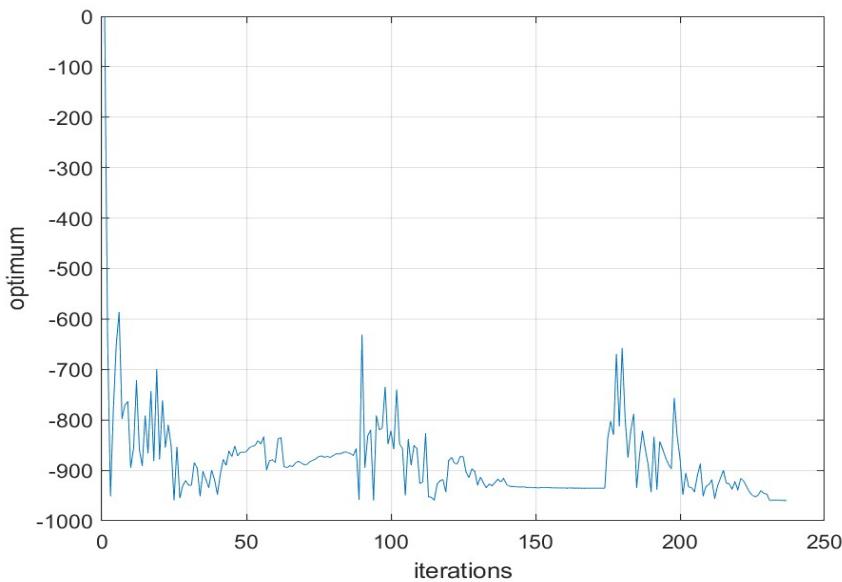
که حدود ۵۰٪ باکتری‌ها به قسمت کمینه جهانی همگرا شده و در نزدیکی آن قرار دارند و عده‌ای نیز در راه رسیدن به کمینه جهانی هستند که این امر نشان‌دهنده سرعت کمتر همگرایی از اکتشاف است. همچنین مشاهده می‌شود که عده‌ای نیز در کمینه‌های محلی گیر افتاده‌اند. لازم به ذکر است که برای بهینه‌سازی پارامترها از روش آزمون و خطا استفاده و از مقالات متعدد کمک گرفته شده‌است. با توجه به این نکته ملاحظه می‌کنیم افزایش تعداد باکتری‌ها کمک شایانی به بهینه‌سازی نمی‌کند و حتی ممکن است به دلیل وجود ذرات بیشتر برای اکتشاف، بسیار زودتر از همگرایی به نتیجه مطلوب برسیم. همچنین کمتر کردن جمعیت نیز موجب می‌شود اگر تعدادی از ذرات در کمینه‌های محلی گیر کند، تعداد ذرات کمتری برای اکتشاف و پیدا کردن کمینه جهانی موجود باشد. افزایش گام سبب می‌شود تا ذرات به سرعت از روی کمینه گذر کنند و همگرایی صورت نگیرد اما در گام‌های کوتاه‌تر نیز سبب می‌شود اکتشاف ضعیف شده و کند پیش رود. با تعداد چرخه‌های تولید‌مثل، همگرایی و با تعداد چرخه‌های حذف-پراکندگی، اکتشاف را می‌توان کنترل کرد که باید بین این پارامترها تناسب خوبی باشد که در راه اکتشاف و رسیدن به دقت مناسب ذرات بیشتری نیز به جواب کمینه جهانی همگرا شوند.

۲ - ۲ - الگوریتم بهینه‌سازی کرم شب‌تاب

کد را ده بار اجرا کرده و مقدار بهینه، تعداد تکرار چرخه، مدت زمان اجرای کد و موقعیت‌های ذره بهینه را استخراج کرده و در جدول (۲-۷) به ترتیب وارد کرده‌ایم؛ با استفاده از جواب‌های به دست آمده و میانگین‌گیری نتایج، مقدار بهینه میانگین برابر ۶۳۸۲، مدت زمان اجرای میانگین برابر ۳۸۸۷ ثانیه و تعداد میانگین تکرار چرخه برابر ۷۸۴ است.

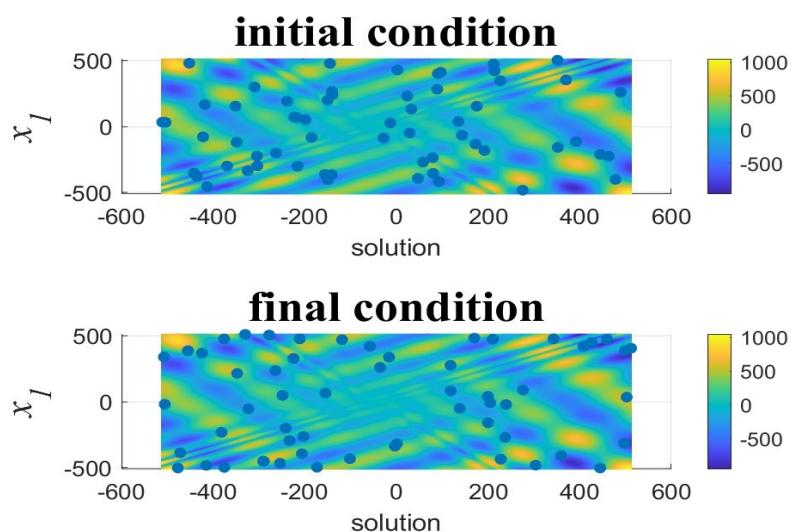
جدول ۲ - ۷: نتایج بدست آمده از الگوریتم بهینه‌سازی کرم شب‌تاب

شماره فهرست	موقعیت بهینه جهانی	نتیجه بهینه جهانی	زمان اجرای(s)	تعداد تکرار حلقه
۱	(۵۱۲, ۴۰۴, ۲۸۰۸)	-۹۵۹/۶۳۷۹	۲/۸۱۹۲۵۳	۲۳۷
۲	(۵۱۲, ۴۰۴, ۲۲۶۱)	-۹۵۹/۶۳۹۳	۰/۷۴۶۴۱۸	۷۴
۳	(۵۱۲, ۴۰۴, ۲۹۷۱)	-۹۵۹/۶۳۵۸	۳/۰۷۴۱۷۶	۲۹۷
۴	(۵۱۲, ۴۰۴, ۲۵۱)	-۹۵۹/۶۴۰۲	۰/۱۶۴۵۰۸	۱۷
۵	(۵۱۲, ۴۰۴, ۲۳۶۴)	-۹۵۹/۶۴۰۶	۴/۲۷۴۹۵۲	۴۰۷
۶	(۵۱۲, ۴۰۴, ۲۵۵۲)	-۹۵۹/۶۴۶۴	۲/۰۲۸۹۵۳	۱۹۵
۷	(۵۱۲, ۴۰۴, ۳۲۳۳)	-۹۵۹/۶۳۱۱	۱۳/۲۵۲۴۵۵	۶۷۹
۸	(۵۱۲, ۴۰۴, ۱۶۷۹)	-۹۵۹/۶۳۶	۰/۵۷۷۱۱۳	۵۴
۹	(۵۱۲, ۴۰۴, ۲۴۹۲)	-۹۵۹/۶۴۰۳	۳/۹۷۸۸۹۱	۴۲۲
۱۰	(۵۱۲, ۴۰۴, ۲۱۶۰)	-۹۵۹/۶۴۰۴	۷/۹۵۹۲۹۵	۷۸۴



شکل ۲ - ۱۲: نمودار تغییرات جواب بهینه بر حسب تعداد تکرار چرخه

شکل ۲ - ۱۲ و شکل ۲ - ۱۳ نتایج یکی از اجراهای کد می‌باشد. در این اجرای نتیجه بهینه برابر -959.6393 و تعداد تکرار چرخه برابر ۷۴ بار می‌باشد. با توجه به نتایج حاصل از ده بار اجرای برنامه‌ی مربوط به الگوریتم کرم شبتاب متوجه می‌شویم در زمانی مشابه با الگوریتم غذایابی باکتریایی ولی با تعداد تکرار چرخه خیلی بیشتر به جواب مطلوب می‌رسد. هر چند با توجه به شکل ۲ - ۱۲ الگوریتم جهش‌های خیلی زیادی دارد تا بتواند به کمینه سراسری خود برسد و این به معنای این است که می‌تواند در موقعی الگوریتم بهیچ عنوان به کمینه‌ی سراسری تابع همگرا نشود.



شکل ۲ - ۱۳: جایگزینی کرم‌های شبتاب در حالت اولیه و نهایی در نمایش ۲ بعدی

جهش‌ها و عدم همگرایی در این الگوریتم را می‌توانیم در پیوست پ-۴ مشاهده کنیم که تنها حدود ۱۰٪ از کرم‌ها در تکرارنهایی توانسته‌اند به نقطه‌ی مینیمم نهایی نزدیک باشند. با مقایسه‌این مورد می‌توانیم متوجه شویم که قدرت همگرایی خیلی کمتری نسبت به الگوریتم غذایابی باکتریایی دارد. در انتها نیز لازم به ذکر است که هر چند پارامترها از روی مقاله برداشت شده‌اند اما با آزمون و خطا به نظر می‌رسید که تغییر پارامترها تاثیر چندانی بر جواب ندارند.

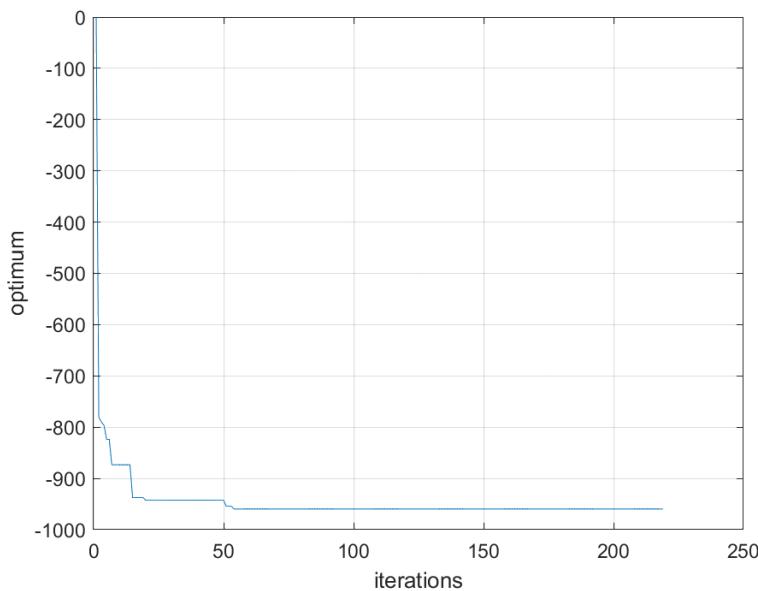
۲- ۳- الگوریتم جستجوی فاخته

برنامه‌ی مربوط به جستجوی فاخته را در مطلب ده بار اجرا کرده و مقدار بهینه، تعداد تکرار چرخه، مدت زمان اجرا کد و موقعیت ذره بهینه را استخراج کرده و در جدول ۸-۲ به ترتیب وارد کرده‌ایم:

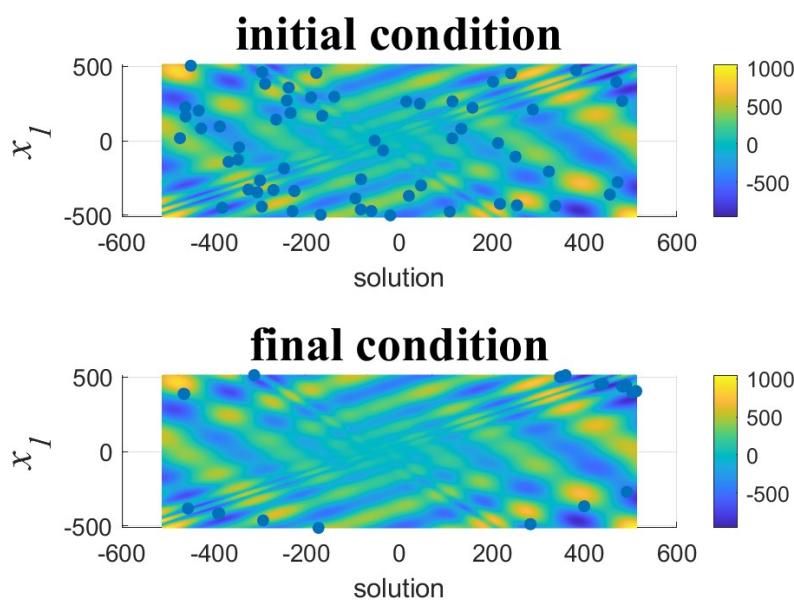
جدول ۲ - نتایج به دست آمده از الگوریتم جستجوی فاخته

شماره فهرست	موقعیت بهینه جهانی	نتیجه بهینه جهانی	زمان اجرا(s)	تعداد تکرار حلقه
۱	(۵۱۲, ۴۰۴, ۱۵۳۴)	-۹۵۹/۶۳۳۷	۰/۲۲۱۱۰۵	۲۱۹
۲	(۵۱۲, ۴۰۴, ۲۸۳۶)	-۹۵۹/۶۳۷۶	۰/۱۷۷۴۷۶	۱۰۹
۳	(۵۱۲, ۴۰۴, ۲۵۱۴)	-۹۵۹/۶۴۰۲	۰/۱۵۳۱۱۹	۱۵۶
۴	(۵۱۲, ۴۰۴, ۲۰۲۳)	-۹۵۹/۶۳۹۷	۰/۲۰۰۳۰۳	۱۸۲
۵	(۵۱۲, ۴۰۴, ۲۳۱۷)	-۹۵۹/۶۴۰۷	۰/۰۱۷۱۴۵	۱۷
۶	(۵۱۲, ۴۰۴, ۲۸۸۴)	-۹۵۹/۶۳۷	۰/۰۸۵۰۴۳	۷۶
۷	(۵۱۲, ۴۰۴, ۲۵۷۹)	-۹۵۹/۶۳۹۹	۰/۱۴۰۸۵۴	۱۲۵
۸	(۵۱۲, ۴۰۴, ۲۴۰۴)	-۹۵۹/۶۴۰۶	۰/۰۸۸۳۸۶	۱۲۱
۹	(۵۱۲, ۴۰۴, ۲۰۳۶)	-۹۵۹/۶۳۹۸	۰/۱۶۵۹۸۳	۲۱۹
۱۰	(۵۱۲, ۴۰۴, ۲۹۹۶)	-۹۵۹/۶۳۵۴	۰/۲۲۰۵۹۱	۳۹۰

با استفاده از جواب‌های به دست آمده و میانگین‌گیری نتایج، مقدار بهینه میانگین برابر ۹۵۹.۶۳۸۵- مدت زمان اجرای برنامه برابر ۱۴۷.۰ ثانیه و تعداد میانگین تکرار چرخه برابر ۱۳۶ است. شکل ۲ - ۱۴ و شکل ۲ - ۱۵ نمودارهای میانگین برنامه ۱۴۷.۰ ثانیه و تعداد میانگین تکرار چرخه برابر ۱۳۶ است. در این اجرا، نتیجه بهینه برابر ۹۵۹.۶۳۳۷- و تعداد تکرار چرخه ۲۱۹ می‌باشد. طبق نتایج بدست آمده از مدل‌سازی الگوریتم و بررسی نتایج تابع محک در این الگوریتم، مشاهده می‌کنیم که میانگین نتایج دقیق حدود ۰.۰۰۲ دارند که نتیجه مطلوبی است. همچنین مشاهده می‌شود که مدت زمان اجرای این کد خیلی پایین است و با حداقل زمان ممکن به نتیجه مطلوب می‌رسیم.



شکل ۲ - ۱۴: نمودار تغییرات جواب بهینه بر حسب تعداد تکرار چرخه



شکل ۲ - ۱۵: جای‌گیری لانه‌ها در حالت اولیه و نهایی در کانتور دو بعدی

اما با این حال از شکل ۲ - ۱۵ در می‌یابیم که این سرعت مانع همگرايی قوى لانه‌ها شده است. هر چند لازم به ذکر است که در این الگوریتم نیز مانند الگوریتم‌های دیگر مقداری خطأ در برخی لانه‌ها وجود دارد که به سبب وجود کمینه‌های محلی بسیار درتابع شانه تخم مرغی است و به این علت، برخی لانه‌ها در کمینه‌های محلی گیر می‌کنند و دچار خطأ می‌شوند. همچنین مشاهده می‌شود که کمینه هر چرخه به صورت دائمًا نزولی است و همگرايی به جواب کمینه نهایی صورت می‌گيرد و کمینه به صورت تصادفي تغيير نمی‌کند. اين امر بدليل وجود شرطی در داخل الگوریتم برای بررسی ارزش لانه‌هایی است که بطور تصادفي جابجا شده‌اند تا اگر نقاط جدید از هزينه‌ی بيشتری برخوردار بودند

نقطه‌ی جدید پذیرفته نشود و نقطه‌ی قبل برای لانه تعیین شود. برخلاف سرعت و نمودار همگرایی مناسب، با توجه به جدول ۲ - ۲ می‌توان گم شدن فاخته‌ها در کمینه محلی و همچنین سرعت کم همگرایی ذرات را مشاهده کرد. همچنین برداشت می‌شود که حدود ۱۵٪ داده‌ها در همسایگی کمینه جهانی هستند و خطای سایر ذرات بالاست. تمامی نتایج ذکر شده بدین معنی است که قدرت اکتشاف بسیار بالاتر از قدرت همگرایی است بنابراین کد بسیار سریع اجرا شده و به نتیجه با دقت مناسب می‌رسد اما درصد کمی از جمعیت می‌توانند به همسایگی کمینه جهانی برسند. پارامترهای کمی در این الگوریتم دخیل هستند که یکی از آن‌ها نرخ اکتشاف است که مشخصاً با افزایش این پارامتر قدرت اکتشاف قوی‌تر و همگرایی کمتر خواهد شد و الگوریتم بیشتر به سمت حرکات تصادفی می‌رود، اما این پارامتر تاثیر چشمگیری در نتایج ما نداشت. اما مشاهده می‌شود که با افزایش اندک بتا می‌توان سرعت همگرایی را بالا برد. بطورکلی برداشت می‌شود که این الگوریتم برای تابع محک مدنظر ما نتیجه بسیار مناسبی در حداقل زمان فراهم کرده که بسیار مطلوب است.

۲ - ۴ الگوریتم امپریالیست رقابتی

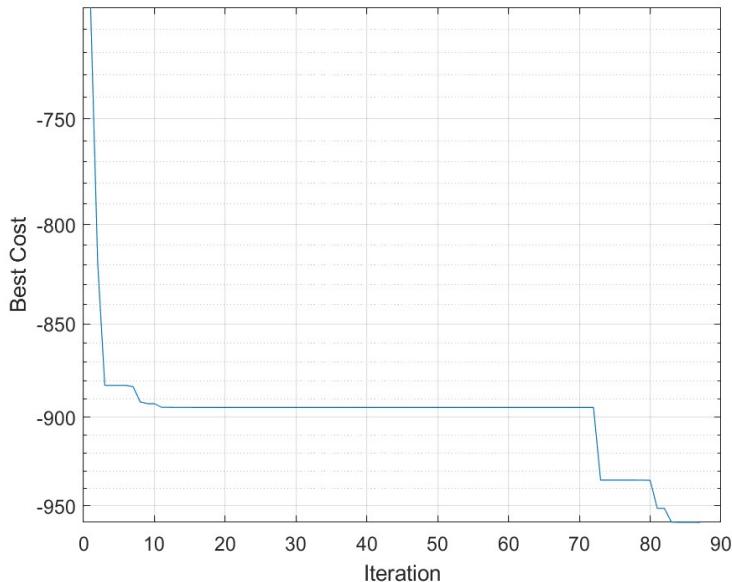
کد را ده بار اجرا کرده و مقدار بهینه، تعداد تکرار چرخه، مدت زمان اجرای کد و موقعیت‌های ذره بهینه را استخراج کرده و در جدول ۲ - ۹ به ترتیب وارد کرده‌ایم: با استفاده از جواب‌های به دست آمده و میانگین‌گیری نتایج، مقدار بهینه میانگین برابر 959.6382 ، مدت زمان اجرا میانگین برابر 1741703.0 ثانیه و تعداد میانگین تکرار چرخه برابر 33 است.

جدول ۲ - ۹: نتایج بدست آمده از الگوریتم امپریالیست رقابتی

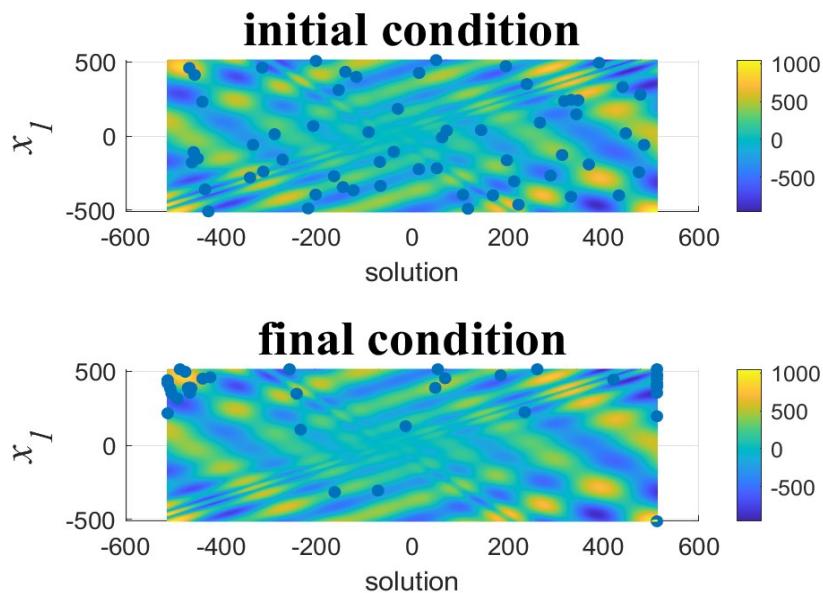
شماره فهرست	موقعیت بهینه جهانی	نتیجه بهینه جهانی	زمان اجرا (s)	تعداد تکرار حلقه
۱	(۵۱۲, ۴۰۴, ۱۷)	-۹۵۹/۶۳۷۳	۰/۱۶۴۴۰۷	۱۳
۲	(۵۱۲, ۴۰۴, ۱۶)	-۹۵۹/۶۳۵۴	۰/۱۹۴۱۸۵	۲۶
۳	(۵۱۲, ۴۰۴, ۲۵۲۹)	-۹۵۹/۶۴۰۲	۰/۲۲۹۹۷۴	۴۸
۴	(۵۱۲, ۴۰۴, ۲۸۳۶)	-۹۵۹/۶۳۷۶	۰/۴۷۳۵۵۶	۸۸
۵	(۵۱۲, ۴۰۴, ۱۰۵۷)	-۹۵۹/۶۳۴۱	۰/۱۸۰۰۵۴	۳۸
۶	(۵۱۲, ۴۰۴, ۲۸۹۹)	-۹۵۹/۶۳۶۸	۰/۰۹۴۵۴۷	۲۲
۷	(۵۱۲, ۴۰۴, ۲۵۳۲)	-۹۵۹/۶۴۰۱	۰/۰۷۱۴۱۵	۱۶
۸	(۵۱۲, ۴۰۴, ۱۹۷۱)	-۹۵۹/۶۳۹۳	۰/۰۸۸۹۵۱	۲۱
۹	(۵۱۲, ۴۰۴, ۲۲۵۷)	-۹۵۹/۶۴۰۶	۰/۱۹۸۵۹۵	۴۹
۱۰	(۵۱۲, ۴۰۴, ۲۳۸۳)	-۹۵۹/۶۴۰۶	۰/۰۴۵۵۱۹	۱۰

شکل ۲ - ۱۶ و شکل ۲ - ۱۷ نمودارهای یکی از اجراهای برنامه‌ی امپریالیست رقابتی است. در این اجرا نتیجه بهینه برابر 959.6376 و تعداد تکرار چرخه برابر 88 است. همانطور که از جدول ۲ - ۹ پیداست، جواب‌ها بطور میانگین در ده

اجرا حدود ۰.۰۰۳ با کمینه جهانی اختلاف دارند و با تعداد تکرار چرخه در هر اجرا مشاهده می‌شود برای بهینه‌سازی، سریع عمل می‌کند و این فرایند در زمان نسبتاً کوتاهی انجام می‌شود. پس هم از نظر زمانی و هم از نظر تعداد تکرار مورد نیاز برای رسیدن به کمینه جهانی به خوبی عمل می‌کند. همچنین از شکل ۲ - ۱۶ و شکل ۲ - ۱۷ مشخص است که سیر همگرایی جواب بهینه در هر چرخه تماماً نزولی است و همگرایی در جواب‌ها صورت گرفته است و از شکل جای‌گیری کشورها در فضای جستجو نیز می‌توان دریافت که ذرات به سمت بهینه تا حدی همگرا شده‌اند اما همگرایی محلی نیز در برخی نقاط وجود دارد. این همگرایی به کمینه محلی سبب شده حدود ۱۰ درصد اجراء‌ها در چرخه بماند و همگرایی به محل کمینه جهانی صورت نگیرد. در جدول پ - ۵ نیز حدود ۲۵ درصد کشورها در همسایگی جهانی هستند و سایر کمینه‌های محلی از کمینه جهانی دور هستند که به همین علت با بالا بردن نرخ انقلاب و حذف و تصادفی کردن جامعه، در کمینه‌های فضای جستجو گم شده و بعد از زمان زیاد اجرا، دقت خود را از دست می‌دهیم و با پایین بردن آنها نیز احتمال حرکت ذرات را کم می‌کنیم.



شکل ۲ - ۱۶: نمودار تغییرات جواب بهینه بر حسب تعداد تکرار چرخه



شکل ۲ - ۱۷: جایگیری کلونی‌ها در حالت اولیه و نهایی در کانتور دو بعدی

اکتشاف و همگرایی باهم در تناسب خوبی هستند و تنها مشکل این الگوریتم احتمال ده درصدی است که به جواب نرسد. اما با افزایش تعداد جمعیت ذرات می‌توان احتمال را کمتر کرد. هر چند این مورد باعث افزایش بار محاسباتی و هزینه بر بودن بهینه‌سازی می‌شود. علاوه بر آن با توجه به اینکه هدف این پژوهه صحه‌گذاری میان الگوریتم‌های فرآیندی انتخاب شده می‌باشد، نمی‌توان از جمعیت‌های گوناگونی در مقایسه‌این الگوریتم‌ها استفاده کرد.

۲- ۵- الگوریتم ماهی الکتریکی

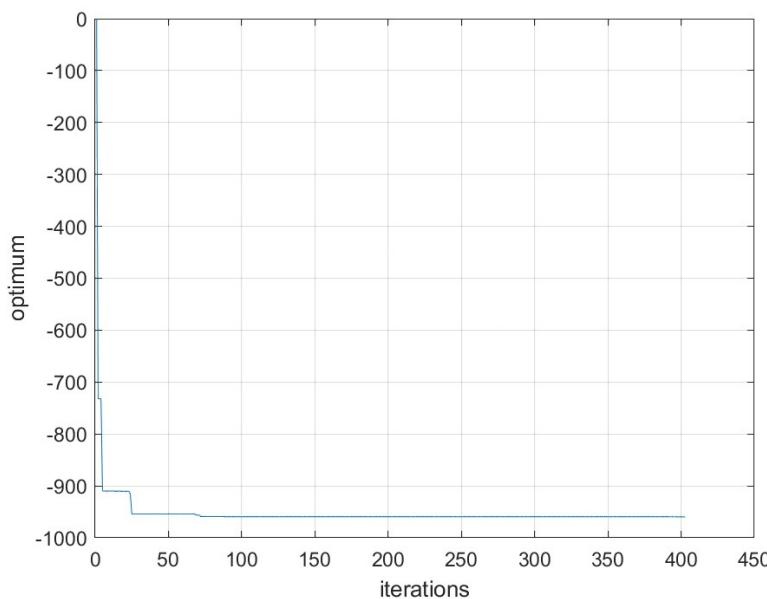
کد مربوط به این الگوریتم را همانند الگوریتم‌های دیگر ده بار در فضای مطلب اجرا می‌کنیم و مقدار تعداد تکرار چرخه، مدت زمان اجرای برنامه و موقعیت‌های ذره بهینه را استخراج کرده و در جدول ۲ - ۱۰ به ترتیب وارد کرده‌ایم:

جدول ۲ - ۱۰: تاییج بدست‌آمده از الگوریتم بهینه‌سازی ماهی الکتریکی

شماره فهرست	موقعیت بهینه جهانی	نتیجه بهینه جهانی	زمان اجرا(s)	تعداد تکرار حلقه
۱	(۵۱۲, ۴۰۴, ۲۰۳)	-۹۵۹/۶۳۹۷	۲/۹۱۴۲۵۱	۵۷۴
۲	(۵۱۲, ۴۰۴, ۲۷۹۷)	-۹۵۹/۶۳۸۱	۰/۴۸۲۱۱۲	۸۲
۳	(۵۱۲, ۴۰۴, ۱۴۰۲)	-۹۵۹/۶۳۲۱	۲/۰۰۳۶۱۵	۴۰۳
۴	(۵۱۲, ۴۰۴, ۱۷۸)	-۹۵۹/۶۳۷۴	۱/۲۴۶۴۷۸	۲۲۳
۵	(۵۱۲, ۴۰۴, ۱۸۴۹)	-۹۵۹/۶۳۸۲	۲/۱۲۶۷۰۳	۴۳۱
۶	(۵۱۲, ۴۰۴, ۱۶۷۷)	-۹۵۹/۶۳۶	۰/۳۱۰۹۵۳	۴۸

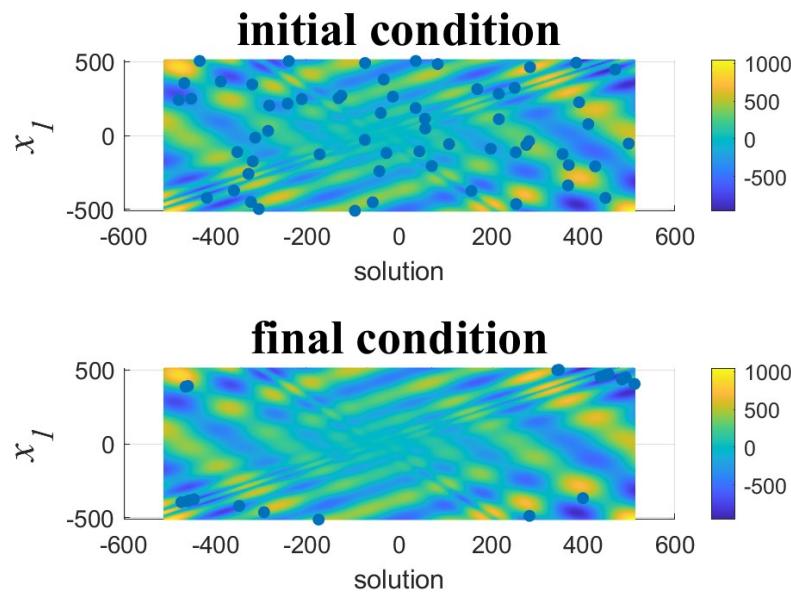
۷۴	۰/۶۱۴۱۱۶	-۹۵۹/۶۳۹۳	(۵۱۲, ۴۰۴, ۱۹۶۸)	۷
۲۳۴	۱/۷۳۸۵۸۱	-۹۵۹/۶۳۱۸	(۵۱۲, ۴۰۴, ۳۲)	۸
۳۹۷	۲/۸۹۴۱۸۲	-۹۵۹/۶۳۳۴	(۵۱۲, ۴۰۴, ۱۵۲)	۹
۱۸۲	۱/۴۱۱۶۲۶	-۹۵۹/۶۳۶۱	(۵۱۲, ۴۰۴, ۱۶۸۷)	۱۰

با استفاده از جواب‌های بدست‌آمده و میانگین‌گیری نتایج، نتیجه بهینه میانگین برابر -959.63621 ، مدت زمان اجرای میانگین برابر 1.574 ثانیه و تعداد میانگین تکرار چرخه برابر 264 است. شکل ۲ - ۱۸ و شکل ۲ - ۱۹ نمودارهای یکی از اجراهای کد می‌باشد. در این اجرا نتیجه بهینه برابر -959.6321 و تعداد تکرار چرخه برابر 403 است.



شکل ۲ - ۱۸: نمودار تغییرات جواب بهینه بر حسب تعداد تکرار چرخه

برحسب نتایج، میانگین خطای نتیجه با کمینه جهانی 0.004 است. این نتایج در مدت زمان اندکی به دست آمده‌اند ولی تعداد دفعات تکرار چرخه به نسبت زیاد است. این قضیه سبب شده‌است تا هنگام رسیدن به جواب مطلوب همگرایی با ضریب خوبی صورت گیرد. برحسب نمودارها، سیر نزدیکی جواب به کمینه جهانی نزولی است و جواب همواره در حال بهینه‌سازی و کاهش است و همزمان ذرات نیز تا حد خیلی خوبی در قسمت کمینه جهانی مرکز می‌شوند. مطابق جدول پ - ۳، تنها حدود 7% ماهی‌ها حدودا در همسایگی کمینه جهانی هستند و فاصله آن‌ها از جواب نسبتاً کم است. سایر ذرات در یک یا دو کمینه محلی دیگر جمع شده‌اند که نشان‌دهنده‌ی همگرایی جمعیت کمتری در نقطه‌ی کمینه سراسری می‌باشد. به طور کلی همگرایی در این الگوریتم بالاست.



شکل ۲ - ۱۹: جای‌گیری ماهی‌ها در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی

بعد از اجرای کد و با استفاده از نتایج به دست آمده در بخش قبلی می‌توان نتیجه‌گیری کرد این الگوریتم به نسبت برای بهینه‌سازی تابع محک موفق بوده است.

۲- ۶- جمع‌بندی

در آخر با بررسی کلی پارامترهای الگوریتم‌های مورد بررسی می‌توان به این نتیجه رسید که برخلاف الگوریتم کرم شبتاب سایر الگوریتم‌ها همگرایی به کمینه‌ی جهانی خود را بهخوبی حفظ کرده‌اند و برای تست تابع محک مورد قبول واقع شده‌اند. علت کلی این استدلال در زیر توضیح داده شده است و بطور خلاصه نقاط ضعف و قوت الگوریتم‌ها ذکر شده‌است. با توجه به جای‌گیری نادرست و پراکنده جواب‌ها و همچنین قله‌های تصادفی الگوریتم کرم شبتاب در شکل ۲ - ۱۲ می‌توان به این نتیجه رسید که این الگوریتم مناسب تابع محک مدنظر نیست و همگرایی بهخوبی صورت نمی‌گیرد. در الگوریتم غذایابی باکتریایی، با توجه به حلقه‌های متعدد در هر چرخه، در ابتدا باکتری‌ها به سرعت به کمینه جهانی نزدیک می‌شوند. پس از آن اگر حلقه حذف-پراکنده‌گی بیشتر تکرار شود، سبب پراکنده‌گی باکتری‌ها شده و اگر حلقه تولیدمثل بیشتر تکرار گردد، سبب متوقف شدن باکتری‌ها در کمینه محلی نادرست می‌شود. بنابراین نرخ تفاوت بهینه در گذر چرخه‌ها بسیار کم است. با این وجود با توجه به همگرایی قوی، دقت مناسب و تکرار کم، این الگوریتم، الگوریتمی مناسب در نظر گرفته می‌شود. هر چند مدت زمان اجرا برنامه به نسبت سایر الگوریتم‌ها بالا

می‌باشد، که این می‌تواند در شیوه نگارش کد مربوط به این الگوریتم باشد. الگوریتم ماهی الکتریکی نیز همانگونه که در شکل ۲ - ۱۸ مشاهده می‌شود، روندی نزولی با نرخ مناسبی همانند الگوریتم غذایابی باکتریابی دارد و این فرآیند در زمان کوتاه‌تری نیز اجرا می‌شود (حوالی ۴٪ زمان کمتری نیاز دارد). هر چند تعداد تکرار حلقه برای رسیدن به جواب مطلوب حدود دو برابر بیشتر از الگوریتم غذایابی باکتریابی می‌باشد و جمعیت ذرات نزدیک به نقطه کمینه سراسری نیز بسیار کمتر از الگوریتم غذایابی است (۷٪ به ۵۰٪) اما با توجه به نرخ همگرایی مناسب و زمان مناسب، این الگوریتم نیز در بهینه‌سازی تابع محک به خوبی عمل کرده است. الگوریتم امپریالیست رقابتی با توجه به کنترل ارزش کلونی‌هایی که انقلاب کرده‌اند توانسته است نرخ همگرایی مطلوبی از خود نشان دهد و در زمانی بسیار کوتاهی که تقریباً یک هشتم زمان اجرای الگوریتم ماهی الکتریکی است به جواب مطلوب برسد. علاوه بر آن تعداد حلقه‌های مورد نیاز برای رسیدن به جواب مطلوب در این الگوریتم بسیار کاهش یافته و از نظر هزینه محاسباتی الگوریتم بسیار بهینه‌ای محسوب می‌شود هر چند نسبت به الگوریتم غذایابی باکتریابی نتوانسته است تعداد زیادی از ذرات خود را به سمت کمینه سراسری بکشاند. در نتیجه می‌توان گفت که این الگوریتم به خوبی توانسته در بهینه‌سازی تابع محک عمل کند و با صرف نظر از موقعیت جمعیت ذرات آن می‌توان گفت بهترین عملکرد را داشته است. در جستجوی فاخته با توجه به پرواز لوى تصادفی، سرعت اکتشاف همانند الگوریتم امپریالیست بالاست اما تعداد تکرار حلقه‌ها برای رسیدن به پاسخ مطلوب مقدار به نسبت بالایی است (تقریباً پنج برابر الگوریتم امپریالیست و غذایابی باکتریابی) و این مورد می‌تواند نشان‌دهنده‌ی همگرایی پایین‌تر باشد هر چند با توجه به دقت بالاتر این الگوریتم و جمعیت حوالی ۲۰٪ ای در نقطه‌ی کمینه سراسری، این الگوریتم را می‌توان موفق در یافتن بهینه جهانی تابع محک شمرد. بطورکلی اگر الگوریتم نرخ نزولی مستمری در تابع هزینه خود داشته باشد، هرچه الگوریتم کنترل باشد دقت بالاتری دارد و هرچه سریع‌تر باشد قدرت اکتشاف بیشتری دارد.

۲-۳- تحلیل نتایج حاصل از استفاده از ورودی‌های غیر تصادفی

در این بخش با استفاده از داده‌ها و نمودارهای به دست آمده، الگوریتم‌ها را برای تابع محک شانه تخم مرغی با یکدیگر مقایسه کرده و ارزیابی می‌کنیم. تفاوت این بخش با بخش قبلی در ورودی‌های الگوریتم می‌باشد، در این بخش ورودی‌ها دیگر تصادفی نیستند و همانطور که در بخش‌های توضیح الگوریتم‌ها با ورودی غیر تصادفی توضیح داده شد ورودی‌ها به هشت مدل مختلف تقسیم شده‌اند و قصد داریم تا با مقایسه‌ی عملکرد الگوریتم‌ها با ورودی‌های یکسان، به مقایسه‌ی عادلانه‌تر و بهتری برسیم. هرچند که نحوه نگارش برنامه هر کدام از الگوریتم‌ها و خطاهای سخت‌افزاری محتمل در طی اجرای برنامه‌ها می‌تواند در این نتایج انحرافاتی ایجاد کند. در این بخش هر الگوریتم ۵ بار بطور

اتوماتیک با هر کدام از دسته‌های ورودی‌ای که در نظر گرفته شده است اجرا می‌شود و با توجه به بیشینه، کمینه و میانگین زمان اجرا، بیشینه و کمینه و میانگین تعداد تکرار حلقه‌ها و میانگین دقت الگوریتم‌ها به مقایسه آن‌ها می‌پردازیم. باید توجه داشت که فاکتورهای مورد بررسی در این الگوریتم‌های ممکن است هیچ رفتار قابل مقایسه شدنی نداشته باشند و در این موارد تلاش می‌شود تنها با اشاره به مقادیر مورد بررسی از این موارد گذر کرده و تمرکز را به داده‌های قابل ارزیابی قرار دهیم. همچنین داده‌های حاصل از اجرای برنامه‌ها نیز در پیوست ج به تفکیک هر الگوریتم، و هر اجرا آورده شده است.

۲ - ۳ - ۱ الگوریتم غذایابی باکتریایی

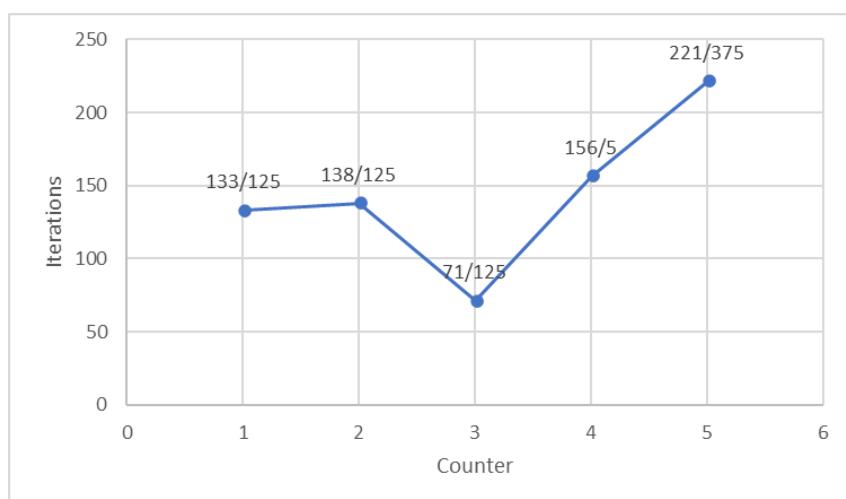
همانطور که در بخش مدل‌سازی الگوریتم با ورودی‌های از قبل تعیین شده مشاهده کردیم هر بار اجرای این برنامه به ما ۸ مدل خروجی می‌دهد که هر کدام از این خروجی‌ها شامل تعداد تکرار چرخه، زمان اجرای بهینه، مقدار بهینه الگوریتم و موقعیت کمینه سراسری می‌باشند. در نتیجه با اجرای پنج بار این برنامه به چهل دسته داده می‌رسیم که بخش ج- الگوریتم غذایابی باکتریایی داده‌های مربوط به این الگوریتم آورده شده است. در ادامه به بررسی ماکریم و مینیم این داده‌ها از منظر تعداد تکرار چرخه و زمان اجرای برنامه و همچنین به بررسی میانگین این داده‌ها می‌پردازیم. **Error!** داده‌ها از توان نظر تکرار چرخه و آخرین اجرای برنامه می‌باشد. از **Error! Reference source not found.** و **Reference source not found.** این تصاویر می‌توانیم قدرت و همگرایی و موقعیت قرارگیری ذرات در انتهای اجرای برنامه را مشاهده کنیم. در ادامه به بررسی تمام داده‌هایی که می‌توان از این داده‌ها بدست آورده می‌پردازیم و در بخش جمع‌بندی نتیجه‌ی نهایی مقایسه را شرح می‌دهیم. با مقایسه مقدار دقت الگوریتم می‌توان به این نتیجه رسید که در هر پنج اجرا دقت نتایج مشابه یکدیگر هستند و سایر داده‌ها به تنهایی نمی‌توانند در مقایسه الگوریتم‌ها کمک کننده باشند، زیرا روند مشخصی در این داده‌ها مشاهده نمی‌شود و انحراف معیار زیادی هم دارند. اما با ساده‌تر کردن داده‌ها از طریق بدست آوردن میانگین این پنج اجرا به عنوان داده‌های نماینده الگوریتم غذایابی باکتریایی، در کنار مقایسه الگوریتم‌ها بر مبنای عملکرد آن‌ها با توجه به ورودی‌های یکسان (مثلا مقایسه عملکرد الگوریتم‌ها با داده ورودی قرار گرفته شده در محور اصلی فضای مورد بررسی) می‌توان به نتیجه‌ی قابل اعتنایی رسید.

جدول ۲ - ۱۱ بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه

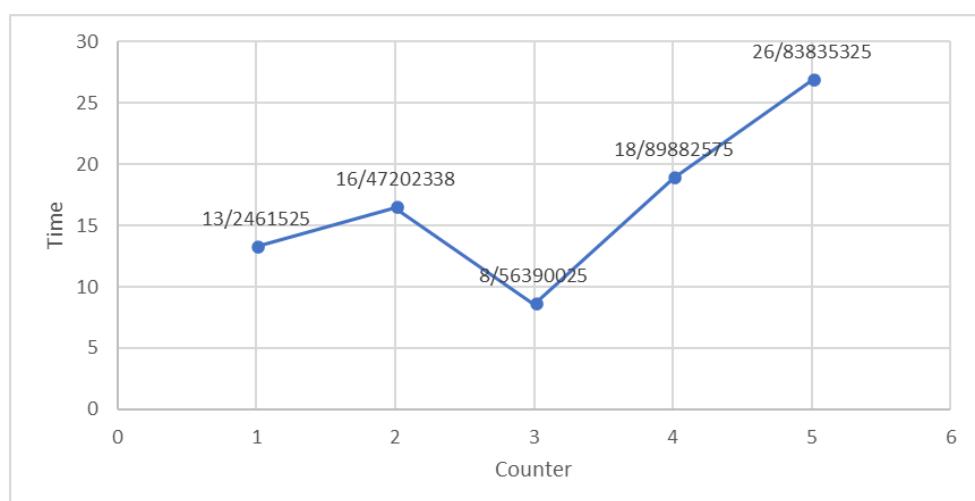
بیشینه تکرار چرخه	بیشینه زمان اجرا	کمینه تکرار چرخه	کمینه زمان اجرا	شماره اجرا
۴۴۵	۴۲.۹۸	۱۸	۱۶۳	۱
۶۸۲	۸۲.۶	۹	۱.۴۸	۲
۳۷۵	۴۵.۲۱	۶	۰.۷۵	۳
۳۱۷	۳۹.۲۷	۱۵	۲.۳۲	۴
۸۹۲	۱۰۶.۵۹	۱۲	۱.۴۶	۵

جدول ۲ - ۱۲: میانگین نتایج اجرا الگوریتم غذایابی باکتریایی در هر اجرا (تمام داده‌ها میانگین هستند)

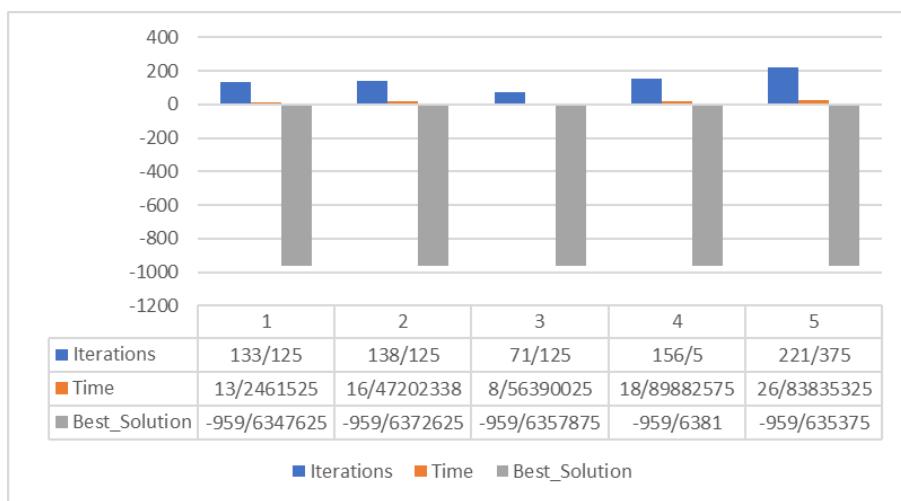
شماره اجرا	تعداد تکرار چرخه	زمان اجرا	سازی نتایج بهینه	X مربوط به نتیجه نهایی	۷ مربوط به نتیجه نهایی
۱	۱۳۳/۱۲۵	۱۳/۲۴۶۱۵۲۵	-۹۰۹/۶۳۴۲۶۲۵	۵۱۲	۴۰۴/۲۳۲
۲	۱۳۸/۱۲۵	۱۶/۴۷۲۰۲۳۳۸	-۹۰۹/۶۳۷۲۶۲۵	۵۱۲	۴۰۴/۲۴۲
۳	۷۱/۱۲۵	۸/۵۶۳۹۰۰۲۵	-۹۰۹/۶۳۵۷۸۷۵	۵۱۲	۴۰۴/۲۰۱
۴	۱۰۶/۵	۱۸/۸۹۸۸۲۵۷۵	-۹۰۹/۶۳۸۱	۵۱۲	۴۰۴/۲۶۶۰
۵	۲۲۱/۳۷۵	۲۶/۸۳۸۳۰۳۲۵	-۹۰۹/۶۳۵۳۷۵	۵۱۲	۴۰۴/۲۱۶۳



شکل ۲ - ۲۰: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم غذایابی باکتریایی

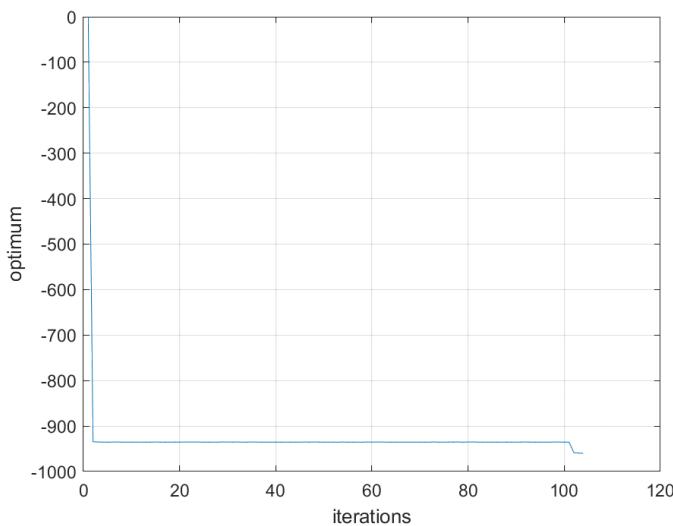


شکل ۲ - ۲۱: نمودار زمان اجرای برنامه در هر اجرای الگوریتم غذایابی باکتریایی

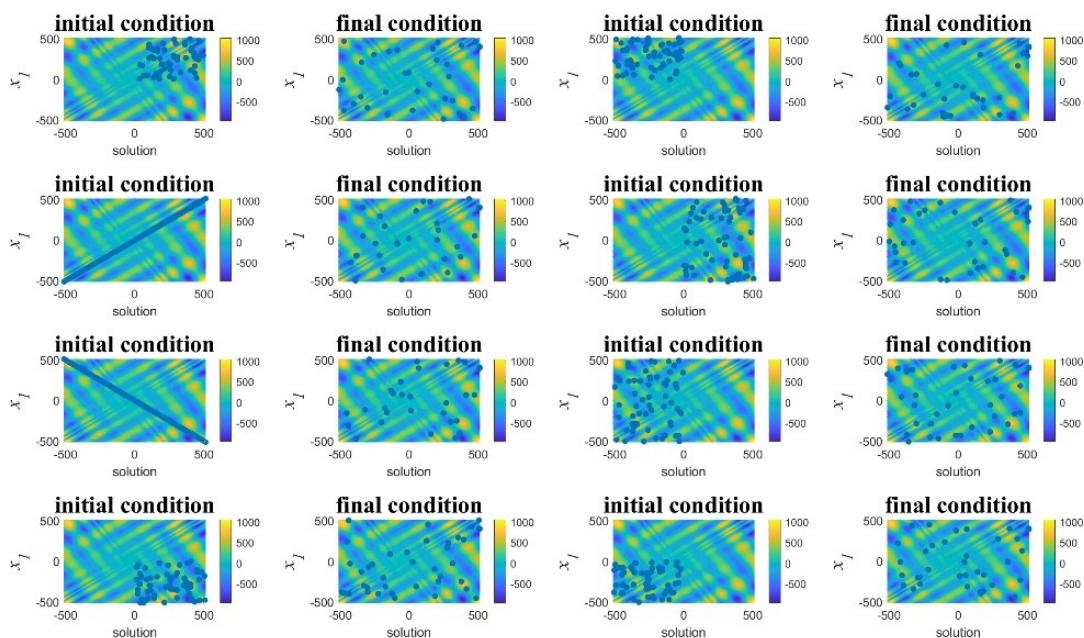


شکل ۲ - ۲۲: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم غذایابی باکتریایی

با مقایسه مقدار دقت الگوریتم می‌توان به این نتیجه رسید که در هر پنج اجرا دقت نتایج مشابه یکدیگر هستند و سایر داده‌ها به تنها بی نمی‌توانند در مقایسه الگوریتم‌ها کمک کننده باشند، زیرا روند مشخصی در این داده‌ها مشاهده نمی‌شود و انحراف معیار زیادی هم دارند. اما با ساده‌تر کردن داده‌ها از طریق بدستآوردن میانگین این پنج اجرا به عنوان داده‌های نماینده الگوریتم غذایابی باکتریایی، در کنار مقایسه الگوریتم‌ها بر مبنای عملکرد آن‌ها با توجه به ورودی‌های یکسان (مثل مقایسه عملکرد الگوریتم‌ها با داده ورودی قرار گرفته شده در محور اصلی فضای مورد بررسی) می‌توان به نتیجه‌ی قابل اعتماد رسید. در ادامه نمودارهای مرتبط با موقعیت‌های اولیه و نهایی ذرات در اجرای آخر را بررسی می‌کنیم تا نرخ همگرایی این الگوریتم و تعداد ذرات موجود در نقطه‌ی کمینه سراسری را مشاهده و در ادامه با سایر الگوریتم‌ها مقایسه کنیم.



شکل ۲ - ۲۳: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه



شکل ۲ - ۲۴: جایگیری باکتری‌ها در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی

با توجه به شکل ۲ - ۲۳ و شکل ۲ - ۲۴ می‌توان مشاهده کرد که الگوریتم پس از یک کاهش خیره کننده در هزینه‌ی تابع در ۵ تکرار اولیه نتوانسته است به دقت ۹۹٪ تا صدمین تکرار دست یابد و این مورد در نحوه قرارگیری ذرات در حالت نهایی نیز مشهود است. از آنجایی که تنها تعداد محدودی از ذرات در نقاط نزدیک به کمینه سراسری حضور داشته‌اند.

۲- ۳- ۲ الگوریتم جستجوی فاخته

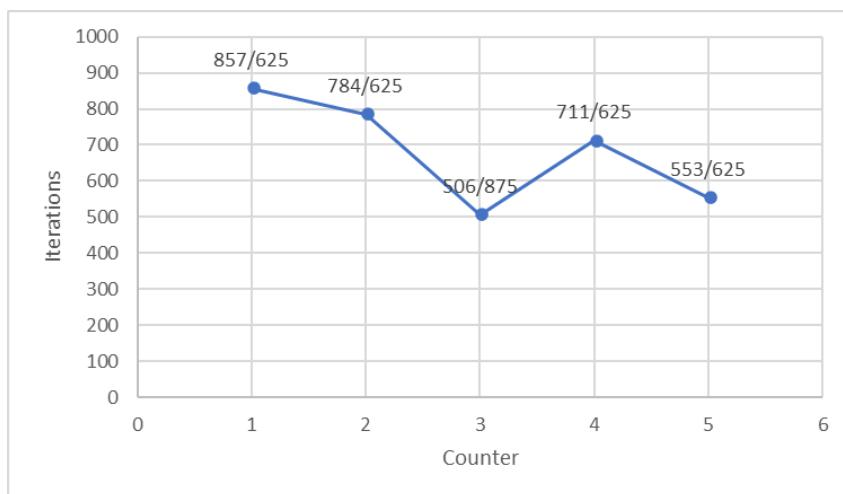
همانطور که در بخش مدل‌سازی الگوریتم با ورودی‌های از قبل تعیین شده مشاهده کردیم هر بار اجرای این برنامه به ما ۸ مدل خروجی می‌دهد که هر کدام از این خروجی‌ها شامل تعداد تکرار چرخه، زمان اجرای بهینه، مقدار بهینه الگوریتم و موقعیت کمینه سراسری می‌باشند. در نتیجه با پنج بار اجرای این برنامه به چهل دسته داده می‌رسیم که در بخش ج-۲ داده‌های مربوط به این الگوریتم آورده شده است.

جدول ۲ - ۱۳ بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه

شماره اجرا	بیشنه تکرار چرخه	بیشنه زمان اجرا	کمینه تکرار چرخه	کمینه زمان اجرا
۱	۲۰۰۲	۱.۰۲۴	۱۹۳	۰.۱۶
۲	۲۷۵۴	۱.۲۵	۲۶۲	۰.۱۲
۳	۹۶۹	۰.۶۳	۲۱۳	۰.۱۸
۴	۱۶۲۳	۰.۹۸	۲۲۰	۰.۱۵
۵	۹۳۱	۰.۶۵	۵۳	۰.۰۴

جدول ۲ - ۱۴: میانگین نتایج اجرا الگوریتم جستجوی فاخته در هر اجرا (تمام داده‌ها میانگین هستند)

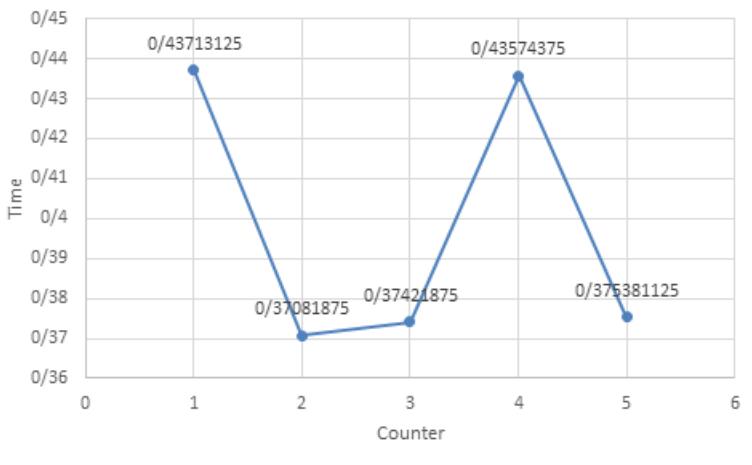
شماره اجرا	تعداد تکرار چرخه	زمان اجرا	نتایج بهینه‌سازی	X مربوط به نتیجه نهایی	Y مربوط به نتیجه نهایی
۱	۸۵۷/۶۲۵	۰/۴۳۷	-۹۵۹/۶۳۶	۵۱۲	۴۰۴/۲۲۶۹
۲	۷۸۴/۶۲۵	۰/۳۷۱	-۹۵۹/۶۳۶	۵۱۲	۴۰۴/۲۲۷
۳	۵۰۶/۸۷۵	۰/۳۷۴	-۹۵۹/۶۳۶	۵۱۲	۴۰۴/۲۴۷
۴	۷۱۱/۶۲۵	۰/۴۳۶	-۹۵۹/۶۳۳	۵۱۲	۴۰۴/۲۱۶
۵	۵۰۳/۶۲۵	۰/۳۷۵	-۹۵۹/۶۳۶	۵۱۲	۴۰۴/۲۳۲



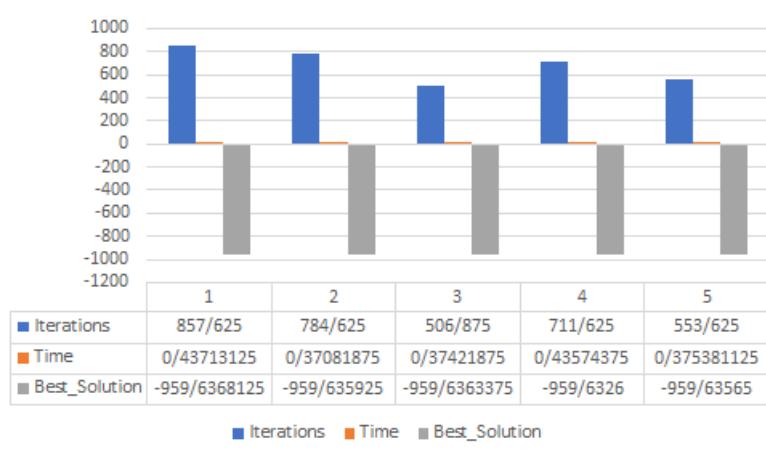
شکل ۲ - ۲۵: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم جستجوی فاخته

در ادامه به بررسی ماکریم و مینیم این داده‌ها از منظر تعداد تکرار چرخه و زمان اجرای برنامه و همچنین به بررسی

میانگین این داده‌ها می‌پردازیم. شکل ۲ - ۲۸ و شکل ۲ - ۲۹ نیز مربوط به آخرین اجرای برنامه می‌باشد. از این تصاویر می‌توانیم قدرت و همگرایی و موقعیت قرارگیری ذرات در انتهای اجرای برنامه را مشاهده کنیم. در ادامه به بررسی تمام داده‌ای که می‌توان از این داده‌ها بدست آورد می‌پردازیم و در بخش جمع‌بندی نتیجه‌ی نهایی مقایسه را شرح می‌دهیم.

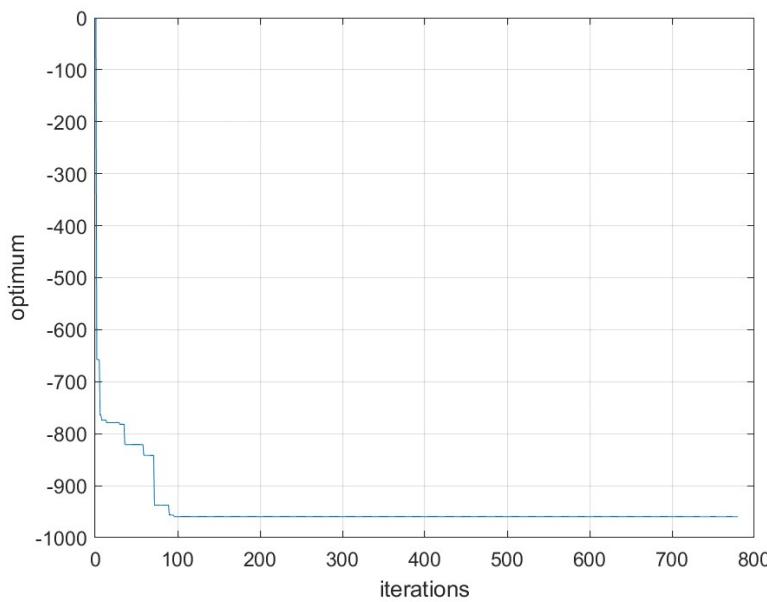


شکل ۲ - ۲۶: نمودار زمان اجرای برنامه در هر اجرای الگوریتم جستجوی فاخته

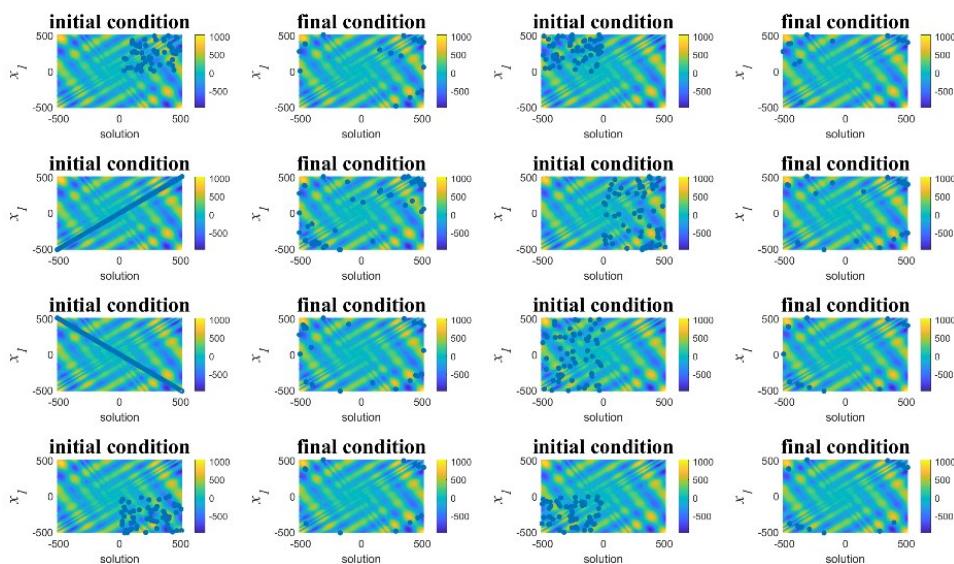


شکل ۲ - ۲۷: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم جستجوی فاخته

با مقایسه مقدار دقت الگوریتم می‌توان به این نتیجه رسید که در هر پنج اجرا دقت نتایج مشابه یکدیگر هستند و سایر داده‌ها به نهایی نمی‌توانند در مقایسه الگوریتم‌ها کمک کنند باشند، زیرا روند مشخصی در این داده‌ها مشاهده نمی‌شود و انحراف معیار زیادی هم دارند. اما با ساده‌تر کردن داده‌ها از طریق بدست آوردن میانگین این پنج اجرا به عنوان داده‌های نماینده الگوریتم جستجوی فاخته، در کنار مقایسه الگوریتم‌ها بر مبنای عملکرد آن‌ها با توجه به ورودی‌های یکسان (مثلًا مقایسه عملکرد الگوریتم‌ها با داده ورودی قرار گرفته شده در محور اصلی فضای مورد بررسی) می‌توان به نتیجه‌ی قابل اعتمایی رسید.



شکل ۲ - ۲۸: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه



شکل ۲ - ۲۹: جایگیری ذرات در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی

در ادامه نمودارهای مرتبط با موقعیت‌های اولیه و نهایی ذرات در اجرای آخر را بررسی می‌کنیم تا نرخ همگرایی این الگوریتم و تعداد ذرات موجود در نقطه‌ی کمینه سراسری را مشاهده و در ادامه با سایر الگوریتم‌ها مقایسه کنیم. با توجه به شکل ۲ - ۲۸ و شکل ۲ - ۲۹ می‌توان مشاهده کرد الگوریتم جستجوی فاخته نرخ همگرایی بهتری نسبت به غذایابی باکتریابی تا صدمین تکرار دارد. هر چند مشاهده می‌شود که پس از آن نرخ همگرایی بشدت کاهش می‌یابد و نمودار به حالت خطی در آمده است که نقطه ضعف محسوب می‌شود و با وجود آنکه تعداد ذرات در نزدیکی کمینه

سراسری به نسبت مشابهی دارد.

۲ - ۳ - ۲ الگوریتم ماهی الکتریکی

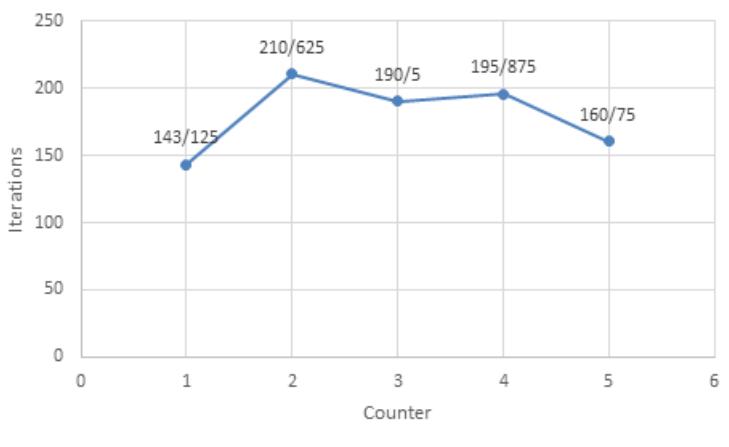
همانطور که در بخش مدل‌سازی الگوریتم با ورودی‌های از قبل تعیین شده مشاهده کردیم هر بار اجرای این برنامه به ما ۸ مدل خروجی می‌دهد که هر کدام از این خروجی‌ها شامل تعداد تکرار چرخه، زمان اجرای بهینه، مقدار بهینه الگوریتم و موقعیت کمینه سراسری می‌باشند. در نتیجه با پنج بار اجرای این برنامه به چهل دسته داده می‌رسیم که بخش ج-۳-داده‌های مربوط به این الگوریتم آورده شده است. در ادامه به بررسی ماقریم و مینیمم این داده‌ها از منظر تعداد تکرار چرخه و زمان اجرای برنامه و همچنین به بررسی میانگین این داده‌ها می‌پردازیم. شکل ۲ - ۳۳ و شکل ۲ - ۳۴ نیز مربوط به آخرین اجرای برنامه می‌باشد. از این تصاویر می‌توانیم قدرت و همگرایی و موقعیت قرارگیری ذرات در انتهای اجرای برنامه را مشاهده کنیم. در ادامه به بررسی تمام داده‌هایی که می‌توان از این داده‌ها بدست آورده می‌پردازیم و در بخش جمع‌بندی نتیجه‌ی نهایی مقایسه را شرح می‌دهیم.

جدول ۲ - ۱۵: بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه

شماره اجرا	کمینه زمان اجرا	کمینه تکرار چرخه	بیشینه زمان اجرا	بیشینه تکرار چرخه
۱	۰.۳	۲۲	۱.۴۲	۲۴۶
۲	۰.۴۶	۸۴	۱.۷۳	۳۳۸
۳	۰.۴۱	۸۶	۲.۸۷	۶۳۹
۴	۰.۴	۶۹	۲.۰۹	۴۴۶
۵	۰.۲	۳۸	۱.۳۳	۲۷۲

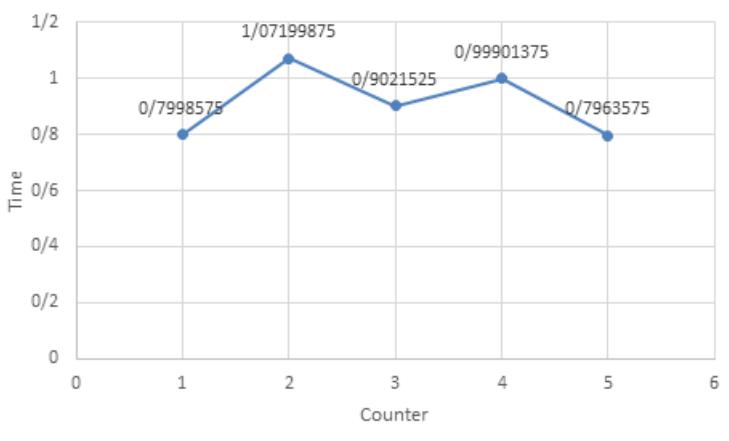
جدول ۲ - ۱۶: میانگین نتایج اجرا الگوریتم ماهی الکتریکی در هر اجرا (تمام داده‌ها میانگین هستند)

شماره اجرا	تعداد تکرار چرخه	زمان اجرا	نتایج بهینه‌سازی	X مربوط به نتیجه نهایی	Y مربوط به نتیجه نهایی
۱	۱۴۳/۱۲۵	۰/۸۰۰	-۹۵۹/۶۳۷۳	۵۱۲	۴۰۴/۲۳
۲	۲۱۰/۶۲۵	۱/۰۷۲	-۹۵۹/۶۳۷۳	۵۱۲	۴۰۴/۲۴۵
۳	۱۹۰/۵	۰/۹۰۲	-۹۵۹/۶۳۷۹	۵۱۲	۴۰۴/۲۳
۴	۱۹۵/۸۷۵	۱	-۹۵۹/۶۳۷۶	۵۱۲	۴۰۴/۲۴۵
۵	۱۶۰/۷۵	۰/۷۹۶	-۹۵۹/۶۳۵۹۵	۵۱۲	۴۰۴/۲۳



شکل ۲ - ۳۰: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم ماهی الکتریکی

با مقایسه مقدار دقت الگوریتم می‌توان به این نتیجه رسید که در هر پنج اجرا دقت نتایج مشابه یکدیگر هستند و سایر داده‌ها به تنهایی نمی‌توانند در مقایسه الگوریتم‌ها کمک کنند باشند، زیرا روند مشخصی در این داده‌ها مشاهده نمی‌شود و انحراف معیار زیادی هم دارند. اما با ساده‌تر کردن داده‌ها از طریق بدست‌آوردن میانگین این پنج اجرا به عنوان داده‌های نماینده الگوریتم ماهی الکتریکی، در کنار مقایسه الگوریتم‌ها بر مبنای عملکرد آن‌ها با توجه به ورودی‌های یکسان (مثلاً مقایسه عملکرد الگوریتم‌ها با داده ورودی قرار گرفته شده در محور اصلی فضای مورد بررسی) می‌توان به نتیجه‌ی قابل اعتنایی رسید.

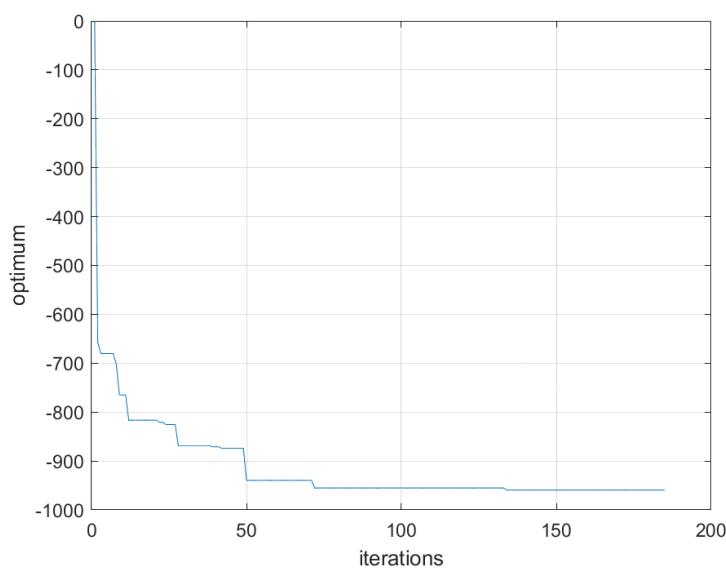


شکل ۲ - ۳۱: نمودار زمان اجرای برنامه در هر اجرای الگوریتم ماهی الکتریکی

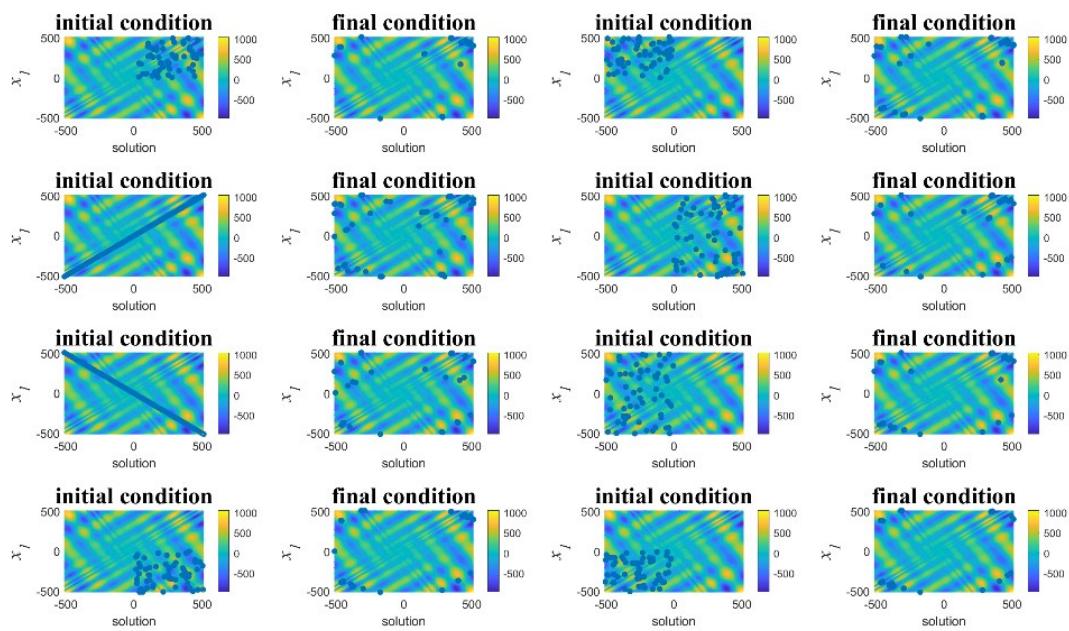


شکل ۲ - ۳۲: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم ماهی‌الکتریکی

در ادامه نمودارهای مرتبط با موقعیت‌های اولیه و نهایی ذرات در اجرای آخر را بررسی می‌کنیم تا نرخ همگرایی این الگوریتم و تعداد ذرات موجود در نقطه‌ی کمینه سراسری را مشاهده و در ادامه با سایر الگوریتم‌ها مقایسه کنیم.



شکل ۲ - ۳۳: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه



شکل ۲ - ۳۴: جایگیری ذرات در حالت اولیه و نهایی در کاترور دو بعدی مورد بررسی

با توجه به شکل ۲ - ۳۳ و شکل ۲ - ۳۴ می‌توان مشاهده کرد که روند حرکتی ذرات مشابه با الگوریتم فاخته می‌باشد با این تفاوت که این الگوریتم نرخ همگرايی بهتری دارد و توانسته است در زمانی بشدت کمتر از الگوریتم غذایابی باکتریایی ولی با تعداد تکرار چرخه مشابه به جواب مورد نظر برسد. هر چند این الگوریتم هم مانند الگوریتم‌هایی که در قبل بررسی کردیم نرخ همگرايی اش به کمینه جهانی به مرور کاهش می‌یابد و بعد از صدمین تکرار کاهش جزیی داشته است.

۲-۴-۴ الگوریتم کرم شب تاب

همانطور که در بخش مدل‌سازی الگوریتم با ورودی‌های از قبیل تعیین شده مشاهده کردیم هر بار اجرای این برنامه به ما ۸ مدل خروجی می‌دهد که هر کدام از این خروجی‌ها شامل تعداد تکرار چرخه، زمان اجرای بهینه، مقدار بهینه الگوریتم و موقعیت کمینه سراسری می‌باشند. در نتیجه با اجرای پنج بار این برنامه به چهل دسته داده می‌رسیم که بخش ج-۴-داده‌های مربوط به این الگوریتم آورده شده است. در ادامه به بررسی ماقریم و مینیمم این داده‌ها از منظر تعداد تکرار چرخه و زمان اجرای برنامه و همچنین به بررسی میانگین این داده‌ها می‌پردازیم. شکل ۲ - ۳۸ و شکل ۲ - ۳۹ نیز مربوط به آخرین اجرای برنامه می‌باشد. از این تصاویر می‌توانیم قدرت و همگرايی و موقعیت قرارگیری ذرات در انتهای اجرای برنامه را مشاهده کنیم. در ادامه به بررسی تمام داده‌هایی که می‌توان از این داده‌ها بدست آورده می‌پردازیم و

در بخش جمع‌بندی نتیجه‌ی نهایی مقایسه را شرح می‌دهیم.

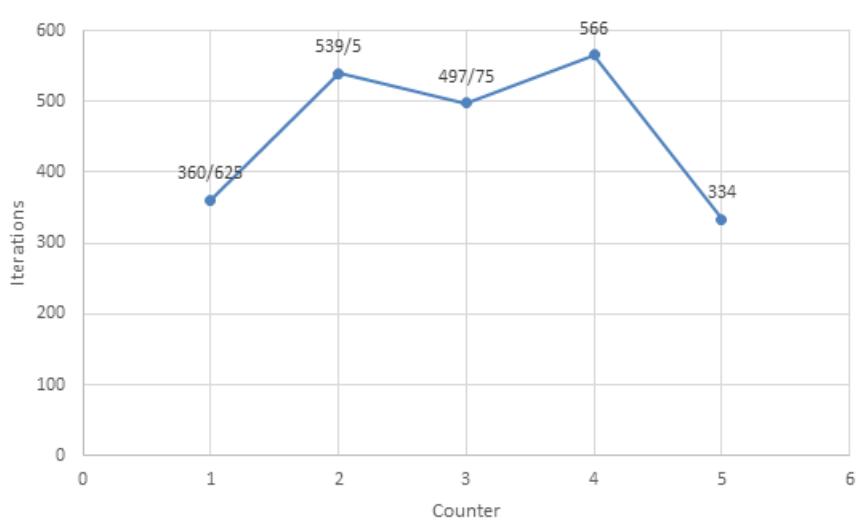
جدول ۲ - ۱۷: بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه

شماره اجرا	کمینه زمان اجرا	کمینه تکرار چرخه	بیشینه زمان اجرا	بیشینه تکرار چرخه
۱	۰.۰۴	۳۲	۱.۴۵	۱۱۷۷
۲	۰.۰۴	۳۰	۱.۸۱	۱۵۴۶
۳	۰.۱۴	۱۰۹	۱.۱۳	۹۴۸
۴	۰.۰۳	۲۸	۱.۶۰	۱۱۶۲
۵	۰.۰۶	۳۷	۰.۹۶	۶۸۰

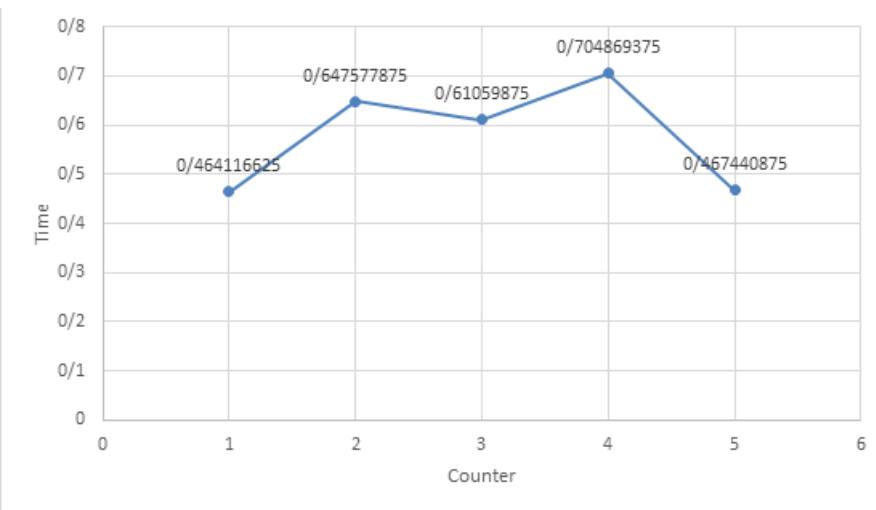
جدول ۲ - ۱۸: میانگین نتایج اجرا الگوریتم کرم شبتاب در هر اجرا (تمام داده‌ها میانگین هستند)

شماره اجرا	تعداد تکرار چرخه	زمان اجرا	نتایج بهینه‌سازی	X مربوط به نتیجه نهایی	Y مربوط به نتیجه نهایی
۱	۳۶۰/۶۲۵	۰/۴۶۴	-۹۰۹/۶۳۹	۵۱۲	۴۰۴/۲۴۵
۲	۵۳۹/۵	۰/۶۴۷	-۹۰۹/۶۳۸	۵۱۲	۴۰۴/۲۱۷
۳	۴۹۷/۷۵	۰/۶۱۰	-۹۰۹/۶۳۸	۵۱۲	۴۰۴/۲۱۰
۴	۵۶۶	۰/۷۰۵	-۹۰۹/۶۳۹	۵۱۲	۴۰۴/۲۴۱
۵	۳۳۴	۰/۴۶۷	-۹۰۹/۶۳۵	۵۱۲	۴۰۴/۲۳۰

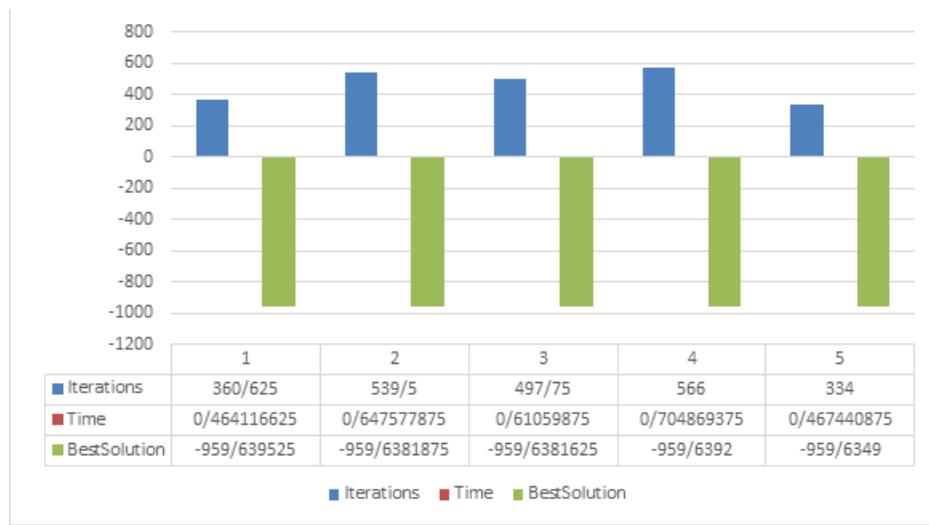
در ادامه نمودارهای مرتبط با موقعیت‌های اولیه و نهایی ذرات در اجرای آخر را بررسی می‌کنیم تا نرخ همگرایی این الگوریتم و تعداد ذرات موجود در نقطه‌ی کمینه سراسری را مشاهده و در ادامه با سایر الگوریتم‌ها مقایسه کنیم.



شکل ۲ - ۳۵: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم کرم شبتاب

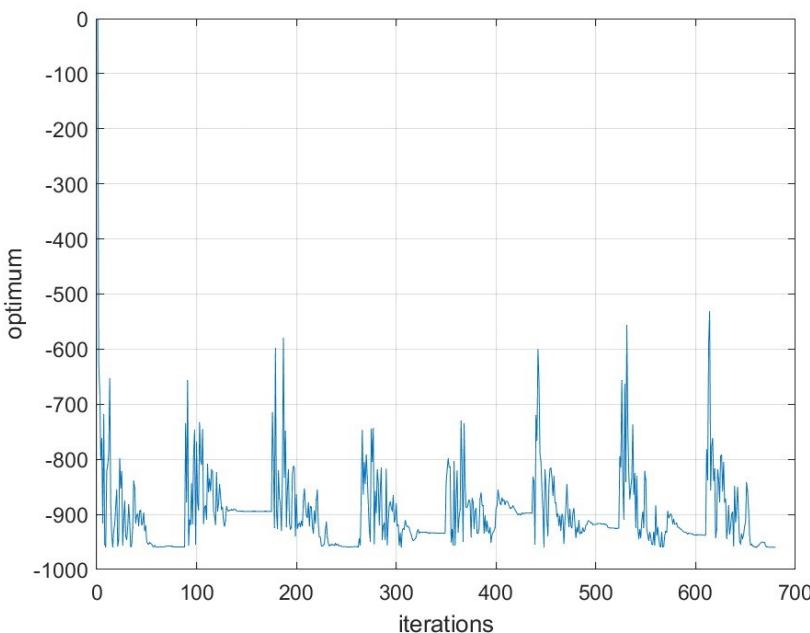


شکل ۲ - ۳۶: نمودار زمان اجرای برنامه در هر اجرای الگوریتم کرم شبتاب



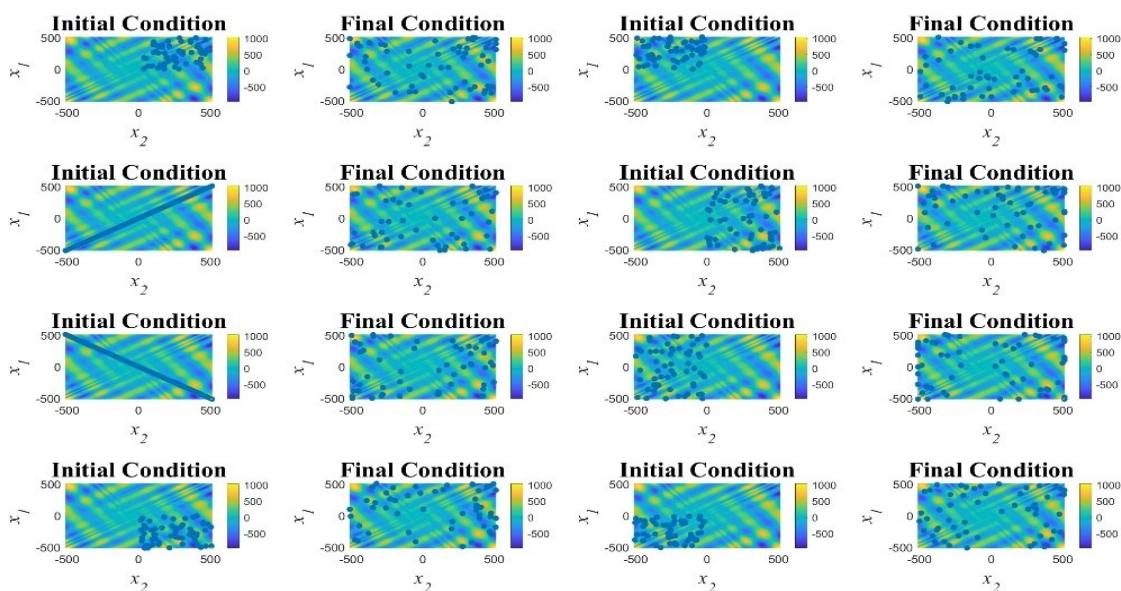
شکل ۲ - ۳۷: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم کرم شبتاب

با مقایسه مقدار دقت الگوریتم می‌توان به این نتیجه رسید که در هر پنج اجرا دقت نتایج مشابه یکدیگر هستند و سایر داده‌ها به تنهایی نمی‌توانند در مقایسه الگوریتم‌ها کمک کنند باشند، زیرا روند مشخصی در این داده‌ها مشاهده نمی‌شود و انحراف معیار زیادی هم دارند.



شکل ۲ - ۳۸: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه کرم شب تاب

اما با ساده‌تر کردن داده‌ها از طریق بدست آوردن میانگین این پنج اجرا به عنوان داده‌های نماینده الگوریتم کرم شب تاب، در کنار مقایسه الگوریتم‌ها بر مبنای عملکرد آن‌ها با توجه به ورودی‌های یکسان (مثلًا مقایسه عملکرد الگوریتم‌ها با داده ورودی قرار گرفته شده در محور اصلی فضای مورد بررسی) می‌توان به نتیجه‌های قابل اعتنایی رسید.



شکل ۲ - ۳۹: جایگیری ذرات در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی

با توجه به شکل ۲ - ۳۸ و شکل ۲ - ۳۹ مهم‌ترین نکته‌ی قابل ملاحظه عدم همگرایی الگوریتم به کمینه جهانی است. همانطور که در شکل ۲ - ۳۸ مشاهده می‌شود این الگوریتم نرخ کاهشی همواره نزولی ندارد و دائمًا در حال جهش

می‌باشد که مشکل بسیار بزرگی است و عملکرد این الگوریتم را در یافتن کمینه جهانی تابع محک زیر سوال می‌برد. باید توجه داشت که حتی با تغییر پارامترهای اکتشاف و بهره‌برداری نیز این الگوریتم به هیچ عنوان فرآیندی نزولی مطلق بخود نمی‌گیرد.

۲ - ۳ - ۵ الگوریتم امپریالیست رقابتی

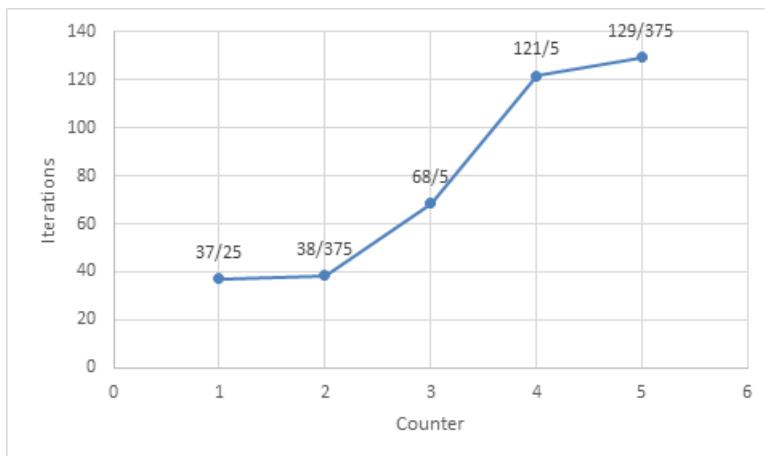
همانطور که در بخش مدل‌سازی الگوریتم با ورودی‌های از قبل تعیین شده مشاهده کردیم هر بار اجرای این برنامه به ما ۸ مدل خروجی می‌دهد که هر کدام از این خروجی‌ها شامل تعداد تکرار چرخه، زمان اجرای بهینه، مقدار بهینه الگوریتم و موقعیت کمینه سراسری می‌باشند. در نتیجه با پنج بار اجرای این برنامه به چهل دسته داده می‌رسیم که بخش ج - ۵ داده‌های مربوط به این الگوریتم آورده شده است. در ادامه به بررسی ماکریم و مینیم این داده‌ها از منظر تعداد تکرار چرخه و زمان اجرای برنامه و همچنین به بررسی میانگین این داده‌ها می‌پردازیم. شکل ۲ - ۴۳ و شکل ۲ - ۴۴ نیز مربوط به آخرین اجرای برنامه می‌باشد. از این تصاویر می‌توانیم قدرت و همگرایی و موقعیت قرارگیری ذرات در انتهای اجرای برنامه را مشاهده کنیم. در ادامه به بررسی تمام داده‌هایی که می‌توان از این داده‌ها بدست آورده می‌پردازیم و در بخش جمع‌بندی نتیجه‌ی نهایی مقایسه را شرح می‌دهیم.

جدول ۲ - ۱۹: بیشینه و کمینه زمان اجرا و تعداد تکرار چرخه

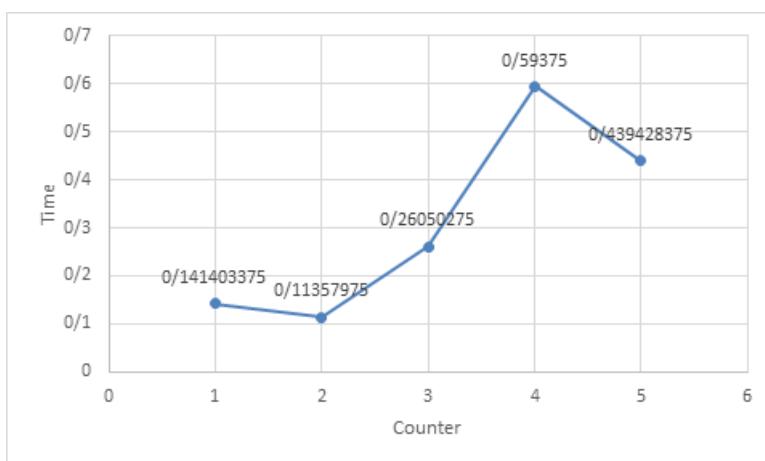
شماره اجرا	کمینه زمان اجرا	کمینه تکرار چرخه	بیشینه تکرار چرخه	بیشینه زمان اجرا
۱	۰.۰۶	۱۷	۰.۲۴	۶۰
۲	۰.۰۵	۱۲	۰.۳۴	۱۰۷
۳	۰.۰۳	۱۰	۱.۲۲	۲۷۲
۴	۰.۰۳	۹	۴.۱۷	۷۵۲
۵	۰.۰۴	۱۳	۱.۲۴	۳۶۱

جدول ۲ - ۲۰: میانگین نتایج اجرا الگوریتم امپریالیست رقابتی در هر اجرا (تمام داده‌ها میانگین هستند)

شماره اجرا	تعداد تکرار چرخه	زمان اجرا	نتایج بهینه‌سازی	X مربوط به نتیجه	Y مربوط به نتیجه
۱	۳۷/۲۵	۰/۱۴۱	-۹۵۹/۶۳۷	۵۱۲	۴۰۴/۲۶۷
۲	۳۸/۳۷۵	۰/۱۱۴	-۹۵۹/۶۳۷	۵۱۲	۴۰۴/۲۳۲
۳	۶۸/۵	۰/۲۶۱	-۹۵۹/۶۳۹	۵۱۲	۴۰۴/۲۶۲
۴	۱۲۱/۵	۰/۵۹۴	-۹۵۹/۶۳۸	۵۱۲	۴۰۴/۲۵۴
۵	۱۲۹/۳۷۵	۰/۴۳۹	-۹۵۹/۶۳۷	۵۱۲	۴۰۴/۲۳۱

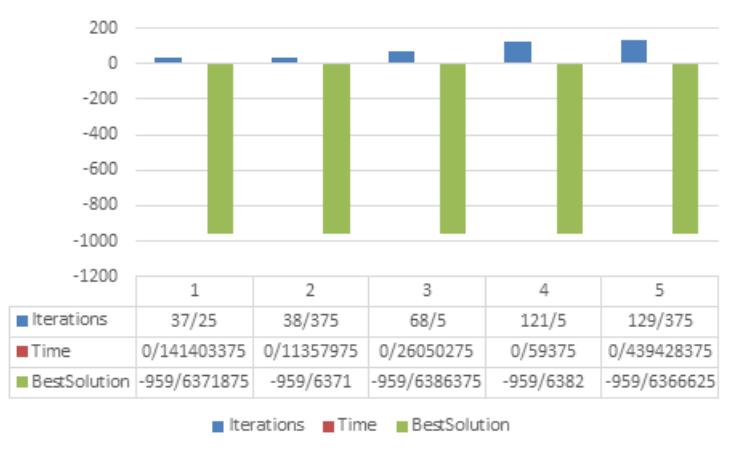


شکل ۲ - ۴: نمودار تعداد تکرار چرخه در هر اجرا در الگوریتم امپریالیست رقابتی



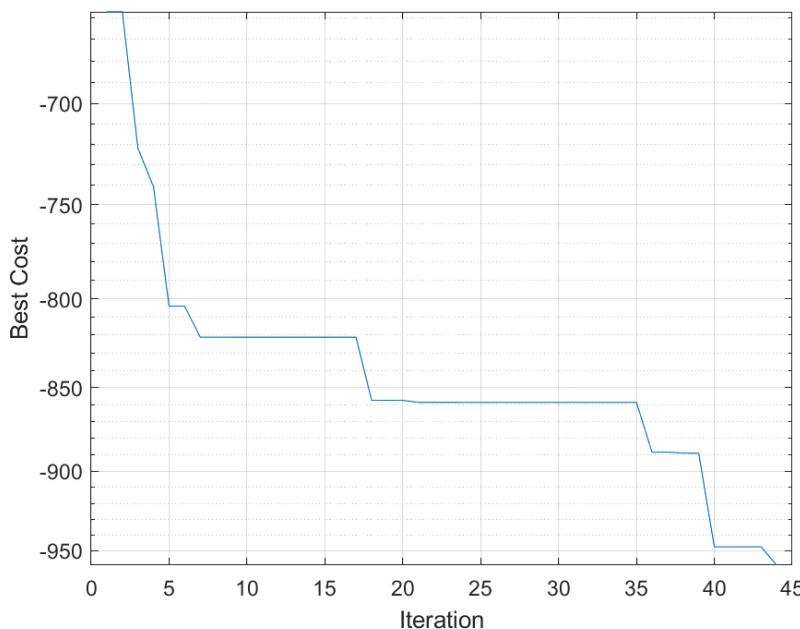
شکل ۲ - ۱: نمودار زمان اجرای برنامه در هر اجرای الگوریتم امپریالیست رقابتی

با مقایسه مقدار دقت الگوریتم می‌توان به این نتیجه رسید که در هر پنج اجرا دقت نتایج مشابه یکدیگر هستند و سایر داده‌ها به تنها نمی‌توانند در مقایسه الگوریتم‌ها کمک کنند باشند، زیرا روند مشخصی در این داده‌ها مشاهده نمی‌شود و انحراف معیار زیادی هم دارند.

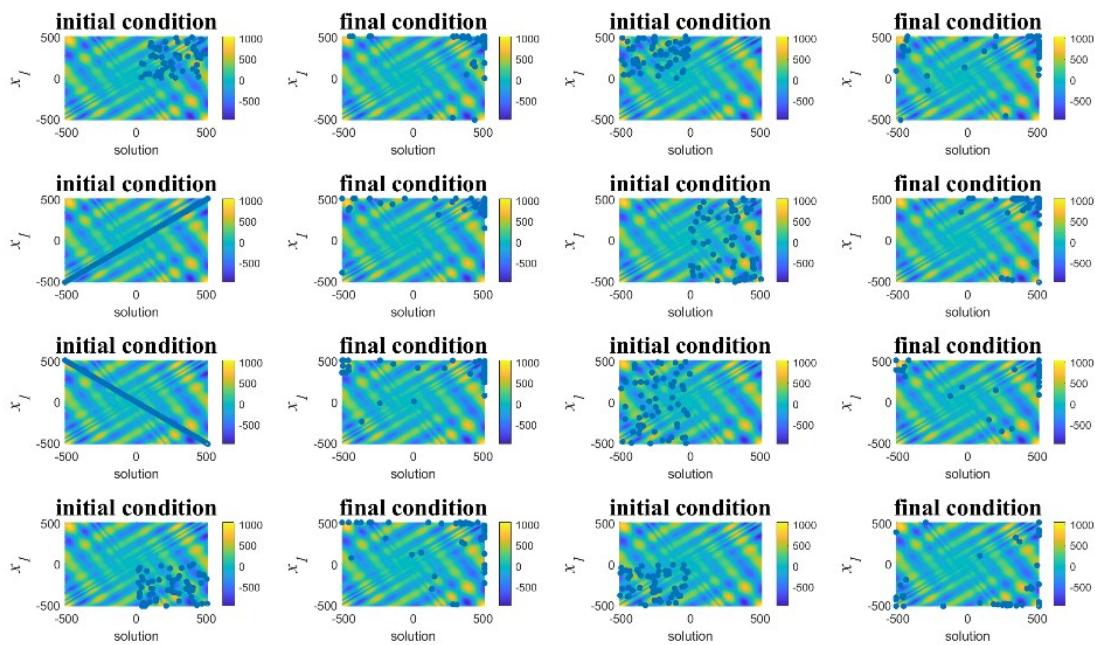


شکل ۲ - ۴۲: نمودار میله‌ای مقایسه زمان اجرا، تعداد تکرار چرخه و بهترین جواب در اجراهای الگوریتم امپریالیست رقابتی

اما با ساده‌تر کردن داده‌ها از طریق بدست آوردن میانگین این پنج اجرا به عنوان داده‌های نماینده الگوریتم کرم شب‌تاب، در کنار مقایسه الگوریتم‌ها بر مبنای عملکرد آن‌ها با توجه به ورودی‌های یکسان (مثلًا مقایسه عملکرد الگوریتم‌ها با داده ورودی قرار گرفته شده در محور اصلی فضای مورد بررسی) می‌توان به نتیجه‌ی قابل اعتمادی رسید. در ادامه نمودارهای مرتبط با موقعیت‌های اولیه و نهایی ذرات در اجرای آخر را بررسی می‌کنیم تا نرخ همگرایی این الگوریتم و تعداد ذرات موجود در نقطه‌ی کمینه سراسری را مشاهده و در ادامه با سایر الگوریتم‌ها مقایسه کنیم.



شکل ۲ - ۴۳: نمودار تغییرات جواب بهینه نسبت به تکرار چرخه الگوریتم امپریالیست رقابتی



شکل ۲ - ۴: جای‌گیری ذرات در حالت اولیه و نهایی در کانتور دو بعدی مورد بررسی

با توجه به شکل ۲ - ۴۳ و شکل ۲ - ۴۴ می‌توان به این نتیجه رسید که در میان الگوریتم‌های بررسی شده بهترین نرخ همگرایی را دارد و به هیچ عنوان روند همگرایی به دقت ۹۹٪ با کاهش سرعت همراه نبوده است، که نشان‌دهنده قدرت این الگوریتم در خروجی از کمینه‌های محلی را نشان می‌دهد. علاوه بر این جمعیت ذرات نزدیک به نقطه کمینه جهانی نیز نسبت به سایر الگوریتم‌ها بیشتر است با وجود آنکه تعداد تکرار چرخه مناسبی هم داشته است. پس می‌توان با اطلاعات کنونی ادعا کرد که در میان الگوریتم‌های مورد بررسی بهترین عملکرد را در یافتن کمینه جهانی تابع محک داشته است.

۲- ۳- ۶- جمع‌بندی

در این بخش قصد داریم تا با ساده سازی و میانگین‌گیری کردن داده‌های بدست‌آمده از پنج بخش قبل به یک نتیجه‌گیری قابل اعتماد در مورد قدرت عملکرد الگوریتم‌ها با استفاده از ورودی‌ها تعیین شده پردازیم. برای این کار میانگین عملکرد کلی هر الگوریتم در هر پنج مرحله را در کنار میانگین کمینه و بیشینه زمانی که داشته است مقایسه می‌کنیم و سپس همین فرآیند را بر روی الگوریتم‌ها با رویکرد بررسی و مقایسه هر الگوریتم با موقعیت ورودی دقیقاً یکسان (هر پنج الگوریتم از موقعیت پایین چپ شروع کرده باشند) انجام می‌دهیم.

جدول ۲ - ۲۱: میانگین بیشینه و کمینه الگوریتم‌ها

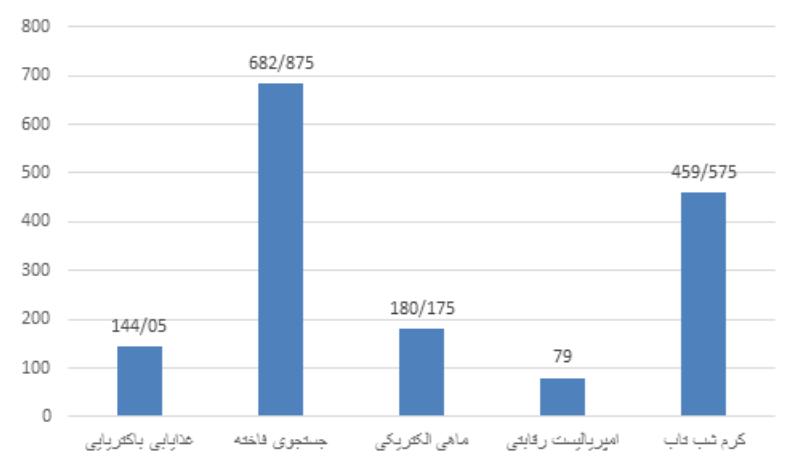
نام الگوریتم	اجرا	چرخه	میانگین کمینه زمان	میانگین بیشینه زمان	میانگین بیشینه تکرار	چرخه	میانگین بیشینه
غذایابی باکتریایی	۱/۵۲۸	۱۲	۶۳/۳۵	۵۴۲/۲	۶۳/۳۵	۰/۹۰۶۸	۱۶۵۵/۸
جستجوی فاخته	۰/۱۳	۱۸۸/۲	۰/۹۰۶۸	۱۶۵۵/۸	۰/۹۰۶۸	۱/۸۸۸	۳۸۸/۲
ماهی الکتریکی	۰/۳۵۴	۶۱/۸	۱/۸۸۸	۳۸۸/۲	۱/۸۸۸	۱/۴۴۲	۳۱۰/۴
امپریالیست رقابتی	۰/۰۴۲	۱۲/۲	۱/۴۴۲	۳۱۰/۴	۱/۴۴۲	۱/۳۹	۱۱۰۲/۶
کرم شبتاب	۰/۰۶۲	۴۷/۲	۱/۳۹	۱۱۰۲/۶	۱/۳۹		

با مقایسه داده‌های مربوط به کمینه و بیشینه متوجه می‌شویم که جستجوی فاخته همواره در تعداد تکرار چرخه نیاز به تعداد بیشتری دارد و امپریالیست رقابتی نیز به کمترین مقدار تکرار نیاز دارد. همچنین الگوریتم غذایابی باکتریایی به بیشترین زمان برای رسیدن به کمینه جهانی و جستجوی فاخته و امپریالیست رقابتی به ترتیب کمترین زمان را نیاز دارند.

جدول ۲ - ۲۲: میانگین نتایج اجراها به تفکیک الگوریتم‌ها

نام الگوریتم	تعداد تکرار چرخه	زمان اجرا	نتایج بهینه‌سازی	X مربوط به نتیجه نهایی	Y نتیجه نهایی مربوط به
غذایابی باکتریایی	۱۴۴/۰۵	۱۶/۸	-۹۵۹/۶۳۶	۵۱۲	۴۰۴/۲۳۱
جستجوی فاخته	۶۸۲/۸۷۵	۰/۴	-۹۵۹/۶۳۵	۵۱۲	۴۰۴/۲۳۰
ماهی الکتریکی	۱۸۰/۱۷۵	۰/۹۱	-۹۵۹/۶۳۷	۵۱۲	۴۰۴/۲۳۴
امپریالیست رقابتی	۷۹	۰/۳۱	-۹۵۹/۶۳۸	۵۱۲	۴۰۴/۲۴۹
کرم شبتاب	۴۵۹/۵۷۵	۰/۰۸	-۹۵۹/۶۳۸	۵۱۲	۴۰۴/۲۲۹

از آنجایی که مقادیر مربوط به موقعیت کاملا مشابه یکدیگر هستند و مقدار بهینه نهایی نیز با توجه به اینکه شرط دقت ۹۹٪ را برای خروج از حلقه شرط درون برنامه‌ها اعمال شده است گزینه‌های خوبی برای مقایسه این الگوریتم‌ها نیستند. برای مقایسه سایر پارامترهای الگوریتم‌ها نیز شکل ۲ - ۴۵ و شکل ۲ - ۴۶ نمودارهای میله‌ای مربوط به این مقادیر میانگین هستند. با توجه به این نمودارها می‌توان مقایسه روشن‌تری داشت.



شکل ۲ - ۴۵: مقایسه تعداد تکرار چرخه‌های الگوریتم‌ها



شکل ۲ - ۴۶: مقایسه زمان اجرا الگوریتم‌ها

با توجه به تصاویر بالا و جدول مربوط به داده‌های کمینه و بیشینه الگوریتم‌ها می‌توانیم به این نتیجه‌گیری برسیم که:

- الگوریتم امپریالیست رقابتی بهترین عملکرد را چه از نظر زمانی و چه از نظر تعداد تکرار دارد. از طرف دیگر الگوی کاهشی در تابع هزینه این الگوریتم نسبت به سایر الگوریتم‌ها بسیار منطقی‌تر و با کمترین درگیری با کمینه‌های محلی بوده است.
- الگوریتم ماهی الکتریکی در تعداد تکرار مناسبی می‌تواند به نتیجه دلخواه برسد. هر چند زمان مورد نیاز برای رسیدن به این نتیجه معقول در این الگوریتم نسبت به کندترین الگوریتم پس از آن، یعنی الگوریتم کرم شبتاب حوالی دو برابر می‌باشد.
- الگوریتم غذایابی باکتریایی تعداد تکرار بسیار مناسبی برای خروج از حلقه شرط خود دارد اما زمان مورد نیاز بسیار زیادی دارد. در تصویر (۴۵-۲) بطور کاملا مشهودی حدود هفده برابر کندترین الگوریتم پس از خود یعنی ماهی الکتریکی است.

- الگوریتم جستجوی فاخته در زمان مناسبی می‌تواند ما را به نتیجه دلخواه برساند هر چند تعداد تکرار مورد نیاز برای خروج از حلقه‌این الگوریتم با اختلاف بیشتر از رقیبان خود می‌باشد.
- الگوریتم کرم شب‌تاب نیز در کنار اینکه توانایی همگرایی بسیار ضعیفی دارد و عملاً در طول تکرارهای خود می‌توان گفت به نحوی شناسی توانسته است موقعیت کمینه سراسری را پیدا کند، و اگر تعداد کرم‌ها را کم کنیم یا فضای جستجوی بزرگتری درست کنیم نمی‌تواند به کمینه جهانی دست یابد. همچنین تعداد تکرار بسیار زیادی نیز نیاز دارد.

حال برای اطمینان بیشتر و نتیجه‌گیری نهایی می‌توانیم الگوریتم‌ها را از منظر زمان تکرار اجرای برنامه و تعداد تکرار چرخه با ورودی‌های یکسان مقایسه کنیم. به این منظور داده‌های مربوط به هر پنج اجرا در هر الگوریتم را با توجه به ورودی‌های آن با یکدیگر مقایسه می‌کنیم. به این معنی که داده‌های بدست‌آمده از پنج اجرایی که مربوط به ورودی‌های داده‌های قرار گرفته در چپ، چپ پایین، چپ بالا، راست، راست پایین، راست بالا، محور اصلی، محور فرعی را جداگانه میانگین می‌گیریم و سپس هر پنج الگوریتم را بر مبنای این داده‌ها مقایسه می‌کنیم. جدول ۲ - ۲۳ و جدول ۲ - ۲۴ و جدول ۲ - ۲۵ داده‌های مرتبط با توضیحات داده‌شده می‌باشد.

جدول ۲ - ۲۳: مقایسه تعداد تکرار چرخه الگوریتم‌ها با ورودی‌های یکسان

نام الگوریتم	بالاراست	بالا_چپ	محور فرعی	راست	محور اصلی	چپ	پایین راست	پایین چپ
غذایابی باکتریابی	۱۰۰/۶	۵۷/۲	۱۱۸/۶	۲۴۳/۸	۲۴۳/۶	۷۱/۸	۵۴/۴	۲۶۳/۴
جستجوی فاخته	۳۲۴/۴	۸۵۸/۶	۲۹۲/۲	۴۹۶/۴	۳۰۹	۱۴۲۰/۴	۶۲۹/۸	۱۱۳۲/۲
ماهی الکترونیکی	۱۴۵/۲	۱۵۹/۲	۲۸۹/۲	۱۱۸/۸	۲۰۵	۱۶۵/۸	۲۲۵	۱۳۳/۲
امپریالیست رقابتی	۱۸/۲	۲۲۵/۶	۱۷	۸۹	۱۱۷/۸	۳۸/۴	۳۹/۲	۸۹/۶
کرم شب‌تاب	۴۵۱/۴	۳۹۹	۳۱۰/۸	۴۳۲/۸	۶۹۰/۲	۵۳۲	۳۵۵	۵۴۰

جدول ۲ - ۲۴: مقایسه دقت الگوریتم‌ها با ورودی‌های یکسان

نام الگوریتم	بالاراست	بالا_چپ	محور فرعی	راست	محور اصلی	چپ	پایین راست	پایین چپ
غذایابی باکتریابی	۰/۰۰۵	۰/۰۰۴	۰/۰۰۶	۰/۰۰۴	۰/۰۰۴	۰/۰۰۶	۰/۰۰۴	۰/۰۰۴
جستجوی فاخته	۰/۰۰۴	۰/۰۰۶	۰/۰۰۶	۰/۰۰۶	۰/۰۰۳	۰/۰۰۷	۰/۰۰۶	۰/۰۰۴
ماهی	۰/۰۰۵	۰/۰۰۴	۰/۰۰۴	۰/۰۰۳	۰/۰۰۳	۰/۰۰۴	۰/۰۰۴	۰/۰۰۴

الگوریتم‌های فراتکاری در بهینه‌سازی

								الکتریکی
۰/۰۰۵	۰/۰۰۲	۰/۰۰۴	۰/۰۰۱	۰/۰۰۲	۰/۰۰۲	۰/۰۰۴	۰/۰۰۵	امپریالیست رقابتی
۰/۰۰۲	۰/۰۰۵	۰/۰۰۳	۰/۰۰۱	۰/۰۰۳	۰/۰۰۳	۰/۰۰۳	۰/۰۰۲	کرم شب تاب

جدول ۲ - ۲۵: مقایسه زمان اجرا الگوریتم‌ها با ورودی‌های یکسان

محور اصلی	راست	محور فرعی	بالا_چپ	بالا_راست	نام الگوریتم
۲۸/۸۷	۲۶/۶۳	۱۴/۴۴	۷۴۹	۱۲/۴۳	غذایابی باکتریابی
۰/۱۸	۰/۳۲	۰/۲۲	۰/۴۴	۰/۲۴	جستجوی فاخته
۱/۰۴	۰/۶۲	۱/۳۷	۰/۸۱	۰/۸۲	ماهی الکتریکی
۰/۴۰	۰/۳۰	۰/۰۵	۱/۰۸	۰/۰۸	امپریالیست رقابتی
۰/۸۲	۰/۵۴	۰/۳۸	۰/۵۰	۰/۳۷	کرم شب تاب

با محاسبه‌ی مقدار کمینه و بیشینه در هر کدام از دسته‌های ورودی می‌توانیم متوجه شویم که بطور عمده کدام یک از الگوریتم‌ها بهتر عمل کرده‌اند. در جدول ۲ - ۲۶ الگوریتم‌ها از حیث تعداد تکرار چرخه بیشینه و کمینه گیری شده‌اند و با نگاه به این جدول متوجه می‌شویم که الگوریتم امپریالیست تنها در داده‌های مربوط به بالا_چپ نتوانسته بهترین عملکرد را داشته باشد و در سایر ورودی‌ها با کمترین تکرار توانسته است به دقت ۹۹٪ برسد. و الگوریتم‌های کرم شب تاب و فاخته بدترین عملکرد را از نظر تعداد تکرار اجرا داشته‌اند.

جدول ۲ - ۲۶: مقایسه کمینه و بیشینه تعداد تکرار چرخه

۸۹/۶	۳۹/۲	۳۸/۴	۱۱۶/۸	۸۹	۱۷	۵۶/۲	۱۸/۲	مقدار کمینه
امپریالیست	امپریالیست	امپریالیست	امپریالیست	امپریالیست	امپریالیست	غذایابی باکتریابی	امپریالیست	نام کمینه
۱۱۳۲/۲	۶۲۹/۸	۱۴۲۰/۴	۶۹۰/۲	۴۹۶/۴	۳۱۰/۸	۸۵۸/۶	۴۵۱/۴	مقدار بیشینه
فاخته	فاخته	فاخته	شب تاب	فاخته	شب تاب	فاخته	کرم شب تاب	نام بیشینه

در جدول ۲ - ۲۷ الگوریتم‌ها از نظر دقت عملکرد مورد بررسی قرار گرفته‌اند، هر چند که تقاضوت دقت این الگوریتم‌ها بسیار جزئی است بدلیل شرط خروج از حلقه‌ای که در برنامه وجود دارد و نتایج این داده‌ها نمی‌تواند باعث جهت گیری مهمی شود ولی بازهم بررسی این مورد خالی از لطف نیست. با بررسی داده‌های جدول متوجه می‌شویم که الگوریتم امپریالیست و کرم شب تاب بیشترین دقت را میان الگوریتم‌ها داشته و جستجوی فاخته در نصف داده‌ها بیشترین خطرا را دارند.

جدول ۲ - ۲۷: مقایسه کمینه و بیشینه دقت عملکرد الگوریتم‌ها

۰/۰۰۲	۰/۰۰۲	۰/۰۰۳	۰/۰۰۱	۰/۰۰۲	۰/۰۰۲	۰/۰۰۳	۰/۰۰۲	مقدار کمینه
۰/۰۰۲	۰/۰۰۲	۰/۰۰۳	۰/۰۰۱	۰/۰۰۲	۰/۰۰۲	۰/۰۰۳	۰/۰۰۲	

الگوریتم‌های فراتکاری در بهینه‌سازی

نام کمینه	کرم شب تاب	امپریالیست	کرم شب تاب	امپریالیست	امپریالیست	امپریالیست	شب تاب	کرم شب تاب	امپریالیست	فاخته	فاخته	فاخته	غذایابی	فاخته	غذایابی	امپریالیست	ماهی	نام بیشینه
مقدار بیشینه	۰/۰۰۷	۰/۰۰۶	۰/۰۰۷	۰/۰۰۴	۰/۰۰۶	۰/۰۰۶	۰/۰۰۴	۰/۰۰۵	۰/۰۰۰									
نام بیشینه	الکتریکی	ماهی	امپریالیست	غذایابی	فاخته	امپریالیست	فاخته	امپریالیست	فاخته	غذایابی	غذایابی	غذایابی	غذایابی	غذایابی	غذایابی	فاخته	فاخته	امپریالیست

در جدول ۲ - ۲۸، الگوریتم‌ها از نظر زمان اجرا بررسی شده‌اند و واضح‌ا ضعف الگوریتم غذایابی باکتریایی را در زمان اجرا نشان می‌دهد که در تمام ورودی‌ها بیشترین زمان را داشته است و باز الگوریتم امپریالیست به خوبی در قسمت زمان اجرا هم درخشیده و این نشان از قدرت این الگوریتم نسبت به سایر الگوریتم‌ها می‌باشد.

جدول ۲ - ۲۸: مقایسه کمینه و بیشینه زمان اجرا الگوریتم‌ها

مقدار کمینه	۰/۰۸	۰/۴۴	۰/۰۵	۰/۳۰	۰/۱۸	۰/۱۱	۰/۱۰	۰/۳۵
نام کمینه	امپریالیست	امپریالیست	امپریالیست	امپریالیست	فاخته	امپریالیست	امپریالیست	امپریالیست
مقدار بیشینه	۱۲/۴۳	۶/۴۹	۱۴/۴۴	۲۶/۶۳	۲۸/۸۷	۸/۰۹	۶/۳۰	۳۱/۱۷
نام بیشینه	غذایابی	غذایابی	غذایابی	غذایابی	غذایابی	غذایابی	امپریالیست	امپریالیست

بطورکلی می‌توان از مقایسه‌های انجام شده در بخش جمع‌بندی، به این نتیجه رسید که الگوریتم امپریالیست در تمام زمینه‌ها توانسته عملکرد درخشنانی را نشان دهد، الگوریتم کرم شب تاب بدلیل عدم بسیار ضعیف عمل کرده است و الگوریتم غذایابی باکتریایی با وجود همگرایی مناسب و تعداد تکرار چرخه نسبتاً کم زمان زیادی برای پردازش نیاز دارد. سایر الگوریتم‌ها از جمله ماهی الکتریکی و فاخته نیز عملکرد متوسطی داشته‌اند زیرا از حیث همگرایی درگیری زیادی با کمینه‌های محلی اطراف کمینه جهانی داشته‌اند (از تعداد تکرار زیاد برای اتمام برنامه بعد از کاهش شدید اولیه در تابع هزینه) و هر کدام در یکی از زمینه‌های زمان اجرا یا تعداد تکرار چرخه از خود ضعف نشان داده‌اند.

فصل ۳:

استفاده از الگوریتم‌های فرآابتكاری در مسیریابی

ربات

۳ - ۱ مقدمه

در فصل سوم، پس از صحه‌گذاری الگوریتم‌ها توسط تابع محک قصد داریم تا الگوریتم‌ها را در زمینه یافتن بهترین مسیر برای رسیدن از نقطه‌ی مبدأ انتخابی به نقطه مقصد انتخابی بسنجدیم. به این منظور نیاز است تا محیط ثابتی بسازیم و در آن نقاطی ثابت را به عنوان مانع در نظر بگیریم. سپس باید تابع هزینه الگوریتم‌ها را از تابع شانه تخم مرغی به تابعی برای محاسبه کوتاه‌ترین مسیر ممکن برای عبور از موانع، بدون برخورد با این موانع تغییر دهیم. این کار نیازمند طراحی این تابع هزینه می‌باشد. همچنین تلاش شد در این فصل برای حذف فاکتور تاثیر نحوه برنامه‌نویسی در عملکرد الگوریتم‌ها از شیوه‌ی یکسانی برای نوشتن کدها در مطلب استفاده شود. شیوه‌ی نگارش جدید الهام گرفته از الگوریتم اپریالیست‌رقابتی است که در فصل گذشته عملکرد بسیار خوبی از خود نشان داده است. در ادامه به بررسی هر یک از برنامه‌ها و نحوه کلی نگارش این کدها می‌پردازیم و در نهایت نتایج حاصل از اجرای برنامه‌ها را بررسی و مقایسه می‌کنیم. باید توجه داشت که تئوری استفاده شده در الگوریتم‌ها کاملاً مشابه با فصل دوم می‌باشد و در نتیجه دیگر به تئوری این الگوریتم‌ها نمی‌پردازیم.

۳ - ۲ مدل‌سازی مسیریابی ربات

در مدل‌سازی مسیریابی ربات از آنجایی که با کوتاه‌ترین مسیری که به موانع نیز برخورد نکرده است به عنوان هزینه سر و کار داریم پس باید به بررسی مسیرها، که یک منحنی هستند پردازیم. از این رو با استفاده از تعداد نقاط مشخصی، تعدادی منحنی می‌سازیم که جمعیت ذرات تعیین شده برای الگوریتم‌ها می‌باشد. بهینه‌سازی را برروی نقاط این منحنی‌ها به جهت کاهش طول آن‌ها انجام می‌دهیم و از پارامترهای هر کدام از الگوریتم‌ها مشابه با فصل دوم برای ایجاد تعادل بین اکتشاف و بهره‌وری استفاده می‌کنیم. همچنین از آنجایی که کوتاه‌ترین مسیر مقدار مشخصی ندارد دیگر از حلقه‌ی while برای انجام بهینه‌سازی استفاده نشده است و بجای آن از حلقه for با تعداد تکرار یکسانی در تمام الگوریتم‌ها برای بهینه‌سازی استفاده شده است. در ادامه نحوه اجرای این فرآیند را توضیح می‌دهیم. باید توجه داشت که برخی از بخش‌های شیوه پیاده‌سازی الگوریتم‌ها یکسان هستند و فقط یکبار توضیح داده می‌شوند. سپس در هر بخش به این برنامه‌ها ارجاع می‌دهیم.

۳-۲-۱ توابع مشابه در پیاده‌سازی الگوریتم‌ها

در این بخش قسمت‌هایی از پیاده‌سازی برنامه که در تمام الگوریتم‌ها مشابه است به تفکیک، آورده شده است.

۳-۲-۱-۱ ایجاد فایل پارامترهای ثابت

در شیوه‌ی نگارش جدید برنامه داده‌ها در ابتدا در فایلی با فرمت و نام data.mat ذخیره می‌شود و در طول الگوریتم‌ها داده‌ها استفاده و بروزرسانی می‌شود. به این منظور از برنامه ۳ - ۱ استفاده شده است. همانطور که مشاهده می‌شود در برنامه ۳ - ۱ مقادیر مرتبط با موقعیت اولیه، موقعیت نهایی، تعداد نقاط کنترل حرکت مسیر و شعاع موانع و مرز فضای مورد بررسی نوشته شده و در استراکچر model ذخیره می‌شود. در انتها نیز این اطلاعات در فایلی بنام data در فolder مرتبط با برنامه ذخیره می‌شود. از آنجایی که نحوه ذخیره داده‌ها و مقادیر ذخیره شده در فایل data.mat در تمام الگوریتم‌ها مشابه است، در توضیح الگوریتم‌ها به برنامه ۳ - ۱ اشاره می‌شود و توضیح دوباره‌ای داده‌نمی‌شود.

برنامه ۳ - ۱: تولید و ذخیره داده‌های الگوریتم کرم شبتاب

```
clc
clear

% Source
xs=-400;ys=-400;

% Target (Destination)
xt=500;yt=200;

xobs = [-142.8150, -119.8318, -493.5311, 27.1833, -20.5996, -427.7995, -308.3201, -106.6312, -62.5280, 497.2782];
yobs = [-204.4902, -43.7162, -381.3250, -357.8582, 50.0318, -271.6420, -185.9180, 104.1066, -263.2235, -395.8582];
robs = [100, 29.7448, 26.9412, 27.7449, 19.0991, 19.5069, 30.3737, 10.0943, 18.6221, 45.0099];

n=5;

xmin=-512;
xmax= 512;

ymin=-512;
ymax= 512;

model.xs=xs;
model.ys=ys;
model.xt=xt;
model.yt=yt;
model.xobs=xobs;
```

```

model.yobs=yobs;
model.robs=robs;
model.n=n;
model.xmin=xmin;
model.xmax=xmax;
model.ymin=ymin;
modelymax=ymax;

xmin=model.xmin;
xmax=model.xmax;

ymin=model.ymin;
ymax=model.ymax;

LB=[xmin*ones(1,n) ymin*ones(1,n)];
UB=[xmax*ones(1,n) ymax*ones(1,n)];

nvar=2*n;

save data

```

۱- ۲- ۳- تابع هزینه

برنامه ۳ - ۲: تابع هزینه

```

function sol=fitness(sol,data)
global NFE
%% Calling Data
load data

if NFE==0
    xx = linspace(model.xs, model.xt, model.n+2);
    yy = linspace(model.ys, model.yt, model.n+2);
    sol.x = [xx(2:end-1) yy(2:end-1)];
end

NFE=NFE+1;
%% Calling Sol
A=sol.x; % A is the solutions

x=A(1:n);y=A(n+1:end); %spliting solutions in two part o Y and X

%%
XS=[xs x xt];
YS=[ys y yt];
k=numel(XS);
TS=linspace(0,1,k);

tt=linspace(0,1,100);
xx=spline(TS,XS,tt);
yy=spline(TS,YS,tt);

dx=diff(xx);
dy=diff(yy);

```

```

L=sum(sqrt(dx.^2+dy.^2));

nobs = numel(xobs); % Number of Obstacles
Violation = 0;
for k=1:nobs
    d=sqrt((xx-xobs(k)).^2+(yy-yobs(k)).^2);
    v=max(1-d/robs(k),0);
    Violation=Violation+mean(v);
end

%%
z=L;
%% Cal CH
CH=Violation;SCH=1000000000*sum(CH);

%% Cal OBJ

fit0=z;

sol.fit=fit0*(1+SCH);
sol.info.x=A;
sol.SCH=SCH;
sol.info.SCH=SCH;
sol.info.CH=CH;
sol.info.fit0=fit0;
sol.info.fit=sol.fit;
sol.info.TS=TS;
sol.info.XS=XS;
sol.info.YS=YS;
sol.info.tt=tt;
sol.info.xx=xx;
sol.info.yy=yy;
sol.info.dx=dx;
sol.info.dy=dy;
sol.info.L=L;

end

```

مهم‌ترین تابعی که نسبت به برنامه‌ی توضیح داده شده در فصل دوم تغییر کرده است، تابع هزینه می‌باشد که از تابع شانه تخم مرغی که در پیوست آ توضیح داده شده است به تابع نشان داده شده در برنامه ۳ - ۲ تبدیل شده است. از این رو توضیحات این بخش کامل‌تر از بخش‌های دیگر ارائه می‌شود. در ابتدای کد، متغیر جهانی‌ای با نام *NFE* تعریف شده است که شمارنده‌ای برای تعداد دفعات فراخوانی تابع *fitness* در حلقه اصلی الگوریتم می‌باشد، علت تعریف این متغیر داشتن اطلاعات درباره‌ی تعداد دفعاتی است که بهینه‌سازی صورت گرفته و علاوه بر آن ایجاد شرط در ابتدای برنامه‌ی *fitness* به‌جهت ایجاد جمعیت اولیه نقاط مورد نیاز برای ایجاد منحنی‌ها می‌باشد. همانطور که اشاره شد شرط *if* به‌جهت این در ابتدای برنامه ۳ - ۲ استفاده شده است که اگر اولین بار است که تابع در حلقه اصلی فراخوانی می‌شود میان‌یابی‌های بین نقطه شروع، پایان را به تعداد $n + 2$ (تعداد نقاط استفاده شده برای ساخت منحنی‌ها) انجام

دهد و ذخیره نماید تا نقاط اولیه برای ساخت منحنی تولید شود. همچنین با حذف نقطه شروع و پایان مقادیر میانیابی را در بخش x استراکچر sol می‌ریزد. باید توجه داشت که sol خروجی تابع M بشد که با توجه به حلقه اصلی همان متغیری (از نوع استراکچری) است که به عنوان جمعیت منحنی‌ها و ویژگی‌های مرتبط با آنها استفاده می‌شود. سپس متغیر پله زمانی (TS) ایجاد می‌شود که شامل تعداد نقاطی یکسان با تعداد نقاط x اما در فاصله‌ی بین 0 تا 1 می‌باشد. این میانیابی برای این است که بتوانیم منحنی x و y را بحسب این پله زمانی (TS) رسم کنیم. همچنین با ایجاد 100 نقطه با فاصله‌ی یکسان سعی می‌کنیم تا این منحنی تا جای ممکن دقیق باشد. سپس با محاسبه‌ی دیفرانسیل x و y بطور جداگانه که در واقع فاصله هر نقطه با نقطه قبلی خود می‌باشد مقدار dx و dy را محاسبه می‌کنیم. در انتهای با محاسبه فاصله‌ی کارتزینی که در معادله‌ی (۱-۳) نشان داده شده است، طول منحنی را محاسبه می‌نماییم و در متغیر L ذخیره می‌کنیم.

$$\sum_0^n \sqrt{dx^2 + dy^2} \quad 1-3$$

پس از محاسبه طول منحنی پارامتر مهم بعدی که در تابع هزینه تاثیرگذار است را بررسی می‌کنیم، یعنی بررسی برخورد نقاط منحنی با موانع، به این منظور نیز تعداد نقاطی که برای میانیابی استفاده شده است را تا جای ممکن زیاد کرده ایم تا بتوانیم با احتمال خیلی زیادی بیان کنیم که در صورت عدم برخورد این نقاط، منحنی نیز برخوردي نداشته است. روش بررسی برخورد یا عدم برخورد با موانع به این صورت است که فاصله تک تک نقاط منحنی را با نقاط مرکزی تک تک مowanع محاسبه کرده و اگر این فاصله کمتر از شعاع مانع باشد به این معنی است که برخوردي صورت گرفته و در این حالت میانگین میزان برخورد منحنی با تمام موانع (مقداری که در مانع فرو رفته است) در متغیر **violation** ریخته می‌شود و در نهایت با ضرب عدد 10^9 یک جریمه‌ی خیلی بزرگ در مقدار **violation** ضرب می‌شود و سپس در طول تابع (L) که در متغیر جدیدی بنام **fit0** ریخته شده است ضرب می‌شود، تا باعث شود منحنی‌ای که برخورد داشته به سرعت مسیر خود را اصلاح کند و از مانع دور شود. در انتهای تابع **fitness** بخش‌های **fit**, **infoSCH** از استراکچر را بروزرسانی می‌کنیم.

۳-۲-۱-۳ نحوه‌ی نمایش نتایج

بخش مشترک دیگری که در تمام برنامه‌ها مشترک می‌باشد، بخش **visualization** است. این قسمت از دو بخش تشکیل شده است. بخش اول در برنامه اصلی نوشته شده است و بخش دوم در تابعی بنام **RES** که فراخوانی می‌شود، علت تفکیک این قسمت به دو بخش تفکیک می‌شود، بخش به تصویرکشیدن داده‌های مرتبط با عملکرد الگوریتم مانند،

نرخ همگرایی، با بخش نمایش منحنی مسیر حرکت ربات می‌باشد. برنامه ۳ - ۳ بخش اول نمایش نتایج را نشان می‌دهد. که ابتدا در بخش **command window**، متلب بهترین مقدار تابع هزینه بدست آمده همراه با مدت زمان اجرای حلقه اصلی نشان داده می‌شود. سپس یک تصویر ایجاد می‌شود که نمودار روند حرکت تابع هزینه در تکرارهای حلقه اصلی تا زمان خروج از حلقه را نشان می‌دهد.

برنامه ۳ - ۳: ایجاد نمودار وضعیت تابع هزینه در هر تکرار

```
disp(' ')
disp([' BEST fitness = ' num2str(gpop.fit)]);
disp([' Time = ' num2str(elapsed_time)]);

figure(1)
plot(BEST(1:iter), 'r', 'LineWidth', 2)

xlabel(' iteration ')
ylabel(' fitness')
legend('BEST')
title('Name of each Algorithm')
```

پس از برنامه ۳ - ۳ تابع *RES* فراخوانی می‌شود که در برنامه ۳ - ۴ این تابع را بررسی می‌کنیم. در ابتدای تابع تعداد فراخوانی تابع هزینه را که در متغیر *NFE* ذخیره کرده بودیم نمایش می‌دهد و سپس تابعی که در خود دارد را فراخوانی می‌کند، این تابع به جهت نمایش فضای مورد جستجو همراه با موانع، نقطه شروع، نقطه پایان و مسیر تایید شده توسط الگوریتم می‌باشد.

برنامه ۳ - ۴: تابع *RES*

```
function RES(sol,data)
load data
global NFE

=====
BEST Solution
=====
disp(sol.info)
    disp([' Number Of Function Evaluations = ' num2str(NFE) ])

x=sol.info.x;
PlotSolution(sol.info,model);

end

function PlotSolution(sol,model)
xs=model.xs;
ys=model.ys;
xt=model.xt;
```

```

yt=model.yt;
xobs=model.xobs;
yobs=model.yobs;
robs=model.robs;

XS=sol.XS;
YS=sol.YS;
xx=sol.xx;
yy=sol.yy;
figure()
theta=linspace(0,2*pi,100);
for k=1:numel(xobs)
    fill(xobs(k)+robs(k)*cos(theta),yobs(k)+robs(k)*sin(theta),[ 0.5 0.7
0.8]);
    hold on;
end
plot(xx,yy,'k','LineWidth',2);
plot(XS,YS,'ro');
plot(xs,ys,'bs','MarkerSize',12,'MarkerFaceColor','y');
plot(xt,yt,'kp','MarkerSize',16,'MarkerFaceColor','g');
hold off;
grid on;
axis equal;

end

```

۳-۲-۲-۲- پیاده‌سازی بهینه‌سازی مسیریات در محیط متلب به تفکیک الگوریتم‌ها

در این بخش نحوه پیاده‌سازی کامل الگوریتم‌های بهینه‌سازی به منظور مسیریابی را به تفکیک الگوریتم‌ها و با جزئیات توضیح می‌دهیم.

۳-۲-۲-۱- الگوریتم غذایابی باکتریایی

جدول ۳ - ۱: جدول پارامترهای الگوریتم بهینه‌سازی غذایابی باکتریایی

پارامتر	مقدار
$Range$	-۵۱۲_۵۱۲
$population$	۲۰
N_c	۳۰
N_s	۲
N_{re}	۴

ϵ	N_{ed}
۳۰	$S_r(S / 2)$
۰.۵	P_{ed}
۰.۸	C
(-۴۰۰, ۴۰۰)	نقطه شروع
(۵۰۰, ۲۰۰)	نقطه هدف
۵	No. control points(n)
۳	No. obstacles
۱۰۰	No. iteration

پارامترهای تعیین شده در جدول (۱-۳) کاملاً مشابه با پارامترهای مربوط به بهینه‌سازی تابع محک توسط الگوریتم بهینه‌سازی غذایابی باکتریایی که در فصل دوم (بخش ۲-۱-۱-۲) بررسی کردیم تعریف شده‌است و فقط پارامترهای مربوط به نقطه شروع، پایان و تعداد موانع به جدول اضافه شده‌است. همچنین بدلیل استفاده از حلقه for تعداد تکرارهای مجاز برای بهینه‌سازی نیز به این پارامترها اضافه شده‌است که در واقع پارامتر ذاتی الگوریتم نمی‌باشد و مختص کاربرد فعلی است. در ادامه همانطور که در برنامه ۳ - ۱ توضیح داده شد داده‌های مورد نیاز را با اجرای برنامه در فolder الگوریتم غذایابی باکتریایی ذخیره می‌کنیم. سپس data.mat را در برنامه اصلی فراخوانی کرده و پارامترهای الگوریتم را نیز در بخش ابتدایی برنامه اصلی می‌نویسیم. این فرآیند در برنامه ۳ - ۵ نشان داده شده‌است.

برنامه ۳ - ۵: تعریف پارامترها و داده‌های الگوریتم غذایابی باکتریایی

```

data = load('data.mat'); % Load path planning data

nvar = data.nvar; % Number of variables
lb = data.LB; % Lower Bound
ub = data.UB; % Upper Bound

maxiter = 100; % Maximum Number of iterations

Np = 20; % Number of Bacteria
Ns = 2; % Swim Length
Nc = 30; % Number of Chemotaxis Steps
Nr = 4; % Number of Reproduction Steps
Ne = 4; % Number of Elimination-Dispersal Events
Ped = 0.5; % Elimination-Dispersal Probability
C = 0.8; % Step size

```

- مرحله بعد آماده‌سازی مقادیر اولیه جواب‌ها یا همان باکتری‌های اولیه برای شروع حلقه‌ی اصلی است. طبق برنامه ۳

۶ باید توجه داشت که تفاوت اصلی این قسمت با مرحله تعیین جمعیت اولیه باکتری‌ها که برای بررسی تابع محک در فصل دوم در فرم جمعیت اولیه است این است که دیگر به فرم ذرات نیستند بلکه هر باکتری در واقع دسته از نقاطی است که منحنی‌ای شکل می‌دهد و جمعیت اولیه بیست منحنی است. علاوه بر این داده در فرمت structure ذخیره شده‌است. این structure از چهار بخش تشکیل شده‌است، که در ابتدا به‌شکل خالی و با نام emp ایجاد شده و سپس با دستور repmat این ساختار به اندازه جمعیت ذرات ساخته شده و با نام جدید pop ایجاد می‌شود.

- *pop.x* بخش ذخیره‌ی موقعیت هر یک از ذرات سازنده بیست منحنی مورد نظر می‌باشد.
 - *pop.SCH* بخش ذخیره‌ی مقدار violation یا نزدیکی منحنی به موانع می‌باشد.
 - *pop.info* بخش ذخیره‌ی اطلاعات مرتبط با تابع هزینه می‌باشد و violation در هر تکرار در این بخش نیز ذخیره می‌شود.
 - *pop.fit* مقدار تابع هزینه هر منحنی را در خود ذخیره می‌کند، که تنها تفاوت آن‌ها در میزان violation می‌باشد.
- در حلقه `for i = 1:Np` در خط اول حلقه نقاطی به صورت رندوم با توزیع نرمال ایجاد می‌شود و سپس با ورود به تابع fitness پارامترهایی که باید در قسمت‌های مختلف استراکچر (i) pop ذخیره شود کامل شده و در (i) pop ریخته می‌شود.

برنامه ۳ - ۶: تعیین جمعیت اولیه باکتری‌ها

```
emp.x = [];
emp.SCH = [];
emp.info = [];
emp.fit = [];

pop = repmat(emp, Np, 1);

for i = 1:Np
    pop(i).x = unifrnd(lb, ub); % Initialize bacteria positions
    pop(i) = fitness(pop(i), data);
end
```

برنامه ۳ - ۷ شروع حلقه‌ی اصلی برنامه می‌باشد که همانند برنامه‌ی بهینه‌سازی غذایابی باکتریابی در فصل دوم از چهار حلقه تو در تو استفاده شده‌است که به ترتیب از حلقه جامع به حلقه کوچکتر عبارتند از:

حلقه حذف پراکندگی، حلقه تولیدمیل، حلقه کموتاکسی و حلقة شنا می‌باشد. در حلقة کموتاکسی باکتری شروع به حرکت به اندازه‌ی N_s در جهتی تصادفی به مقدار گام مشخص C می‌نماید و در هر حلقة نیز در انتهای طول N_s اقدام به شنا می‌کند مگر اینکه جواب بهتری نیابد و از این حلقة خارج شود و در نهایت پس از پایان حلقة کموتاکسی، مجموع جواب‌های بهینه در پارامتر J_{health} ریخته می‌شود، که فاکتور تعیین کننده برای مرتب سازی جمعیت باکتری‌ها براساس تابع هزینه‌ی آن‌ها می‌باشد. سپس در حلقة تولیدمیل نصفه‌ی ضعیف جمعیت براساس

J_{health} با نصفهای بالایی جایگزین می‌شوند به معنای دیگر جمعیت ضعیف حذف می‌شود و جمعیت قوی‌تر با تعداد دو برابر به جستجوی جواب بهینه بر می‌آیند (این قسمت از کد با فصل دوم تفاوت دارد زیرا در فصل دوم باکتری‌های جدید با موقعیت‌های تصادفی انتخاب می‌شوند). پس از خروج از حلقه‌ی توپلیدمث حلقه‌ی حذف و پراکندگی با توجه به احتمال حذف تعیین شده در برنامه به حذف تصادفی باکتری‌ها و جایگزین کردن آن‌ها با مقادیر تصادفی می‌پردازد و در نهایت بهترین جواب این چهار حلقه تو در تو در هر تکرار حلقه‌ی اصلی در gpop ریخته می‌شود و شرطی برای اینکه gpop جدید که ایجاد شده است ضعیفتر از جواب‌های گذشته که ممکن است حذف شده باشد نباشد ایجاد می‌شود. این مقدار در command window مطلب در هر تکرار گزارش می‌شود همراه با وضعیت اینکه بهترین جواب به موضع برخورد داشته است یا خیر.

برنامه ۳ - ۷: بهینه‌سازی غذایابی باکتری‌ای

```

for Loopiter = 1:maxiter

    for iter = 1:Ne
        for rep = 1:Nr
            Jchem = zeros(Np, Nc + 1); % Initialize Jchem matrix
            Jchem(:, 1) = [pop.fit]'; % First column of Jchem is the initial
fitness

            for chem = 1:Nc
                % Chemotaxis Loop %
                for i = 1:Np
                    del = (rand(1, nvar) - 0.5) * 2; % Random direction
                    pop(i).x = pop(i).x + (C / sqrt(sum(del.^2))) * del; %

Tumble
                    pop(i).x = min(max(pop(i).x, lb), ub);

                    pop(i) = fitness(pop(i), data);
                    m = 0;
                    while m < Ns
                        m = m + 1;
                        if pop(i).fit < gpop.fit
                            gpop = pop(i);
                            pop(i).x = pop(i).x + C * (del / sqrt(sum(del.^2)));
%
Swim
                            pop(i).x = min(max(pop(i).x, lb), ub);
                            pop(i) = fitness(pop(i), data);
                        else
                            break; % End of swim
                        end
                    end
                end
            end
            Jchem(:, chem + 1) = [pop.fit]'; % Store the fitness after each
chemotactic step
        end

        % Calculate Jhealth as the sum of all chemotactic costs for each
bacterium
        Jhealth = sum(Jchem, 2); %calculates the sum of the elements in each
row
    end
end

```

```

% Sort bacteria and chemotactic parameters
[~, I] = sort(Jhealth, 'ascend');
pop = [pop(I(1:Np/2)); pop(I(1:Np/2))];
gpop = pop(1);

end

% Elimination-Dispersal
for i = 1:Np
    if rand < Ped
        pop(i).x = unifrnd(lb, ub);
        pop(i) = fitness(pop(i), data);
    end
end

% Update Best Solution
[minpop, ind] = min([pop.fit]);

if minpop < gpop.fit
    % Ensuring that Elimination dispersal did not eliminate the best
solution
    gpop = pop(ind);
end

% In each iteration, store gpop in the array
gpopArray(Loopiter) = gpop; % Store the structure

BEST(Loopiter) = gpop.fit;
MEAN(Loopiter) = mean([pop.fit]);

disp(['Iteration ' num2str(Loopiter) ' Best= ' num2str(BEST(Loopiter))]);
end

```

باید توجه داشت کهتابع هزینه در بهینه‌سازی الگوریتم غذایابی باکتریابی نیز طبق برنامه ۳ - ۲ نوشته می‌شود. در نهایت نتایج بدست‌آمده پس از تعداد تکرار مجاز حلقه، مطابق با برنامه ۳ - ۳ به تصویر کشیده می‌شود.

۳-۲-۲-۲- الگوریتم کرم شبتاب

جدول ۳ - ۲: جدول پارامترهای الگوریتم بهینه‌سازی کرم شبتاب

پارامتر	مقدار
<i>Range</i>	-۵۱۲ _ ۵۱۲
<i>population</i>	۲۰
γ	۰,۰۱
β_0	۱
α	۰,۹
ضریب افت	۰,۹۵

(-٤٠٠، ٤٠٠)	نقطه شروع
(٥٠٠، ٢٠٠)	نقطه هدف
٥	No. control points(n)
٣	No. obstacles
١٠٠	No. iteration

پارامترهای تعیین شده در جدول ۳ - ۲ کاملا مشابه با پارامترهای مربوط به بهینه‌سازی تابع محک توسط الگوریتم بهینه‌سازی کرم شب‌تاب که در فصل دوم بررسی کردیم تعریف شده است و فقط پارامترهای مربوط به نقطه شروع، پایان و تعداد موانع به آن اضافه شده است. همچنین بدلیل استفاده از حلقه for تعداد تکرارهای مجاز برای بهینه‌سازی نیز به این پارامترها اضافه شده است که در واقع پارامتر ذاتی الگوریتم نمی‌باشد و مختص کاربرد فعلی است. در ادامه همانطور که در برنامه ۳ - ۱ توضیح داده شد داده‌های مورد نیاز را با اجرای برنامه در فولدر الگوریتم کرم شب‌تاب ذخیره می‌کنیم. سپس data.mat را در برنامه اصلی فراخوانی کرده و پارامترهای الگوریتم را نیز در بخش ابتدایی می‌نویسیم. این فرآیند در برنامه ۳ - ۸ نشان داده شده است.

برنامه ۳ - ۸: تعریف پارامترها و داده‌های الگوریتم کرم شب‌تاب

```
data=load('data.mat'); %load classes in the createdata file

nvar=data.nvar; %number of variables
lb=data.LB;      % Lower Bound
ub=data.UB;      % Upper Bound

maxiter=100;      % Maximum Number of iterations
npop=20;          % Number of Fireflies(Population)
L=1;              % Length of the path
gamma=0.01;        % Light Absorption Coefficient
beta0=1;          % Attraction Coefficient Base Value
alpha=0.9;         % Mutation Coefficient
alpha_RF=0.95;    %Radius Reduction Factor

data.lb=lb; data.ub=ub;
```

مرحله بعد آماده‌سازی مقادیر اولیه جواب‌ها یا همان کرم‌های اولیه برای شروع حلقه اصلی است. طبق برنامه ۳ - ۹ باید توجه داشت که تفاوت اصلی این قسمت با مرحله تعیین جمعیت اولیه کرم‌های شب‌تاب که برای بررسی تابع محک در فصل دوم استفاده شد در این است که هر کرم در واقع دسته‌ای از نقاط است که منحنی‌ای شکل می‌دهد و جمعیت اولیه بیست منحنی است (به‌فرم ذرات نیستند بلکه به‌فرم منحنی می‌باشند). علاوه بر این داده در فرمت structure ذخیره شده است. این structure از چهار بخش تشکیل شده است که در ابتدا به‌شکل خالی و با نام emp ایجاد شده و سپس با دستور repmat این ساختار به اندازه جمعیت ذرات ساخته شده و با نام جدید pop ایجاد می‌شود.

- pop.x بخش ذخیره‌ی موقعیت هر یک از ذرات سازنده‌ی بیست منحنی مورد نظر می‌باشد.
- pop.SCH بخش ذخیره‌ی مقدار violation یا نزدیکی منحنی به موانع می‌باشد.

- pop.info بخش ذخیره‌ی اطلاعات مرتبط باتابع هزینه می‌باشد و violation در هر تکرار در این بخش نیز ذخیره می‌شود.
- pop.fit مقدار تابع هزینه‌ی هر منحنی را در خود ذخیره می‌کند، که تنها تفاوت آن‌ها در میزان violation می‌باشد.
- در حلقه اول حلقه نقاطی به صورت رندوم با توزیع نرمال ایجاد می‌شود و سپس با ورود به تابع fitness رسایر پارامترهایی که باید در قسمت‌های مختلف استراکچر pop(i) ذخیره شود کامل شده و در pop(i) ریخته می‌شود.

برنامه ۳ - ۹: تعیین جمعیت اولیه کرم‌های شبتاب

```
emp.x=[];
emp.SCH=[];
emp.info=[];
emp.fit=[];
pop=repmat(emp,npop,1);

for i=1:npop
pop(i).x=unifrnd(lb,ub); %makes n number of x and n number of y as a vector
pop(i)=fitness(pop(i),data);
end
```

برنامه ۳ - ۱۰ شروع حلقه‌ی اصلی برنامه می‌باشد. در این بخش از برنامه از دو تابع fitness و MoveSol استفاده شده‌است که به ترتیب در برنامه ۳ - ۱۱ و برنامه ۳ - ۱۲ توضیح داده شده‌است. همانطور که در برنامه ۳ - ۱ مشخص است، کلیت برنامه مطابق بهینه‌سازی کرم شبتاب در فصل دوم می‌باشد. هر چند این حلقه دیگر از نوع while نیست و همچنین موقعیت‌های جدید ایجاد شده دیگر جایگزین موقعیت‌های قبلی نمی‌شوند بلکه در کنار آن‌ها ذخیره شده و در نهایت با استفاده از تابع sort تمام موقعیت‌های بررسی شده از نظر تابع هزینه مرتب شده و بهترین‌ها را به اندازه‌ی جمعیت منحنی تعیین شده انتخاب می‌کنند (سایر موقعیت‌ها حذف می‌شوند). این فرآیند به منظور جلوگیری از حذف جواب بهتر انجام شده‌است. علاوه بر این باید توجه داشت که برخلاف فصل دوم که هر ذره با ذرات دیگر از نظر مقدار هزینه مقایسه می‌شد در این بخش منحنی‌ها که هر کدام از تعداد مشخصی نقطه تشکیل شده‌اند، با یکدیگر مقایسه می‌شوند و بهترین آن‌ها انتخاب می‌شود (به عنوان مثال pop(1).x موقعیت نقاط تشکیل دهنده منحنی اول را در خود دارد). همچنین در انتهای برنامه ۳ - ۱۰ برخورد یا عدم برخورد منحنی برتر با موانع نیز اشاره می‌شود.

برنامه ۳ - ۱۰: بهینه‌سازی کرم شبتاب

```
for iter=1:maxiter
newpop=pop;
k=npop;
for i=1:npop
    for j=1:npop
```

```

if pop(j).fit<=pop(i).fit
k=k+1;
newpop(i)=pop(i);
newpop(i).x=MoveSol(pop(i).x,pop(j).x,alpha,beta0,gamma,lb,ub);
newpop(i)=fitness(newpop(i),data);
newpop(k)=newpop(i);

end
end
% Merge
[pop]=[pop;newpop;gpop];
% Sort and Select
[~, ind]=sort([pop.fit]);
pop=pop(ind);
pop=pop(1:npop);
% Select Best Sol
gpop=pop(1);
BEST(iter)=gpop.fit;
MEAN(iter)=mean([pop.fit]);

NO=' Feasible';
if any([gpop.SCH]>0)
    NO=' Infeasible';
end
disp(['iter ' num2str(iter) ' Best= ' num2str(BEST(iter)) NO]);
% Reduction Mutation Coefficient
alpha=alpha*alpha_RF;
end

```

برنامه ۱۱ - ۳: حرکت ذرات

```

function newx=MoveSol(x,xj,alpha,beta0,gamma,lb,ub)

newx=x;
rij=norm(x-xj);
beta=beta0*exp(-gamma*rij^2);
E=alpha*(unifrnd(-1,1,size(x)).*(ub-lb));

newx=x+...
    beta*(xj-x)+...
    E;
end

```

تابع حرکت ذرات نشان داده شده در برنامه ۳ - ۱۱ کاملاً مشابه با روش توضیح داده شده در فصل دوم نوشته شده است و تنها بطور جداگانه به یک تابع تبدیل شده و در کد اصلی فراخوانی می‌شود. هر چند که باید اشاره کرد که در این تابع، منحنی یک گام به سمت منحنی جذاب‌تر نزدیک می‌شود، بر خلاف فصل دوم که یک ذره به ذره‌ی جذاب‌تر نزدیک می‌شد. باید توجه داشت که تابع هزینه در بهینه‌سازی الگوریتم کرم شبتاب نیز طبق برنامه ۳ - ۲ نوشته می‌شود. در نهایت نتایج بدست آمده پس از صد تکرار نیز مطابق با برنامه ۳ - ۳ به تصویر کشیده می‌شود.

۳-۲-۳- الگوریتم امپریالیست رقابتی

جدول ۳ - ۳: جدول پارامترهای الگوریتم امپریالیست رقابتی

مقدار	پارامتر
-۵۱۲_۵۱۲	<i>Range</i>
۲۰	<i>population</i>
۴	<i>nimp</i>
۱	<i>alpha</i>
۲	<i>beta</i>
۰.۳	<i>p_{Revolution}</i>
۰.۵	<i>mu</i>
۰.۰۲	<i>zeta</i>
(-۴۰۰،۴۰۰)	نقطه شروع
(۵۰۰،۲۰۰)	نقطه هدف
۵	No. control points(n)
۳	No. obstacles
۱۰۰	No. iteration

پارامترهای تعیین شده در جدول ۳ - ۳ کاملاً مشابه با پارامترهای مربوط به بهینه‌سازی تابع محک توسط الگوریتم بهینه‌سازی امپریالیست رقابتی که در فصل دوم بررسی کردیم تعریف شده است و فقط پارامترهای مربوط به نقطه شروع، پایان و تعداد موانع به آن اضافه شده است هر چند مقادیر ثابت با توجه به تغییر مسئله کمی از نظر مقداری تغییر کرده‌اند. همچنین بدلیل استفاده از حلقه for تعداد تکرارهای مجاز برای بهینه‌سازی نیز به این پارامترها اضافه شده است که در واقع پارامتر ذاتی الگوریتم نمی‌باشد و مختص کاربرد فعلی است. در ادامه همانطور که در برنامه ۳ - ۱ توضیح داده شد، داده‌های مورد نیاز را با اجرای برنامه در فolder الگوریتم امپریالیست رقابتی ذخیره می‌کنیم. سپس data.mat را در برنامه اصلی فراخوانی کرده و پارامترهای الگوریتم را نیز در بخش ابتدایی می‌نویسیم. این فرآیند در برنامه ۳ - ۱۲ نشان داده شده است.

برنامه ۳ - ۱۲: تعریف پارامترها و داده‌های الگوریتم امپریالیست رقابتی

```
data=load('data.mat');
nvar=data.nvar;
lb=data.LB; % Lower Bound
ub=data.UB; % Upper Bound
ncountries=20;
nimp=4; % number of imperialists
maxiter=100;
alpha=1; % Consistent parameter for probability estimation
beta=2;
```

```
P_revolve=0.3;
zeta=0.02;
mu=0.5; % Revolution Rate
```

مرحله بعد آماده‌سازی مقادیر اولیه جواب‌ها یا همان کشورهای اولیه برای شروع حلقه‌ی اصلی است. طبق برنامه ۳

۱۳ باید توجه داشت که تفاوت اصلی این قسمت با مرحله تعیین جمعیت اولیه کشورها که برای بررسی تابع محک استفاده شد در این است که جمعیت اولیه دیگر به فرم ذرات نیستند، بلکه هر کشور در واقع دسته‌ای از نقاط است که منحنی‌ای شکل می‌دهد و جمعیت اولیه بیست منحنی است. همچنین از آنجایی که جمعیت کشورها تغییر کرده است به همان نسبت هم تعداد امپراطوری‌ها نیز کاهش یافته است. علاوه بر این داده با وجود انکه همانند فصل قبلی به فرمت ذخیره شده‌است. این structure از چهار بخش تشکیل شده‌است که در ابتدا به‌شکل خالی و با نام emp ایجاد شده‌است و سپس با دستور repmat این ساختار به اندازه جمعیت ذرات ساخته شده و با نام جدید colony ایجاد می‌شود.

- بخش ذخیره‌ی موقعیت هر یک از ذرات سازنده‌ی بیست منحنی مورد نظر می‌باشد.
- بخش ذخیره‌ی مقدار violation یا نزدیکی منحنی به موانع می‌باشد.
- بخش ذخیره‌ی اطلاعات مرتبط با تابع هزینه می‌باشد و violation در هر تکرار در این بخش نیز ذخیره می‌شود.

• مقدار تابع هزینه‌ی هر منحنی (کشور) را در خود ذخیره می‌کند.
باید توجه داشت که بعد از تقسیم‌بندی کشورها به امپراطور و مستعمره‌هایشان، به هر امپراطور دو بخش دیگر به فرم استراکچر اضافه می‌شود که در زیر توضیح می‌دهیم.

- این بخش مستعمرات امپراطوری در فرمت structure را نشان می‌دهد. مستعمرات چهار بخش اشاره شده در بالارا دارا می‌باشند.
- *imp.colony* این بخش مستعمرات امپراطوری کل امپراطوری، یعنی امپراطور و کلونی‌هایش ذخیره می‌کند که پارامتر اصلی در مقایسه‌ی امپراطوری‌ها می‌باشد. فرمول محاسبه‌ی این هزینه در فصل دوم، فرمول (۲۹-۲) اشاره شده‌است.

در برنامه ۳ - ۱۳، باید توجه داشت که این بخش از برنامه اصلی خارج شده‌است و تحت برنامه‌ی *for* نوشته شده‌است و در برنامه‌ی اصلی فراخوانی می‌شود. در حلقه‌ی مقابل *create_initial_population*، ابتدا برای هر کشور به تعداد طول بردار *Lb* و *ub* عدد رندوم در بازه‌ی فضای مورد بررسی تولید می‌کنیم و آنرا به تابع *fitness* تحويل می‌دهیم تا منحنی را ایجاد کرده و طول و شرایط برخورد به موانع را بررسی کند. سپس با مرتب کردن کلونی‌ها براساس مقدار هزینه‌ی آن‌ها امپراطورها را مشخص می‌کنیم و با تابع رولتویل (*RouletteWheel*) که در

برنامه ۳ - ۱۴ توضیح داده شده‌است، کلونی‌ها را بطور رندوم بین امپراطورها پخش می‌کنیم. باید توجه داشت که احتمال انتخاب شدن هر امپراطور در محاسبات رولتویل به مقدار نرمالایز شده‌ی تابع هزینه‌ی آن‌ها که در متغیر *P*

ذخیره می‌شود بستگی دارد. هرچه یک امپراطور وضع بهتری داشته باشد، احتمال داشتن کلونی بیشتری دارد. در انتها نیز با استفاده از تابع $cal_total_fitness$ مقدار هزینه‌ی کل امپراطوری‌ها، که به معنی هزینه‌ی هر امپراطور و کلونی‌هایش می‌باشد را محاسبه می‌کنیم. در برنامه ۳ - ۱۵ تابع $cal_total_fitness$ توضیح داده شده است.

برنامه ۳ - ۱۳: تعیین جمعیت اولیه کشورها

```

emp.x=[];
emp.info=[];
emp.fit=[];
emp.SCH=[];

colony=repmat(emp,ncountries,1);

for i=1:ncountries
colony(i).x=unifrnd(lb,ub);
colony(i)=fitness(colony(i),data);
end

[value,index]=sort([colony.fit]);

colony=colony(index);

imp=colony(1:nimp);

colony=colony(nimp+1:end);

ncolony=length(colony);

colony=colony(randperm(ncolony));

P=exp(-alpha*[imp.fit]/max([imp.fit]));
P=P/sum(P);
a=0;
b=0;
c=0;
d=0;
for j=1:ncolony

    k=RouletteWheel(P);

    if k == 1
        a=a+1;
        imp(k).colony(a)=colony(j);
    elseif k==2
        b=b+1;
        imp(k).colony(b)=colony(j);

    elseif k==3
        c=c+1;
        imp(k).colony(c)=colony(c);
    end
end

```

```

elseif k==4
d=d+1;
imp(k).colony(d)=colony(j);

end

imp=cal_total_fitness(imp,zeta);

```

در

برنامه ۳ - ۱۴ مقدار احتمال را که از فرمول (۲-۳) محاسبه می‌کنیم، به عنوان ورودی می‌دهیم. سپس این احتمال را نرمالایز کرده و مقدار تجمعی آن را با مقدار عدد رندوم مقایسه می‌کنیم. باید توجه داشت که هرچه احتمال یک امپراطور بیشتر باشد فاصله‌ی آن با امپراطور بعدی بیشتر می‌شود و بدلیل افزایش اندازه فاصله‌ی این دو مقدار احتمال انتخاب شدن بیشتری دارند. همچنین بدلیل مقایسه‌ی بین عدد تصادفی و مقدار احتمال تجمعی هر اجرای برنامه می‌تواند نتیجه‌ی متفاوتی داشته باشد.

$$P = e^{-\alpha \frac{imperial_fitness_i}{\max_imperial_fitness}} \quad 2-3$$

برنامه ۳ - ۱۴ RouletteWheel :

```

function k=RouletteWheel(P)

P=P./sum(P);
P=cumsum(P);

k=find(rand<=P,1,'first');
end

```

برنامه ۳ - ۱۵ براساس فرمول (۲۹-۲) که در فصل دوم توضیح داده شد نوشته شده‌است و هدف آن محاسبه‌ی مقدار قدرت هر امپراطوری برای رقابت با سایر امپراطوری‌ها می‌باشد. باید توجه داشت که هر چه پارامتر zeta بیشتر شود تاثیر کلونی‌ها در قدرت امپراطوری بیشتر می‌شود و بلعکس.

برنامه ۳ - ۱۵ cal_total_fitness :

```

function imp=cal_total_fitness(imp,zeta)

nimp=length(imp);

for i=1:nimp

    imp(i).totalfit=imp(i).fit+zeta*mean([imp(i).colony.fit]);

end

end

```

پس از ایجاد جمعیت اولیه کشورها، تفکیک امپراطور و تخصیص مستعمره به سراغ حلقه‌ی اصلی در برنامه ۳ - ۱۶ می‌رویم. باید توجه داشت که در فرآیند بهینه‌سازی امپریالیست رقابتی چند گام وجود دارد.

۱. فرآیند جذب و حرکت مستعمرات به سمت امپراطور خود (درتابع *assimilation* اتفاق می‌افتد).

۲. فرآیند انقلاب که برخی مستعمرات بطور تصادفی جابجا می‌شوند تا بتوانند موقعیت‌های بهتری پیدا کنند و جایگزین امپراطور شوند.

۳. مقایسه‌ی امپراطور و مستعمرات تا در صورتی که یکی از مستعمرات در شرایط بهتری نسبت به امپراطور قرار داشت با یکدیگر جایگزین شوند (درتابع *exchange* رخ می‌دهد).

۴. مقایسه‌ی امپراطوری‌ها با یکدیگر برای حذف تدریجی ضعیف‌ترین امپراطور.

در برنامه ۳ - ۱۶ مشخص است که این فرآیند تا زمانی که فقط یک امپراطور باقی بماند و یا صدمین تکرار هم اتفاق بیافتد ادامه پیدا می‌کند و در هر تکرار حلقه بهترین گزینه ممکن ذخیره شده و در command window نمایش داده می‌شود.

برنامه ۳ - ۱۶: بهینه‌سازی امپریالیست رقابتی

```
for iter=1:maxiter

    % Assimilation
    imp=assimilation(imp,nvar,beta,data); %ok

    % Revolution
    imp=Revolution(imp,nvar,data,P_revolve,mu);

    % Exchange
    imp=Exchange(imp);

    % Total fitness
    imp=cal_total_fitness(imp,zeta);

    % imperialistic competition
    imp=imperialistic_competition(imp, alpha);

    [value,index]=min([imp.fit]);

    if value<gimp.fit
        gimp=imp(index);
    end

    BEST(iter)=gimp.fit;

    nimp=length(imp);

    NO=' Feasible';
    if gimp.SCH>0
```

```

NO= ' Infeasible';
end

disp([' Iter = ' num2str(iter)...
      ' BEST = ' num2str(BEST(iter))...
      ' NIMP = ' num2str(nimp) NO]);

if nimp==1
    break
end
end
end

```

- در ادامه به توضیح تابع‌های اشاره شده در حلقه اصلی می‌پردازیم. ابتدا به سراغ تابع *assimilation* در برنامه ۳

می‌رویم.

assimilation : ۱۷ - ۳ برنامه

```

function imp=assimilation(imp,nvar,beta,data)

nimp=length(imp);
varsizesize(imp(1).x);
VarMin=data.LB;
VarMax=data.UB;
for i=1:nimp
    ncolony=length(imp(i).colony);

    for j=1:ncolony

        imp(i).colony(j).x = imp(i).colony(j).x ...
            + beta*rand(varsize).*(imp(i).x-imp(i).colony(j).x);

        imp(i).colony(j).x = max(imp(i).colony(j).x,VarMin);
        imp(i).colony(j).x = min(imp(i).colony(j).x,VarMax);

        imp(i).colony(j)=fitness(imp(i).colony(j),data);

    end
end
end

```

در تابع *assimilation* همانطور که توضیح داده شد تلاش برای حرکت مستعمرات به سمت امپراطور خود می‌باشد

که این گام حرکتی عبارت است از فاصله‌ی بین مستعمره و امپراطور در عددی تصادفی و پارامتر جذب (β) که در

فرمول (۳-۳) نشان داده شده است.

$$Position_{colony_j} = Position_{colony_j} + \beta \phi (Position_{imperial} - Position_{colony_j}) \quad ۳-۳$$

در برنامه ۳ - ۱۸ نحوه‌ی اجرای انقلاب به فرم کد به تصویر کشیده شده است. بطورکلی با استفاده از چند پارامتر از

جمله: μ که برای محدود کردن تعداد مستعمرات انتخابی برای انقلاب می‌باشد، σ که برای محدود کردن بازه حرکت و جهش مستعمره انتخابی است و $P_{revolve}$ که برای تعیین احتمال انتخاب شدن مستعمرات می‌باشد و با یک عدد رندوم مقایسه می‌شود. در انتهای نیز نقاط جدید بدست آمده برای مستعمراتی که تحت انقلاب قرار گرفته‌اند به تابع $fitness$ داده می‌شود تا منحنی و مقدار هزینه‌ی توابع محاسبه و مقدار جدید قدرت امپراطوری‌ها تعیین شود.

برنامه ۱۸ - ۳ Revolution :

```
function imp = Revolution(imp, nx, data, P_revolve, mu)
%mutate the whole curve
VarMin = data.LB;
VarMax = data.UB;
varsizes = size(imp(1).x); % Assuming imp(1).x is a vector

nimp = length(imp);
sigma = 0.4 * (VarMax - VarMin);

% Precompute the number of mutations per empire and random permutations
nmus = zeros(1, nimp);
rands = cell(1, nimp);

for i = 1:nimp
    ncolony = length(imp(i).colony);
    nmus(i) = ceil(mu * ncolony);
    rands{i} = randperm(ncolony);
end

% Apply mutations
for k = 1:nimp
    ncolony = length(imp(k).colony);
    for i = 1:nmus(k)
        idx = rands{k}(i);
        colony = imp(k).colony(idx);

        % Ensure NewPos has the same size as colony.x
        NewPos = colony.x + sigma .* randn(varsizes);

        % Check mutation probability
        if rand <= P_revolve
            % Update the position
            colony.x = NewPos;

            % Clip values to be within bounds
            colony.x = max(colony.x, VarMin);
            colony.x = min(colony.x, VarMax);

            % Update fitness
            imp(k).colony(idx) = fitness(colony, data);
        end
    end
end
end
```

پس از تابع $revolution$ از تابع $exchange$ استفاده می‌شود تا در صورتی که مستعمره‌ای در موقعیت بهتری قرار

گفته است به عنوان امپراطور جدید تعیین شود و امپراطور قبلی در مقام مستعمره قرار گیرد. نحوه پیاده‌سازی این تابع در برنامه ۳ - ۱۹ نشان داده شده است. باید توجه داشت که پس از فراخوانی این تابع در حلقه اصلی، دوباره مقدار قدرت کل امپراطوری محاسبه می‌شود.

برنامه ۳ - ۱۹ : *exchange*

```
function imp=Exchange(imp)
%competition inside each colony
nimp=length(imp);

for i=1:nimp
    [value,index]=min([imp(i).colony.fit]);

    if value<imp(i).fit

        bestcolony=imp(i).colony(index);

        imp(i).colony(index).x=imp(i).x;
        imp(i).colony(index).fit=imp(i).fit;
        imp(i).colony(index).info=imp(i).info;

        imp(i).x=bestcolony.x;
        imp(i).fit=bestcolony.fit;
        imp(i).info=bestcolony.info;
    end
end
end
```

در انتها به سراغ رقابت بین امپراطوری‌ها برای تعیین بهترین و بدترین امپراطوری می‌رویم، که در برنامه ۳ - ۲۰ نشان داده شده است. بدترین امپراطوری ضعیفترین مستعمره خود را از دست می‌دهد و این مستعمره بطور تصادفی با استفاده از چرخ رولت به یکی از امپراطوری‌های دیگر سپرده می‌شود. در صورتی که مستعمره‌ای نداشته باشد خود امپراطور به عنوان مستعمره به امپراطور دیگر سپرده می‌شود و امپراطوری آن بطورکلی حذف می‌شود.

برنامه ۳ - ۲۰ : *imperialistic_competition*

```
function imp = imperialistic_competition(imp, alpha)

nimp = length(imp);

if nimp >= 2
    [~, index1] = max([imp.totalfit]);
    wimp = imp(index1);

    [~, index2] = max([wimp.colony.fit]);
    wcolony = wimp.colony(index2);

    imp(index1).colony = imp(index1).colony([1:index2-1 index2+1:end]);

    TotalCost = [imp.totalfit];
    P = exp(-alpha * TotalCost / max(TotalCost));
    P(index1) = 0;
    P = P / sum(P);
    k = RouletteWheel(P);
```

```

n = length(imp(k).colony);
imp(k).colony(n+1).x = wcolon.y;
imp(k).colony(n+1).fit = wcolon.fit;
imp(k).colony(n+1).info = wcolon.info;
n = length(imp(index1).colony);
if n == 0
    imp = imp([1:index1-1 index1+1:end]);
    TotalCost = [imp.totalfit];
    P = exp(-alpha * TotalCost / max(TotalCost));
    P = P / sum(P);
    k = RouletteWheel(P);
    n = length(imp(k).colony);
    imp(k).colony(n+1).x = wimp.x;
    imp(k).colony(n+1).fit = wimp.fit;
    imp(k).colony(n+1).info = wimp.info;
end
end

```

باید توجه داشت که تابع هزینه در بهینه‌سازی الگوریتم امپریالیست رقابتی نیز طبق برنامه ۳ - ۲ نوشته می‌شود. در نهایت نتایج بدست‌آمده پس از صد تکرار یا رسیدن به یک امپراتور مطابق با برنامه ۳ - ۳ به تصویر کشیده می‌شود.

۳-۲-۴- الگوریتم جستجوی فاخته

جدول ۳ - ۴: جدول پارامترهای الگوریتم جستجوی فاخته

پارامتر	مقدار
Range	-۵۱۲_۵۱۲
population	۲۰
alpha	۰.۰۱
Pa	۰.۶
نقطه شروع	(-۴۰۰,۴۰۰)
نقطه هدف	(۵۰۰,۲۰۰)
No. control points(n)	۵
No. obstacles	۳

پارامترهای تعیین شده در جدول ۳ - ۴ کاملاً مشابه با پارامترهای مربوط به بهینه‌سازی تابع محک توسط الگوریتم بهینه‌سازی جستجوی فاخته که در فصل دوم بررسی کردیم تعریف شده است و فقط پارامترهای مربوط به نقطه شروع، پایان و تعداد موانع به آن اضافه شده است هر چند مقادیر ثابت با توجه به تغییر مسئله کمی از نظر مقداری تغییر کرده‌اند. همچنین بدلیل استفاده از حلقه for تعداد تکرارهای مجاز برای بهینه‌سازی نیز به این پارامترها اضافه شده است که در واقع پارامتر ذاتی الگوریتم نمی‌باشد و مختص کاربرد فعلی است. در ادامه همانطور که در برنامه ۳ - ۱ توضیح داده شد داده‌های مورد نیاز را با اجرای برنامه در فolder الگوریتم جستجوی فاخته ذخیره می‌کنیم. سپس data.mat را در برنامه

اصلی فراخوانی کرده و پارامترهای الگوریتم را نیز در بخش ابتدایی می‌نویسیم. این فرآیند در برنامه ۳ - ۲۱ نشان داده شده است.

برنامه ۳ - ۲۱: تعریف پارامترها و داده‌های الگوریتم جستجوی فاخته

```
data = load('data.mat');
nvar = data.nvar;
lb = data.LB; % Lower Bound
ub = data.UB; % Upper Bound

%% Parameters setting
npop = 20;      % Population size
maxiter = 100;   % Number of generations
Alpha = 0.01;    % Step Size
pa = 0.6;       % Discovery rate of alien eggs/solutions

data.lb = lb;
data.ub = ub;
```

مرحله بعد آماده‌سازی مقادیر اولیه جواب‌ها یا همان لانه‌های اولیه برای شروع حلقه‌ی اصلی است. طبق برنامه ۳ - ۲۲ باید توجه داشت که تفاوت اصلی این قسمت با مرحله تعیین جمعیت اولیه لانه‌ها که برای بررسیتابع محک استفاده شد در این است که جمعیت اولیه دیگر به‌فرم ذرات نیستند بلکه هر لانه در واقع دسته‌ای از نقاط است که منحنی‌ای شکل می‌دهد و جمعیت اولیه بیست منحنی است، نه بیست نقطه. علاوه بر این داده به فرمت structure ذخیره شده است. این structure از چهار بخش تشکیل شده است که در ابتدا به‌شكل خالی و با نام emp ایجاد شده است و سپس با دستور repmat این ساختار به اندازه جمعیت ذرات ساخته شده و با نام جدید pop ایجاد می‌شود.

- pop.x بخش ذخیره‌ی موقعیت هر یک از ذرات سازنده بیست منحنی مورد نظر می‌باشد.
 - pop.SCH بخش ذخیره‌ی مقدار violation یا نزدیکی منحنی به موانع می‌باشد.
 - pop.info بخش ذخیره‌ی اطلاعات مرتبط با تابع هزینه می‌باشد و violation در هر تکرار در این بخش نیز ذخیره می‌شود.
 - pop.fit مقدار تابع هزینه‌ی هر منحنی را در خود ذخیره می‌کند، که تنها تفاوت آنها در میزان violation می‌باشد
- باید توجه داشت که این بخش تعیین جمعیت اولیه لانه‌ها از برنامه اصلی خارج شده است و تحت تابع create_initial_population نوشته شده است و در برنامه ۳ - ۲۲ نشان داده شده است و در برنامه اصلی فراخوانی می‌شود. در این تابع جمعیت لانه‌ها، حدود فضا و ویژگی‌های فضای جستجو که در فایل data وجود دارد به عنوان ورودی داده می‌شود. ابتدا استارکچر طبق توضیحات بالاتر ساخته می‌شود و سپس در حلقه‌ای به تعداد جمعیت لانه‌ها، نقاطی تصادفی ایجاد شده و سپس این نقاط به تابع fitness که در برنامه ۳ - ۲ توضیح داده شده، داده می‌شود برای ایجاد منحنی و تعیین مقدار تابع هزینه.

برنامه ۳ - ۲۲: تعیین جمعیت اولیه لانه‌ها

```
% Create initial population
```

```

function pop = create_initial_population(npop, lb, ub, data)
    emp.x = [];
    emp.fit = [];
    emp.info = [];
    emp.SCH = [];
    pop = repmat(emp, npop, 1);

    for i = 1:npop
        pop(i).x = unifrnd(lb, ub);
        pop(i) = fitness(pop(i), data);
    end
end

```

پس از ایجاد جمعیت اولیه لانه‌ها و محاسبه مقدار هزینه هر لانه و مرتب سازی جمعیت براساس مقدار هزینه، به سراغ حلقه‌ی اصلی الگوریتم می‌رویم. بطورکلی مشابه با فصل گذشته الگوریتم جستجوی فاخته از سه تابع برای بهینه‌سازی استفاده می‌کند. گام‌هایی که در این حلقه وجود دارد عبارتند از:

۱. ایجاد جواب‌های جدید برای افزایش اکتشاف الگوریتم و خروج از کمینه‌های محلی با کمک قاعده پراوز لوی (در

تابع *(get_cuckoos)*.

۲. مقایسه بین جواب‌های جدید و قدیم برای یافتن جواب‌های بهتر و بهینه‌سازی (در تابع *(get_best_nest)*

۳. ایجاد جهش تصادفی در برخی از لانه‌ها (در تابع *(empty_nests)*)

۴. استفاده دوباره از مرحله ۲

۵. نمایش بهترین جواب در هر تکرار حلقه و اینکه آیا بهترین جواب به مانع برخورد کرده است یا خیر.

مراحل توضیح داده شده در برنامه ۳ - ۲۳ در قالب کد به نمایش در آمدند.

برنامه ۳ - ۲۳: بهینه‌سازی جستجوی فاخته

```

for iter = 1:maxiter
    % Get new solutions using Levy flights
    newpop = get_cuckoos(pop, gpop, npop, Alpha, data);

    % Update the population and the global best
    [fmin, gpop, pop] = get_best_nest(pop, newpop, data);

    % Discovery and randomization
    newpop = empty_nests(pop, npop, pa, data);

    % Update the population and the global best
    [fmin, gpop, pop] = get_best_nest(pop, newpop, data);

    % Store best and mean fitness values
    BEST(iter) = gpop.fit;
    MEAN(iter) = mean([pop.fit]);

    NO = 'Feasible';
    if gpop.SCH > 0
        NO = 'Infeasible';
    end
    disp(['Iter = ' num2str(iter) ' BEST = ' num2str(BEST(iter)) NO ])

```

end

در ادامه به توضیح تابع‌های اشاره شده در حلقه اصلی می‌پردازیم. ابتدا به سراغ تابع `get_cuckoos` در برنامه ۳ - ۲۴ می‌رویم. همانطور که اشاره شد هدف این تابع ایجاد جواب‌های جدید در متغیری به نام `newx` می‌باشد، به این منظور در ابتدا پارامترهایی که پرواز لوی نیاز دارند را محاسبه می‌کنیم و سپس در حلقه‌ای به تعداد جمعیتی که داریم به ازای هر لانه از پرواز لوی برای حرکت لانه‌ها استفاده می‌کنیم و موقعیت جدید آن‌ها را به تابع `fitness` می‌دهیم تا مقدار تابع هزینه جدید را محاسبه نماید.

برنامه ۳ - ۲۴ : `get_cuckoos`

```
% Get cuckoos by Levy flights
function newpop = get_cuckoos(pop, gpop, npop, Alpha, data)
    beta = 3/2;
    sigma = (gamma(1 + beta) * sin(pi * beta / 2) ...
        / (gamma((1 + beta) / 2) * beta * 2^((beta - 1) / 2))) ^ (1 / beta);
    newpop = pop;
    for i = 1:npop
        x = pop(i).x;
        % Use the levy flight mechanism for generating new solutions
        newx = levy_flight(x, gpop.x, beta, sigma, Alpha);
        newpop(i).x = newx;
        newpop(i) = fitness(newpop(i), data);
    end
end

% Levy flight function
function newx = levy_flight(x, best_x, beta, sigma, Alpha)
    u = randn(size(x)) * sigma;
    v = randn(size(x));
    S = u ./ abs(v) .^ (1 / beta); % Levy flight step
    r = randn(size(x));
    newx = x + (Alpha * S .* (x - best_x) .* r); % Update solution
end
```

خروجی تابع `get_cuckoos` به عنوان یکی از ورودی‌ها در کنار موقعیت قبلی لانه‌ها وارد تابع `get_best_nest` که در برنامه ۳ - ۲۵ نشان داده شده است، می‌شود، تا میان موقعیت‌های جدید هر لانه با موقعیت قبلی آن‌ها مقایسه‌ای داشته باشد و هر موقعیتی که تابع هزینه بهتری (کمتر) داشته باشد را انتخاب و به عنوان جمعیت در نظر بگیرد و در انتهای بهترین جواب میان لانه‌ها را مشخص و در متغیر `gpop` قرار می‌دهد.

برنامه ۳ - ۲۵ : `get_best_nest`

```
% Find the current best nest
function [fmin, gpop, pop] = get_best_nest(pop, newpop, data)
    for i = 1:length(pop)
        if newpop(i).fit < pop(i).fit
            pop(i) = newpop(i);
        end
    end
    [~, index] = min([pop.fit]);
    gpop = pop(index);
```

```
fmin = gpop.fit;
end
```

در مرحله بعد به سراغ ایجاد جهش در لانه‌ها برای افزایش اکتشاف و جلوگیری از گیر کردن الگوریتم در کمینه‌های محلی بوسیله‌ی تابع *empty_nests* می‌رویم. بطورکل در این تابع دو لانه بطور تصادفی انتخاب می‌شوند و فاصله این دو لانه‌ی تصادفی در مقداری تصادفی بین صفر تا یک ضرب شده و در نهایت این مقدار در پارامتر *pa* که احتمال انجام جهش می‌باشد، ضرب می‌شود. پارامتر *pa* تنها می‌تواند مقدار صفر یا یک داشته باشد، در صورتی که صفر باشد در واقع لانه جهشی نمی‌کند و اگر یک باشد به مقدار محاسبه شده جهش ایجاد می‌شود. باید توجه داشت که هرچه این پارامتر کوچک‌تر باشد احتمال ایجاد جهش در جمعیت لانه‌ها بیشتر می‌شود (تعداد بیشتری از لانه‌ها جهش می‌کنند). نحوه‌ی اجرای این فرآیند در قالب کد در برنامه (۲۶-۳) نشان داده شده است.

برنامه ۲۶ - ۳

```
% Replace some nests by constructing new solutions/nests
```

```
function newpop = empty_nests(pop, npop, pa, data)
    newpop = pop;
    for i = 1:npop
        j1 = randi([1 npop]);
        j2 = randi([1 npop]);
        x = pop(i).x;

        % Discovery and randomization
        S = rand(size(x)) .* (pop(j1).x - pop(j2).x);
        Pij = rand(size(x));
        Pij(Pij > pa) = 1;
        Pij(Pij < pa) = 0;

        newx = x + (Pij .* S);
        newpop(i).x = newx;
        newpop(i) = fitness(newpop(i), data);
    end
end
```

در مرحله نهایی دوباره جواب‌های جدید با جواب‌های قدیمی توسط تابع *get_best_nests* مقایسه شده و جواب‌های بهتر انتخاب می‌شوند. باید توجه داشت که تابع هزینه در بهینه‌سازی الگوریتم جستجوی فاخته نیز طبق برنامه ۳ - ۲ نوشته می‌شود. در نهایت نتایج بدست‌آمده پس از صد تکرار مطابق با برنامه ۳ - ۳ به تصویر کشیده می‌شود.

۳-۲-۵- الگوریتم ماهی الکتریکی

جدول ۳ - ۵: جدول پارامترهای الگوریتم ماهی الکترونیکی

مقدار	پارامتر
-۵۱۲_۵۱۲	<i>Range</i>
۲۰	<i>population</i>
۰.۹۹۹۹۹	<i>x</i>
۷	<i>active_pop</i>
e^{-5}	<i>thres</i>
(-۴۰۰, ۴۰۰)	نقطه شروع
(۲۰۰, ۵۰۰)	نقطه هدف
۵	No. control points(n)
۳	No. obstacles
۱۰۰	No. iteration

پارامترهای تعیین شده در جدول ۳ - ۵ کاملا مشابه با پارامترهای مربوط به بهینه‌سازی تابع محک توسط الگوریتم بهینه‌سازی ماهی الکترونیکی که در فصل دوم بررسی کردیم تعریف شده است هر چند مقادیر ثابت با توجه به تغییر مسئله کمی از نظر مقداری تغییر کرده‌اند. به علاوه بر این پارامترهای مربوط به نقطه شروع، پیان و تعداد موانع به آن اضافه شده است و بدلیل استفاده از حلقه for تعداد تکرارهای مجاز برای بهینه‌سازی نیز به این پارامترها اضافه شده است که در واقع پارامتر ذاتی الگوریتم نمی‌باشد و مختص کاربرد فعلی است. در ادامه همانطور که در برنامه ۳ - ۱ توضیح داده شد داده‌های مورد نیاز را با اجرای برنامه در فولدر الگوریتم ماهی الکترونیکی ذخیره می‌کنیم. سپس data.mat را در برنامه اصلی فراخوانی کرده و پارامترهای الگوریتم را نیز در بخش ابتدایی می‌نویسیم. این فرآیند در برنامه ۳ - ۲۷ نشان داده شده است.

برنامه ۳ - ۲۷: تعریف پارامترها و داده‌های الگوریتم ماهی الکترونیکی

```
data=load('data.mat');
nvar=data.nvar;
lb=data.LB; % Lower Bound
ub=data.UB; % Upper Bound
lowerBound=data.xmin;
upperBound=data.xmax;
maxiter=100;
populationSize = 20; % Population size, (M)
% Control parameters of EFO
x = 0.99999; % Balances the magnitudes of
current frequency and old amplitude values
activeIndSize =7; % Subpopulation size, (K)
thres = 1e-5; % A small constant value
```

- مرحله بعد آماده‌سازی مقادیر اولیه جواب‌ها یا همان ماهی‌های اولیه برای شروع حلقه اصلی است. طبق برنامه ۳

۲۸ باید توجه داشت که تفاوت اصلی این قسمت با مرحله تعیین جمعیت اولیه ماهی‌ها که برای بررسی تابع محک استفاده شد در این است که جمعیت اولیه دیگر به‌فرم ذرات نیستند بلکه هر ماهی در واقع دسته از نقاطی است که منحنی‌ای شکل می‌دهد و جمعیت اولیه بیست منحنی است. علاوه بر این داده به فرمت structure ذخیره شده است که زیر مجموعه‌های بیشتری نسبت به مدل‌سازی انجام شده در فصل دوم دارد که در ابتدا به‌شکل خالی و با نام emp ایجاد شده است و سپس با دستور repmat این ساختار به اندازه جمعیت ذرات ساخته شده و با نام جدید population ایجاد می‌شود. توضیح این زیر بخش‌های جدید بشرح زیر می‌باشد.

- بخش ذخیره‌ی موقعیت هر یک از ذرات سازنده بیست منحنی مورد نظر می‌باشد.
- بخش ذخیره‌ی مقدار violation یا نزدیکی منحنی به موانع می‌باشد.
- بخش ذخیره‌ی اطلاعات مرتبط با تابع هزینه می‌باشد و violation در هر تکرار در این بخش نیز ذخیره می‌شود.
- violation مقدار تابع هزینه‌ی هر منحنی را در خود ذخیره می‌کند، که تنها تفاوت آن‌ها در میزان population.fit می‌باشد.

و ساختار قبلی structure مربوط به جمعیت نیز به فرمت زیر تعریف شده‌اند:

- population.isActive
- population.amplitude
- population.frequency

باید توجه داشت که این بخش تعیین جمعیت اولیه ماهی‌ها از برنامه اصلی فراخوانی می‌شود. این کد در برنامه ۳ - ۲۸ نشان داده شده است. پس از ایجاد نقاط اولیه به صورت تصادفی این نقاط را وارد تابع fitness که در برنامه ۳ - ۲ توضیح داده شده می‌شود و پس از آن تمام ماهی‌ها را بطور پیشفرض به عنوان جمعیت در جستجوی فعال در نظر می‌گیریم. سپس ضعیفترین ماهی و قوی‌ترین ماهی را در متغیر جدای ذخیره می‌کنیم به‌منظور محاسبه فرکانس و دامنه در مرحله بعد مقدار انحراف معیار تمام داده‌های تابع هزینه را محاسبه کرده و با پارامتر thres مقایسه می‌کنیم، اگر از این مقدار کمتر باشد در این صورت مقدار فرکانس و دامنه تابع را بطور تصادفی وارد می‌کنیم و اگر مقدار بیشتری داشته باشد آنگاه برای محاسبه فرکانس و دامنه ماهی از [فرمول ۳۴-۲](#) استفاده می‌کنیم.

برنامه ۳ - ۲۸: تعیین جمعیت اولیه ماهی‌ها

```
emp.x=[];
emp.info=[];
emp.fit=[];
emp.SCH=[];
emp.isActive=[];
emp.amplitude=[];
emp.frequency=;
```

```

population=repmat(emp,populationSize,1);

%% initializing

for p = 1: populationSize

    population(p).x = unifrnd(lb,ub);
    population(p) = fitness(population(p),data);
    population(p).isActive = true;

end

populationInitial=population; %initial swarm

% The best individual information is stored
[bestCost, index] = min( [population(:).fit] );
bestIndividual = population(index);

% The worst individual information is also stored for frequency calculation
[worstCost, index] = max( [population(:).fit] );
worstIndividual = population(index).x;

globalBest = bestCost; %minimum cost
globalBestSol = bestIndividual; %the solution for minimum cost

% Frequency (f) and amplitude (A) values are also initialized
for p = 1: populationSize

    if std([population(:).fit]) < thres

        population(p).frequency = rand;

    else

        population(p).frequency = (worstCost - population(p).fit) / (worstCost - bestCost);

    end

    population(p).amplitude = population(p).frequency;

end

```

پس از ایجاد جمعیت اولیه ماهی‌ها و محاسبه مقدار هزینه، فرکانس و دامنه‌ی هر ماهی و ذخیره‌ی بهترین جواب به سراغ

حلقه‌ی اصلی الگوریتم می‌رویم. بطورکلی مشابه با فصل گذشته الگوریتم ماهی الکترونیکی از سه تابع برای بهینه‌سازی

استفاده می‌کند. گام‌هایی که در این حلقه وجود دارد عبارتند از:

۱. تخصیص ماهی انتخابی از جمعیت در متغیری بنام newindividual و در نظر گرفتن باقی ماهی‌ها، به عنوان

همسایه.

۲. بررسی اینکه ماهی انتخابی از دسته‌ی ماهی‌های active یا passive است تا بروی آن جستجوی فعال یا جستجوی غیرفعال انجام شود.
 ۳. ایجاد جهش در برخی از موارد.
 ۴. مقایسه‌ی بهترین جواب حاصل از یک تکرار چرخه با بهترین جواب بدست‌آمده و در صورت بهتر بودن جواب جایگزین کردن جواب جدید بدست‌آمده با بهترین جواب قبلی.
 ۵. خروج از حلقه و بروزرسانی مقدار فرکانس و دامنه.
- مراحل توضیح داده شده در برنامه ۳ - ۲۹ در قالب کد به نمایش در آمده‌اند و در ادامه به تفصیل توضیح داده شده است.

برنامه ۳ - ۲۹: بهینه‌سازی ماهی الکتریکی

```

for iter=1:maxiter

for p = 1: populationSize
    newIndividual = population(p);
    neighborIndividual = population;
    neighborIndividual(p) = [];

    if (newIndividual.frequency > rand)

        population(p).isActive = true;
        newIndividual.x = activeSearch(newIndividual, neighborIndividual,
lowerBound, upperBound);
    else

        population(p).isActive = false;
        activePopulation = neighborIndividual(
[neighborIndividual(:).isActive] );
        if ~isempty(activePopulation) %defining passive ones

            newIndividual.x = passiveSearch(newIndividual, activePopulation,
activeIndSize); %  Performs search through passive electrolocation

        end

        if rand < rand %  Mutates one or more paramete in a stochastic manner
to keep the population diverse
            I = randi(nvar);
            newIndividual.x(I) = lowerBound + (upperBound - lowerBound) *
rand; %random new individual
        end
    end
    newIndividual.x = boundaryCheck(newIndividual.x, lowerBound,
upperBound); %  Solution set of candidate individual, (x_cand)
    newIndividual = fitness(newIndividual, data); %  Fitness value of
candidate individual
    if newIndividual.fit < population(p).fit %  Accepts if better source
found
        population(p) = newIndividual;
    end
end

```

```

end
% The best individual information is stored
[bestCost, index] = min( [population(:).fit] );
bestIndividual = population(index).x;
gpopulation = population(index);

% The worst individual information is also stored for frequency
calculation
[worstCost, index] = max( [population(:).fit] );
worstIndividual = population(index).x;

% Frequency (f) and amplitude (A) values are updated
for p = 1: populationSize

    if std([population(:).fit]) < thres

        population(p).frequency = rand;

    else

        population(p).frequency = (worstCost - population(p).fit) /
(worstCost - bestCost);

    end
    population(p).amplitude = population(p).amplitude * (x) +
population(p).frequency * (1 - x);

end

if bestCost < globalBest % The best cost and x found so far are stored

    globalBest = gpopulation.fit;
    globalBestSol = gpopulation.x;
    gpopulation = gpopulation;

end
% bests=[bests , globalBest];
BEST(iter)=gpopulation.fit;

NO=' Feasible';
if gpopulation.SCH>0
    NO=' Infeasible';
end

disp([' Iter = ' num2str(iter)...
      ' BEST = ' num2str(BEST(iter))...
      ' feasibility' NO]);

% End of the iteration
end

```

حال به بررسی برنامه‌های استفاده شده در حلقه اصلی می‌رویم. اولین تابع مورد بررسی تابع *activeSearch* می‌باشد.

ورودی‌های این تابع یکی از ماهی‌ها همراه با همسایه‌های آن که شامل تمام جمعیت بجز این ماهی می‌باشد می‌شود. از آنجایی که در جستجوی فعال تنها باید یکی از پارامترها تغییر کنند پس بطور تصادفی یکی از موقعیت‌هایی که منحنی (

ماهی) را شکل می‌دهند را انتخاب کرده و ایندکس آن را در متغیر parameter می‌ریزیم. سپس بوسیلهٔ تابع getDistance مقدار فاصلهٔ ماهی انتخاب شده را از همسایه‌های آن بررسی می‌کنیم و یکی از همسایه‌هایی که در محدودهٔ جستجوی فعال می‌باشد را انتخاب می‌کنیم. باید توجه داشت که اگر هیچ همسایه‌ای در محدوده نباشد حرکت ماهی انتخاب شده به صورت تصادفی می‌باشد و در صورتی که همسایه وجود داشته باشد، اختلاف فاصلهٔ ماهی تعیین شده از ماهی همسایه را محاسبه و در ضربی ضرب می‌کنیم و با موقعیت ماهی انتخابی جمع می‌کنیم. سپس داده را به عنوان خروجی به حلقه اصلی می‌دهیم. نحوه انجام این فرآیند در برنامه ۳-۳۰ به تصویر کشیده شده است.

برنامه ۳-۳۰ activeSearch

```

function newSolution = activeSearch(newIndividual, neighborIndividual,
lowerBound, upperBound)
% Active Electrolocation Phase

activeRange = (upperBound - lowerBound) * newIndividual.amplitude; %
Calculates active search range
parameter = randi(size(newIndividual.x, 2)); % Determines jth parameter to
modify
dist = getDistance(newIndividual.x, [neighborIndividual(:).x]); % Obtains
cartesian distance between ith individual and its neighbors ( i.e., N \ {i} )
index = find(dist < activeRange); % Determines if at least a neighbor exists
in the active search range
if isempty(index) % If no neighbor exists in the search range (see Eq. 7)
    newIndividual.x(parameter) = newIndividual.x(parameter) + (2 * rand - 1) *
activeRange;
else % If at least one neighbor exists in the search range (see Eq. 6)
    selectedNeighbor = index(randi(size(index, 1))); % Randomly selects one
neighbor individual (k) among those inside active range area
    newIndividual.x(parameter) = newIndividual.x(parameter) + ...
    (neighborIndividual(selectedNeighbor).x(parameter) - ...
    newIndividual.x(parameter)) * (2 * rand - 1);
end
newSolution = newIndividual.x;
end

function dist = getDistance(newIndividualSol, solutionSet)
% Distance Calculator
parameterSize = size(newIndividualSol, 2);
popSize = size(solutionSet, 2) / parameterSize;
solutionSet = reshape(solutionSet, parameterSize, popSize)'; %sorting the
solutionset for dimensions(be andaze har set joda mikone)
dist = sqrt(sum((solutionSet - repmat(newIndividualSol, popSize, 1)).^2, 2));
end

```

در مرحله بعدی با استفاده از شرط، بررسی می‌کنیم که اگر active population خالی نباشد، تابع passiveSearch اجرا شود. ورودی‌های این تابع عبارتند از: ماهی مورد بررسی، جمعیت ماهی‌هایی که در حالت active قرار دارند که در متغیر active population قرار دارد و تعداد ماهی‌های فعالی که می‌توانند به جستجوی غیرفعال ماهی مورد بررسی جهت دهند. در ابتدا بررسی می‌کنیم که activeindszie از active population کمتر نباشد و اگر کمتر بود مقدار تعداد ماهی‌های فعال

مجاز (activepopulation) را به عنوان مقدار activeindsizr قرار می‌دهیم. سپس از آنجایی که می‌دانیم مقدار دامنه‌ی هر ماهی با فاصله آن نسبت عکس دارد و تاثیرگذاری اش را کمتر می‌کند مقدار دامنه‌ی amplitude را بر فاصله ماهی همسایه مورد نظر از ماهی انتخاب شده تقسیم می‌کنیم و مقدار بدست‌آمده را به حالت یک ارایه تجمیعی در می‌آوریم که طبیعتاً بدلیل اینکه این فرآیند در واقع بدست‌آوردن احتمال انتخاب شدن هر کدام از جمعیت ماهی‌های فعال می‌باشد، هر کدام که پتانسیل بیشتری داشته باشد و قدرت بیشتری داشته باشد احتمال انتخاب شدن بیشتری نیز دارد. سپس در حلقه‌ای به اندازه تعداد ماهی فعال مجاز، از میان تمام ماهی‌های در حالت فعال تعداد مشخص شده را با استفاده از قاعده رولتویل انتخاب می‌کنیم. سپس موقعیت ماهی‌های انتخاب شده را در متغیری بنام solutionset ذخیره کرده و مقدار مبنایی به‌شکل ایجاد میانگین وزنی بر حسب دامنه‌ی ماهی‌های active انتخاب شده ایجاد می‌کنیم و براساس این موقعیت مبنایی ایجاد شده ماهی‌ها به‌سمت آن حرکت می‌کنند و در نهایت تنها تعدادی تصادفی از ماهی‌هایی که حرکت کرده‌اند انتخاب می‌شوند و باقی ماهی‌ها حرکتشان در واقع تاییدنامی شود و در موقعیت قبلی خود می‌مانند. جمعیت جدید که برخی از ماهی‌ها حرکت کرده‌اند به حلقه اصلی باز می‌گردد. تمام مراحل گفته شده در برنامه ۳ - ۳۱ قابل مشاهده است.

برنامه ۳ - ۳۱
passivesearch

```
function newSolution = passiveSearch(newIndividual, activePopulation,
activeIndSize)
% Passive Electrolocation Phase

e = 1e-6; % A small constant used in distance calculation
D = size(newIndividual.x, 2); % Parameter size of the problem
if size(activePopulation, 1) < activeIndSize % If the number of active
individuals is less than that of predetermined neighbor, (K)
    activeIndSize = size(activePopulation, 1); % Set K as the number of
active individuals
end

dist = getDistance(newIndividual.x, [activePopulation(:).x]); % Obtains
cartesian distance between ith individual and those in active modes
p = cumsum([activePopulation(:).amplitude] ./ (dist' + e)); % The
probability values of active mode individuals (see Eq. 8)
selectedNeighbor = zeros(1, activeIndSize);
for i = 1: activeIndSize % K individuals are selected through fitness
proportionate (or roulette wheel) selection (see Eq. 8)
    I = find (rand*p(end) < p);
    selectedNeighbor(i) = I(1);
end
neighbor = activePopulation(selectedNeighbor);
solutionSet = reshape([neighbor.x], D, activeIndSize)';
xReference = sum(repmat([neighbor.amplitude]', 1, D) .* solutionSet) ./
sum(repmat([neighbor.amplitude]', 1, D)); % A reference point is created
(see Eq. 9)
newSolution = newIndividual.x + (xReference - newIndividual.x) .* (2 * rand(1,
D) - 1); % Generates new location (see Eq. 10)
I = find(rand(1, D) > newIndividual.frequency); % Determines which modified
parameters will be accepted
newIndividual.x(I) = newSolution(I);
newSolution = newIndividual.x;
```

end

پس از اجرای تابع `passivesearch` در حلقه اصلی، بطور تصادفی یکی از نقاط ماهی انتخاب شده (*newindividual*) بطور تصادفی انتخاب و جهش پیدا می‌کند تا مقدار اکتساف نیز افزایش پیدا کند. در ادامه ماهی به تابع هزینه داده می‌شود و اینکه در فضای مورد بررسی واقع شده است چک می‌شود و در انتهای اگر مقدار هزینه مطلوب‌تری نسبت به موقعیت قبلی خود داشته باشد جایگزین می‌شود و در غیر اینصورت موقعیت ماهی در طول حلقه انجام شده بطور کل حرکت نمی‌کند و موقعیت قبلی خود را حفظ می‌کند. باید توجه داشت که جهش ماهی در مرحله اخر نیز بطور حتمی نمی‌باشد و از یک شرط تصادفی استفاده می‌کند. در انتهای نیز مقدار دامنه و فرکانس به شیوه قبلی بروزرسانی می‌شود و اگر جواب بدست‌آمده از بهترین جواب چرخه قبلی بهتر باشد به عنوان بهترین جواب ذخیره می‌شود. در مرحله نهایی در نهایت نتایج بدست‌آمده پس از صد تکرار مطابق با برنامه ۳ - ۳ به تصویر کشیده می‌شود.

۳-۳ نتایج

پس از آشنایی با تئوری و نحوه نوشتن برنامه‌های الگوریتم‌ها در محیط متلب، حال به مقایسه نحوه عملکرد این پنج الگوریتم می‌پردازیم. به این منظور ابتدا باید مناسب‌ترین تعداد نقاط برای تشکیل هر منحنی از الگوریتم‌ها را که در بخش قبل این فصل به عنوان متغیر n تعریف شده بود را با روش سعی و خطا بدست آوریم. با امتحان مقادیر ۱۰۰، ۵۰، ۴۰، ۳۰، ۲۰، ۱۰، ۵ و ۳ مشاهده گشت که سه نقطه به عنوان نقاط تشکیل دهنده منحنی‌ها بهترین پاسخ را در کمترین زمان ممکن به ما می‌دهد. در نتیجه نتایج بدست‌آمده برای مقایسه با مقدار ثابت $n = 3$ بدست‌آمده است. برای مقایسه الگوریتم‌ها هر الگوریتم پنج بار با تعداد تکرارهای مختلف بیست و صد تکرار اجرا شده‌اند و نتایج نهایی مقایسه شده است. در پیوست ژ داده‌های بدست‌آمده از هر اجرای الگوریتم‌ها به تفکیک آمده است و به کمک این داده‌ها تلاش می‌شود تا مقایسه‌ای اصولی میان این الگوریتم‌ها صورت گیرد.

۳-۱ تحلیل نتایج با صد تکرار

با استفاده از داده‌های موجود در جداول بخش اول پیوست ژ می‌توانیم مقدار میانگین و درصد انحراف معیار داده‌ها را برای هر الگوریتم با صد تکرار بدست‌آورده و با یکدیگر مقایسه کنیم. با استفاده از میانگین حدود طول و زمان اجرای هر الگوریتم می‌توانیم عملکرد این الگوریتم‌ها را با یکدیگر مقایسه نمائیم و از طریق بررسی درصد انحراف معیار حدود بازه تغییرات داده‌ها در هر اجرا را برای هر الگوریتم متوجه می‌شویم. در نتیجه با بررسی این دو نوع داده می‌توانیم

استفاده از الگوریتم‌های فراتکاری در مسیریابی ربات

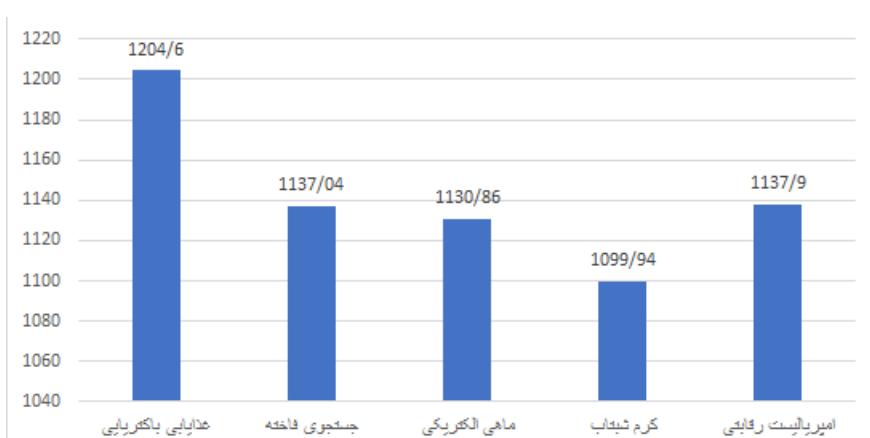
بهترین الگوریتم را از جهات گوناگون بیابیم.

جدول ۳ - ۶: نتایج اجرا الگوریتم‌ها با ۱۰۰ تکرار

نام الگوریتم	درصد انحراف معیار زمان	درصد انحراف معیار طول	انحراف معیار طول مسیر	انحراف معیار زمان اجرا (ثانیه)	میانگین طول مسیر	میانگین زمان اجرا (ثانیه)
غذایابی پاکتربایی	۱۳.۵	۱۴.۶	۱۷۵.۶	۲۰۹.۸	۱۲۰۴.۶	۱۵۵۶.۴
جستجوی فاخته	۱.۵	۲.۰	۲۲.۲	۰.۱	۱۱۳۷.۰۴	۶.۵
ماهی‌الکترویکی	۲.۲	۳.۳	۳۷.۴	۰.۱	۱۱۳۰.۸۶	۳.۷
کرم شبتاب	۱۴.۳	۰.۱	۰.۷	۶.۰	۱۰۹۹.۹۴	۴۱.۹
امپریالیست رقابتی	۱۳.۷	۳.۷	۴۲.۰	۰.۵	۱۱۳۷.۹	۳.۸

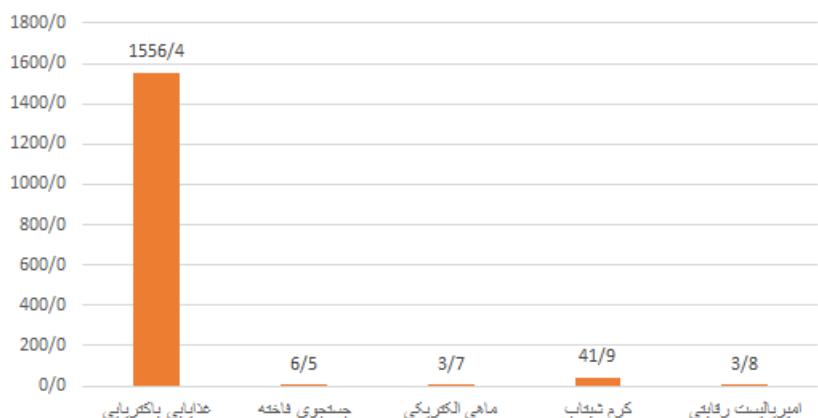
از آنجایی که تفاوت داده‌ها در نمودار واضح‌تر از فرم عددی می‌باشد، در شکل‌های زیر داده‌های جدول ۳ -

۶ به صورت نمودارهای میله نمایش داده‌ایم تا بتوانیم مقایسه واضح‌تری داشته باشیم.

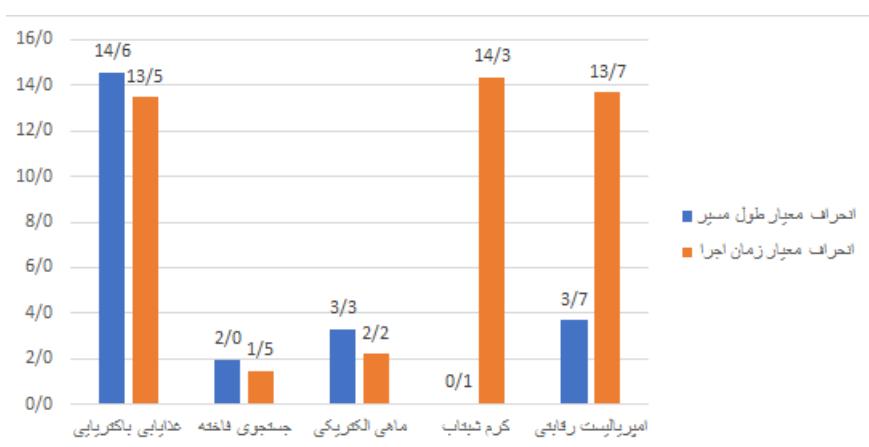


شكل ۳ - ۱: نمودار میله‌ای میانگین طول مسیر الگوریتم‌ها با ۱۰۰ تکرار

استفاده از الگوریتم‌های فراتکاری در مسیریابی ربات



شکل ۳ - ۲: نمودار میله‌ای میانگین زمان اجرا الگوریتم‌ها با ۱۰۰ تکرار



شکل ۳ - ۳: نمودار میله‌ای درصد انحراف معيار زمان و طول مسیر الگوریتم‌ها با ۱۰۰ تکرار

با نمودارهای مرتبط با اجرای الگوریتم‌ها با ۱۰۰ تکرار بهترین الگوریتم‌ها از هر منظر به شکل زیر مرتب می‌شوند:

- از حیث طول مسیر الگوریتم کرم شبتاب توانسته است کوتاه‌ترین مسیر را میان الگوریتم‌ها پیدا کند. طول مسیر یافته شده توسط الگوریتم کرم شبتاب حدود سه درصد از مسیرهای یافته شده توسط الگوریتم‌های امپریالیست رقابتی، جستجوی فاخته و ماهی الکترونیکی کوتاه‌تر بوده و حدود ده درصد کوتاه‌تر از مسیر یافته شود توسط الگوریتم غذایابی باکتریالیست می‌باشد. هرچند، اختلاف سه درصدی در مسیر رسیدن به هدف با توجه به طول مسیر مقدار خیلی زیادی نمی‌باشد و می‌توان عنوان کرد که الگوریتم‌های امپریالیست رقابتی، جستجوی فاخته و ماهی الکترونیکی نیز به خوبی عمل کرده‌اند.
- از منظر زمان اجرا برگزینه امپریالیست رقابتی و ماهی الکترونیکی سریع‌ترین زمان اجرا را میان الگوریتم‌ها ثبت کرده‌اند. زمان اجرا این دو الگوریتم حدود هفتادوپنج درصد سریع‌تر از جستجوی فاخته بوده‌اند و با دو الگوریتم دیگر تفاوت بسیار زیادی در زمان اجرا داشته‌اند، بطوریکه مدت زمان اجرا الگوریتم کرم شبتاب حدود یازده برابر و الگوریتم غذایابی باکتریالیست چهارصد و بیست برابر بیشتر از این دو الگوریتم بوده‌است.

- از منظر درصد انحراف معیار طول مسیر، الگوریتم کرم شبتاب بطور فاحشی بهتر از سایر الگوریتم‌ها عمل کرده است و عملاً انحراف معیاری نداشته است. درصد انحراف معیار طول مسیر الگوریتم‌های جستجوی فاخته، ماهی الکتریکی، امپریالیست رقابتی، غذایابی باکتریایی به ترتیب دو درصد، سه و نیم درصد و چهارده و نیم درصد بیشتر از درصد انحراف معیار الگوریتم کرم شبتاب می‌باشد.
 - از حیث درصد انحراف معیار زمان اجرا جستجوی فاخته بهترین عملکرد را داشته است. مقدار درصد انحراف معیار الگوریتم‌های ماهی الکتریکی، غذایابی باکتریایی، امپریالیست رقابتی و کرم شبتاب به ترتیب حدود نیم درصد، دوازده درصد و سیزده درصد بیشتر از این الگوریتم می‌باشد.
- با وجود آنکه اختلاف مسیر ایجاد شده توسط الگوریتم کرم شبتاب با اکثر الگوریتم‌های دیگر جزیی بوده است، با در کنار هم گذاشتن داده طول مسیر و درصد انحراف معیار طول مسیر می‌توان عنوان کرد که الگوریتم کرم شبتاب در یافتن کوتاهترین مسیر با کمترین بازه تغییرات در اجرای گوناگون بهترین عملکرد را داشته و در موقعیت‌هایی که نیاز به دقت و اطمینان بالا در یافتن مسیرهای کوتاه مشابه وجود دارد، بهترین عملکرد را دارد. همچنین با کنار هم گذاشتن داده‌های سریع‌ترین زمان اجرا و مقدار انحراف معیار می‌توان عنوان کرد الگوریتم ماهی الکتریکی در کوتاهترین زمان با کمترین تغییرات از حیث زمان اجرا بهترین عملکرد را داشته و در موقعیت‌هایی که زمان نقش تاثیرگذاری دارد مانند، استفاده از ربات در حالت Realtime بهتر از سایر الگوریتم‌ها می‌تواند عمل کند. الگوریتم ماهی الکتریکی بدلیل قرارگرفتن در رتبه‌ی دوم در تمام زمینه‌های مورد بررسی و اختلاف جزیی با رتبه اول میان الگوریتم‌ها از حیث طول مسیر و زمان اجرا نیز می‌تواند گزینه‌ی مطلوب‌تری برای موقعیت‌هایی باشد که علاوه بر دقت نسبی به زمان کوتاه برای پردازش نیاز است. از طرف دیگر الگوریتم غذایابی باکتریایی برخلاف یافتن کوتاهترین مسیر بدلیل مقدار زمان اجرا، درصد انحراف معیار طول و زمان بسیار بالا گزینه‌ی مطلوب و قابل اتكایی نمی‌باشد

۳ - ۲ - تحلیل نتایج با بیست تکرار

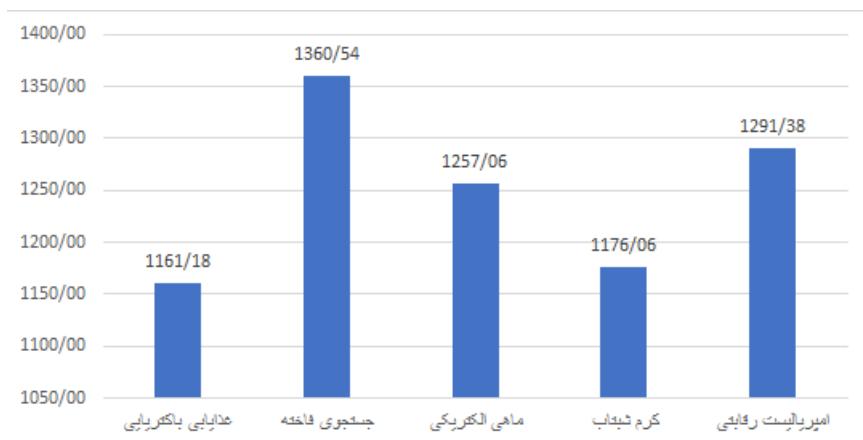
با استفاده از داده‌های موجود در جداول بخش دوم پیوست ژ می‌توانیم مقدار میانگین و درصد انحراف معیار داده‌ها را برای هر الگوریتم با بیست تکرار بدست آورده و با یکدیگر مقایسه کنیم. با استفاده از میانگین حدود طول و زمان اجرای هر الگوریتم می‌توانیم عملکرد این الگوریتم‌ها را با یکدیگر مقایسه نمائیم و از طریق بررسی درصد انحراف معیار حدود بازه تغییرات داده‌ها در هر اجرا را برای هر الگوریتم متوجه می‌شویم. در نتیجه با بررسی این دو نوع داده می‌توانیم بهترین الگوریتم را از جهات گوناگون بیاییم.

استفاده از الگوریتم‌های فراتکاری در مسیریابی ربات

جدول ۳ - ۷: نتایج اجرا الگوریتم‌ها با ۲۰ تکرار

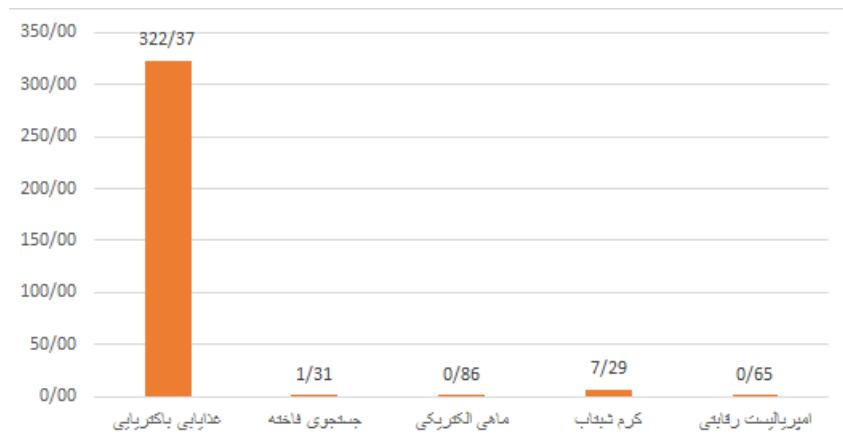
نام الگوریتم	درصد انحراف معیار زمان	درصد انحراف معیار طول	درصد انحراف معیار مسیر	انحراف معیار طول اجرا(ثانیه)	انحراف معیار زمان اجرا	میانگین مسیر	میانگین زمان اجرا(ثانیه)
غذایابی باکتریابی	۶.۲	۴.۴	۵۱.۴۱	۲۰.۰۸	۱۱۶۱.۱۸	۱۱۶۱.۱۸	۳۲۲.۳۷
جستجوی فاخته	۷.۷	۳.۲	۴۳.۹۷	۰.۱۰	۱۳۶۰.۰۴	۱۳۶۰.۰۴	۱.۳۱
ماهی الکتریکی	۸.۸	۶.۰	۷۵.۳۲	۰.۰۸	۱۲۵۷.۰۶	۱۲۵۷.۰۶	۰.۸۶
کرم شبتاب	۴.۵	۳.۴	۴۰.۴۶	۰.۳۳	۱۱۷۶.۰۶	۱۱۷۶.۰۶	۷.۲۹
امپریالیست رقابتی	۶.۰	۹.۷	۱۲۴.۸۰	۰.۰۴	۱۲۹۱.۳۸	۱۲۹۱.۳۸	۰.۶۵

از آنجایی که تفاوت داده‌ها در نمودار واضح‌تر از فرم عددی می‌باشد، در شکل‌های زیر داده‌های جدول ۳ - ۷ به صورت نمودارهای میله نمایش داده‌ایم تا بتوانیم مقایسه واضح‌تری داشته باشیم.

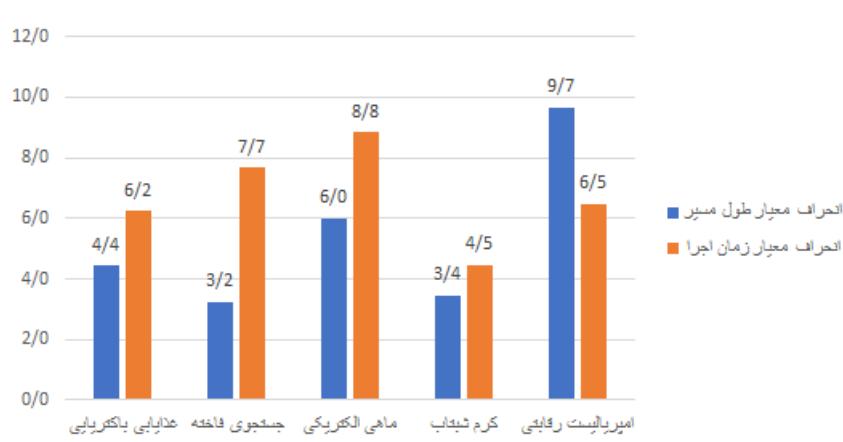


شکل ۳ - ۴: نمودار میله‌ای میانگین طول مسیر الگوریتم‌ها با ۲۰ تکرار

استفاده از الگوریتم‌های فراتکاری در مسیریابی ربات



شکل ۳ - ۵: نمودار میله‌ای میانگین زمان اجرا الگوریتم‌ها با ۲۰ تکرار



شکل ۳ - ۶: نمودار میله‌ای درصد انحراف معیار زمان و طول مسیر الگوریتم‌ها با ۲۰ تکرار

با نمودارهای مرتبط با اجرای الگوریتم‌ها با ۲۰ تکرار بهترین الگوریتم‌ها از هر منظر به شکل زیر مرتب می‌شوند:

- از حیث طول مسیر الگوریتم غذایابی باکتریایی موفق به یافتن کوتاهترین مسیر میان پنج الگوریتم شده است. طول مسیر یافته شده توسط الگوریتم‌های کرم شبتاب، ماهی‌الکتریکی، امپریالیست‌رقابتی و جستجوی‌فاخته به ترتیب پنج درصد، هشت درصد، یازده درصد و هفده درصد بیشتر از الگوریتم غذایابی باکتریایی می‌باشدند. در نتیجه می‌توان گفت در کنار الگوریتم غذایابی باکتریایی، الگوریتم کرم شبتاب هم به خوبی عمل کرده است.
- از حیث زمان اجرا الگوریتم، امپریالیست‌رقابتی سریع‌ترین عملکرد را داشته است. زمان اجرای الگوریتم‌های ماهی‌الکتریکی، جستجوی‌فاخته، کرم شبتاب، غذایابی باکتریایی به ترتیب سی درصد، دویست درصد، هزار و صد و بیست درصد و چهارهزار و نهصد و پنجاه درصد بیشتر از الگوریتم امپریالیست‌رقابتی بوده است. در نتیجه این الگوریتم از منظر سرعت پردازش نسبت به سایر الگوریتم‌های مورد بررسی بی‌رقیب محسوب می‌شود.
- از منظر درصد انحراف معیار طول مسیر، جستجوی‌فاخته و کرم شبتاب بهترین عملکرد را داشته‌اند. مقدار درصد

انحراف معیار الگوریتم‌های غذایابی باکتریایی، ماهی‌الکتریکی، امپریالیست رقابتی به ترتیب یک درصد، سه درصد و شش درصد بیشتر از درصد انحراف معیار جستجوی فاخته و کرم شبتاب می‌باشد.

- از حیث درصد انحراف معیار زمان اجرا، کرم شبتاب بهترین عملکرد را داشته‌اند. مقدار درصد انحراف معیار الگوریتم غذایابی باکتریایی، امپریالیست رقابتی، جستجوی فاخته و ماهی‌الکتریکی به ترتیب دو درصد، دو درصد، سه درصد و چهار درصد بیشتر از درصد انحراف معیار کرم شبتاب می‌باشد.

در نتیجه می‌توان گفت هر الگوریتم در یکی از زمینه‌های بررسی بهتر از سایر گزینه‌ها عمل کرده است. هر چند با کنار هم گذاشتن داده‌های بالا می‌توان نتیجه‌گیری کرد که الگوریتم کرم شبتاب در تمام زمینه‌ها بجز زمان اجرای الگوریتم بهتر از سایر الگوریتم‌ها عمل کرده است. در نتیجه این الگوریتم برای موقعیت‌هایی که نیاز به یافتن کوتاه‌ترین مسیر ممکن در کنار کمترین احتمال تغییرات در طول مسیر و زمان اجرا می‌باشد، بهترین گزینه ممکن است.

۳-۳-۳ جمع‌بندی

با توجه به داده‌های دو بخش قبلی می‌توان به این نتیجه رسید که:

- الگوریتم غذایابی باکتریایی با توجه به اینکه زمان زیادی برای اجرای هر حلقه‌ی خود دارد در تعداد تکرارهای پایین‌تر می‌تواند نتایج بهتری را از منظر طول مسیر ایجاد کند، هرچند این الگوریتم زمان اجرای بسیار بزرگ‌تری نسبت به رقیبان خود دارد.
- الگوریتم امپریالیست رقابتی کمترین زمان اجرا را هم با بیست تکرار و هم با صد تکرار از خود ثبت کرده است اما باید توجه داشت که با توجه به خاصیت اتمام حلقه در صورت رسیدن به یک امپراطوری در این الگوریتم مقدار انحراف معیار زمان اجرای زیادی را در تکرارهای کمتر از خود نشان می‌دهد (هر چند که این انحراف معیار با توجه به زمان اجرای کوتاه تغییرات بزرگی ایجاد نمی‌کند). بطور کل می‌توان گفت این الگوریتم مناسب موقعیت‌هایی است که سرعت، بیشتر از دقت نیاز است.
- الگوریتم ماهی‌الکتریکی نیز بسیار مشابه الگوریتم امپریالیست رقابتی عمل کرده است، هر چند این الگوریتم تفاوت‌هایی نیز با الگوریتم امپریالیست رقابتی دارد. الگوریتم ماهی‌الکتریکی کمی از الگوریتم امپریالیست رقابتی کندتر می‌باشد و مسیرهای نسبتاً کوتاه‌تری را می‌تواند کشف کند. همچنین، این الگوریتم انحراف معیار نسبتاً کمتری دارد. در نتیجه می‌توان گفت این الگوریتم نیز مشابه الگوریتم امپریالیست رقابتی مناسب موقعیت‌هایی است که سرعت، بیشتر از دقت نیاز است ولی اهمیت دقت و کوتاهی مسیر کمی بیشتر از موقعیت‌هایی است که از الگوریتم امپریالیست استفاده می‌کنیم.

- الگوریتم جستجوی فاخته نیز نسبت به کرم شبتاب و غذایابی باکتریایی زمان کمتری برای اجرا نیاز دارد اما با توجه به داده‌ها هرچه تعداد تکرار این الگوریتم بیشتر باشد، عملکرد بهتری را از منظر یافتن مسیر مطلوب با درصد انحراف معقول دارد. هرچند در میان الگوریتم‌های مورد بررسی از منظر دقیق، کرم شبتاب، از منظر سرعت امپریالیست‌رقابتی و ماهی‌الکتریکی و از منظر نتایج بهتر در تعداد تکرار کمتر الگوریتم غذایابی باکتریایی می‌تواند گزینه‌های مطلوب‌تری باشند. با این وجود با توجه به درصد انحراف معیار کم، هم با بیست تکرار و هم با صد تکرار، گزینه‌ی مناسبی برای موقعیت‌هایی است که به کمترین بازه تغییرات از نظر طول مسیر نیاز داریم.
- الگوریتم کرم شبتاب نیز با توجه به نتایج اجرا خود هم در بیست تکرار و هم در صد تکرار نشان داده است که بهترین گزینه برای موقعیت‌هایی است که به کوتاه‌ترین مسیر با کمترین نرخ تغییرات نیاز دارد، می‌باشد. این الگوریتم پس از الگوریتم غذایابی باکتریایی طولانی‌ترین زمان اجرا را ثبت کرده است، هر چند با افزایش تعداد تکرار توانسته انحراف معیار زمان اجرای بسیار خوبی را نشان دهد. در نتیجه می‌توان گفت در صورت نیاز به نرخ تغییرات کم زمانی، در کنار یافتن کوتاه‌ترین مسیر با کمترین نرخ تغییرات، بهتر است از این الگوریتم با تعداد تکرار حلقه بیشتر استفاده شود.

مراجع

- [١] Guo, C., et al. (٢٠٢١). "A survey of bacterial foraging optimization." *Neurocomputing* ٤٥٢: ٧٢٨-٧٤٦.
- [٢] Fister, I., et al. (٢٠١٣). "A comprehensive review of firefly algorithms." *Swarm and evolutionary computation* ١٣: ٣٤-٤٦.
- [٣] Hosseini, S. and A. Al Khaled (٢٠١٤). "A survey on the imperialist competitive algorithm metaheuristic: implementation in engineering domain and directions for future research." *Applied Soft Computing* ٢٤: ١٠٧٨-١٠٩٤.
- [٤] Li, F.-F., et al. (٢٠٢٢). "Path planning and smoothing of mobile robot based on improved artificial fish swarm algorithm." *Scientific reports* ١٢(١): ١٠٩.
- [٥] Mohanty, P. K. and D. R. Parhi (٢٠١٦). "Optimal path planning for a mobile robot using cuckoo search algorithm." *Journal of Experimental & Theoretical Artificial Intelligence* ٤٨(١-٢): ٣٥-٥٢.
- [٦] Patle, B., et al. (٢٠١٨). "Path planning in uncertain environment by using firefly algorithm." *Defence technology* ١٤(٦): ٧٩١-٧٩١.
- [٧] Sariff, N. and N. Buniyamin (٢٠٠٧). *An overview of autonomous mobile robot path planning algorithms*. ٢٠٠٧ ٤th student conference on research and development, IEEE.
- [٨] Shehab, M., et al. (٢٠١٧). "A survey on applications and variants of the cuckoo search algorithm." *Applied Soft Computing* ٦١: ١٠٤١-١٠٥٩.
- [٩] Yilmaz, S. and S. Sen (٢٠٢٠). "Electric fish optimization: a new heuristic algorithm inspired by electrolocation." *Neural Computing and Applications* ٣٤(١٥): ١١٥٤٣-١١٥٧٨.
- [١٠] Zeng, Z., et al. (٢٠١٥). "Imperialist competitive algorithm for AUV path planning in a variable ocean." *Applied artificial intelligence* ٢٩(٤): ٤٠٢-٤٢٠.
- [١١] Zhang, H.-y., et al. (٢٠١٨). "Path planning for the mobile robot: A review." *Symmetry* ١٠(١٠): ٤٥٠.

پیوست‌ها

پیوست آ: توابع تست برای بهینه‌سازی تک هدفه

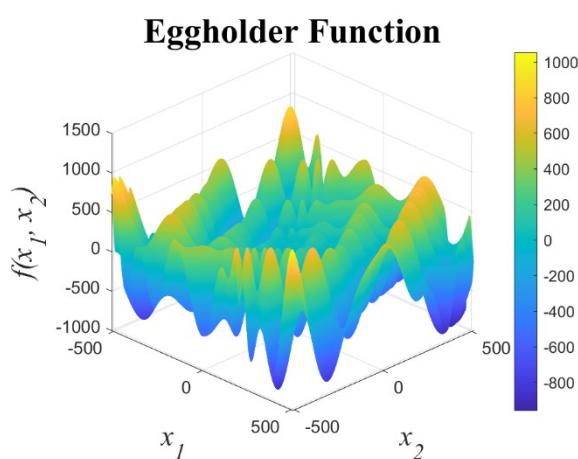
در ریاضیات کاربردی، توابع تست برای ارزیابی ویژگی‌های الگوریتم‌های بهینه‌سازی مفید هستند، مانند:

- نرخ همگرایی
- دقت
- قدرت
- عملکرد عمومی

در این پیوست به بررسی تنها تابع تست استفاده شده در این پروژه، یعنی تابع تست شانه تخم مرغی^۱ می‌پردازیم. در ادامه شکل کلی معادله، فرم ۳ بعدی این تابع تست و نحوه نگارش کد مربوط به نمایش این تابع آورده شده است.

آ-۱ تابع شانه تخم مرغی

در تابع شانه تخم مرغی به دلیل وجود تعداد زیاد مینیمم‌های محلی، بهینه‌سازی مشکل است. زیرا در تمام این مینیمم‌ها گرادیان تابع می‌تواند به صفر میل کند و به عبارت دیگر این نقطه را به عنوان مینیمم سراسری در نظر بگیرد. در نتیجه، این تابع گزینه‌ی مناسبی برای بررسی قدرت و دقت الگوریتم‌های بهینه‌سازی ما می‌باشد.



^۱ Egg holder function

این تابع معمولاً بر روی مربع $[-512, 512]$ برای تمامی $i = 1, 2$ ارزیابی می‌شود.

فرمول	اپتیموم جهانی
$f(x, y) = -(y + 47) \sin \sqrt{\frac{x}{2} + (y + 47)} - x \sin \sqrt{ x - (y + 47) }$	$f(512, 404, 2319) = -959.6407$

در ادامه کد مطلب برای رسم کانتور دو بعدی و سه بعدی تابع شانه تنخ مرغی آورده شده است:

رسم نمودار دو بعدی

```
% egg holder plot
warning off

x1min=-512;
x1max=512;
x2min=-512;
x2max=512;
R=1500; % steps resolution
x1=x1min:(x1max-x1min)/R:x1max;
x2=x2min:(x2max-x2min)/R:x2max;

for j=1:length(x1)
    for i=1:length(x2)
        f(i)=-x1(j)*sin(sqrt(abs(x1(j)-x2(i)-47))) - (x2(i)+47)*...
            sin(sqrt(abs(0.5*x1(j)+x2(i)+47)));
    end
    f_tot(j,:)=f;
end

% 1-dimensional plot is not applicable with this benchmark function
nexttile % contour plotting
mesh(x1,x2,f_tot); view(0,90); colorbar; set(gca, 'FontSize',12);
xlabel('x_2', 'FontName', 'Times', 'FontSize', 20, 'FontAngle', 'italic');
ylabel('x_1', 'FontName', 'Times', 'FontSize', 20, 'FontAngle', 'italic');
zlabel('f(X)', 'FontName', 'Times', 'FontSize', 20, 'FontAngle', 'italic');
title('X-Y Plane View', 'FontName', 'Times', 'FontSize', 24, 'FontWeight', 'bold');
```

رسم نمودار سه بعدی

```
% Egg Holder function plot
warning off

% Define the range for x1 and x2
x1min = -512;
x1max = 512;
x2min = -512;
x2max = 512;

% Steps resolution
R = 1500;

% Generate values for x1 and x2
x1 = linspace(x1min, x1max, R);
x2 = linspace(x2min, x2max, R);

% Initialize f_tot matrix for storing function values
f_tot = zeros(R, R);

% Calculate the Egg Holder function for each (x1, x2) pair
for j = 1:length(x1)
    for i = 1:length(x2)
        f(i) = -x1(j) * sin(sqrt(abs(x1(j) - x2(i) - 47))) - (x2(i) + 47) * ...
            sin(sqrt(abs(0.5 * x1(j) + x2(i) + 47)));
    end
    f_tot(j,:) = f;
end

% 3D mesh plot of the Egg Holder function
figure;
mesh(x1, x2, f_tot);

% Configure the view and labels
view(45, 30); % Adjust view angle for better visualization
colorbar; % Display a color bar for reference
set(gca, 'FontSize', 12);
xlabel('x_1', 'FontName', 'Times', 'FontSize', 20, 'FontAngle', 'italic');
ylabel('x_2', 'FontName', 'Times', 'FontSize', 20, 'FontAngle', 'italic');
zlabel('f(x_1, x_2)', 'FontName', 'Times', 'FontSize', 20, 'FontAngle', 'italic');
title('Eggholder Function', 'FontName', 'Times', 'FontSize', 24, 'FontWeight',
'bold');
```

پیوست ب: الگوریتم‌ها

در این پیوست شبه کدهای الگوریتم‌های مورد بحث ارائه می‌شوند:

ب-۱ الگوریتم کرم شب‌تاب

الگوریتم ۱ الگوریتم کرم شب‌تاب

۱: تابع هدف $f(X)$ ، $X = (x_1, \dots, x_d)^T$

۲: جامعه اولیه کرم شب‌تاب $X_i (i = 1, 2, \dots, n)$ را ایجاد کنید.

۳: شدت نور I_i در X_i را توسط $f(X_i)$ مشخص کنید.

۴: ضریب جذب نور α را تعریف کنید.

۵: تا زمانی که شرط خروج برقرار شود

۶: برای $i = 1 : n$ (تمام کرم شب‌تاب‌ها)

برای $j = 1 : n$

اگر $I_j > I_i$

کرم شب‌تاب i را به سمت کرم شب‌تاب j در فضای d حرکت دهید

پایان شرط

جذابیت با فاصله r توسط $\exp[-\gamma r]$ تغییر می‌کند.

جواب‌های جدید را ارزیابی کنید و شدت نور را به روزرسانی کنید.

پایان حلقه

۷. پایان حلقه

۸. کرم‌های شب‌تاب را رتبه بندی کنید و بهترین کرم فعلی را پیدا کنید.

۹. پایان حلقه

۱۰. نتایج و نمودارها را بررسی کنید.

ب-۲ الگوریتم جستجوی فاخته

الگوریتم ۲ الگوریتم جستجوی فاخته

۱: تابع هدف $f(X)$, $X = (x_1, \dots, x_d)^T$

۲: جامعه اولیه لانه میزبان $X_i (i = 1, 2, \dots, n)$ را ایجاد کنید.

۳. تا زمانی که شرط خروج برقرار نشود

یک فاخته تصادفی توسط پرواز لوی به دست آورید و شایستگی F_i آن را ارزیابی کنید.

یک لانه به صورت تصادفی (j) انتخاب کنید.

اگر $(F_i > F_j)$

جواب زرا جایگزین i کنید.

پایان شرط

قسمتی (P_a) از بدترین لانه‌ها را ترک کنید و لانه‌های جدید ایجاد کنید.

لانه‌ها با بهترین جواب را نگه دارید.

جواب‌ها را رتبه بندی کنید و بهترین لانه فعلی را پیدا کنید.

۴. پایان چرخه

۵. نتایج و نمودارها را بررسی کنید.

ب-۳ الگوریتم امپریالیست رقابتی

الگوریتم ۳ الگوریتم امپریالیست رقابتی

۱: پارامترهای الگوریتم را تعیین کرده و جمعیت اولیه امپراتوری را ایجاد کنید.

$$i = 1 : N_{population} \quad \text{برای ۲}$$

شایستگی هر کشور را حساب کرده و رتبه بندی کنید.

به تعداد مشخص از بین کشورها امپریالیست انتخاب کنید.

قدرت نرمالیزه شده هر امپریالیست را محاسبه کنید.

سایر کشورها را به عنوان کلونی به هر امپریالیست نسبت دهید.

۳: پایان حلقه

۴: تا زمانی که شرط خروج برقرار شود

مستعمرات را به امپریالیست نزدیک کنید.

اگر جواب بهتر بود

جواب جدید را جایگزین قبلی کنید.

در غیر این صورت

جواب قبل را تغییر ندهید.

پایان شرط

تعداد کلونی‌هایی که باید تغییر کند را بدست آورید

به تعداد بدست آمده موقعیت‌های جدید و تصادفی برای کلونی محاسبه کنید.

اگر احتمال انقلاب از عددی تصادفی بیشتر بود

موقعیت جدید به دست آمده را جایگزین موقعیت قبل کنید

پایان شرط

$$i = 1 : N_{imp} \quad \text{برای}$$

اگر هزینه کلونی کمتر از هزینه امپریالیست آن است

کلونی جایگزین امپریالیست می‌شود

پایان شرط

پایان حلقه

ضعیف‌ترین امپریالیست را انتخاب کنید

اگر کلونی داشته باشد

ضعیف‌ترین کلونی آن را پیدا کرده و به امپریالیستی که بیشترین احتمال را دارد، نسبت دهید

در غیر اینصورت

امپریالیست را حذف کنید

پایان شرط

۵: پایان حلقه

ب-۴ الگوریتم ماهی الکترونیکی

الگوریتم ۴ الگوریتم ماهی الکترونیکی مختصر

۱: ایجاد جمعیت اولیه

۲: ارزیابی کیفیت افراد

۳: تکرار کنید

بسته به فرکانس هر فرد^f، جمعیت را به دو زیر جمعیت فعال(N_A) و غیرفعال(N_P) تقسیم کنید.

انجام الکتروولوکیشن فعال و غیرفعال به ترتیب برای افراد متعلق به N_A و N_P .

مقادیر فرکانس(f) و دامنه(A) را برای هر فرد بروز کنید.

۴: تا زمانی که معیار پایان برآورده شود.

ب-۵ الگوریتم غذایابی باکتریایی

الگوریتم ۵ الگوریتم غذایابی باکتریایی

۱: با پارامترهای $P, S, N_s, N_{Re}, N_{ed}, P_{ed}, C(i)$ مقدار اولیه دهید.

۲: تا زمانی که شرط خروج برقرار شود

$l = 1, 2, \dots, N_{ed}$ برای

حلقه حذف پراکندگی را شروع کنید.

$K = 1, 2, \dots, N_{Re}$ برای

حلقه تولیدمثل را شروع کنید.

$i = 1, 2, \dots, S$ برای

حلقه کموتاکسی را شروع کنید.

شاپرستگی $J(i, j, k, l)$ را محاسبه کنید.

$J(i, j, k, l)$ را برابر J_{Last} قرار دهید.

بغلتید $\Delta(i)$ را در $[1, 10]$ ایجاد کنید.

حرکت کنید $\theta(i, j+1, k, l)$ را محاسبه کنید.

($t = 1 : N_s$ برای) شنا کنید

اگر $J(i, j=1, k, l)$ کمتر از J_{Last} باشد.

J را به روز رسانی کنید.

θ را به روز رسانی کنید.

در غیر اینصورت

توقف کنید

خروج از شرط

خروج از حلقه

وضعیت سلامت J_{health} باکتری i را محاسبه کنید.

باکتری‌ها را توسط J_{health} به صورت صعودی رتبه بندی کنید.

S_r برای

بهترین نصف J_{health} را تکثیر کنید.

پایان حلقه

پایان حلقه

برای $m = 1, 2, \dots, S$

هر باکتری با $P_{ed} \leq rand$ را حذف کنید.

پایان حلقه

پایان حلقه

۳: پایان حلقه

پیوست پ: موقعیت ذرات در بررسی الگوریتم با ورودی‌های تصادفی

برای بررسی نتایج همگرایی و قدرت هر الگوریتم، در صحه‌گذاری انجام شده بروی تابع شانه تخم مرغی با مقدار کمینه ۹۵۹.۶۴۰۷، موقعیت تمام ذرات هر جمعیت را استخراج کرده و در جداول این پیوست آورده‌ایم تا بتوان از این نتایج برای جمع‌بندی نهایی در مورد عملکرد هر الگوریتم استفاده کرد. باید توجه داشت که داده‌های زیر مربوط به موردی از اجرای کد می‌باشد که به تنهایی در حال بررسی می‌باشد و صرفا برای درک نسبی از اینکه آیا الگوریتم‌ها نقاط خود را به خوبی به سمت نقطه‌ی مورد نظر همگرا می‌کنند یا نه می‌باشد.

پ-۱ بهینه‌سازی غذایابی باکتریایی

جدول پ - ۱: موقعیت باکتری‌های الگوریتم بهینه‌سازی غذایابی باکتریایی

شماره	موقعیت X	موقعیت Y	شماره	موقعیت X	موقعیت Y
۱	۵۱۲	۴۰۴/۳۱۸	۳۱	۴۳۱/۱۰۹	۴۶۹/۰۲۸
۲	۵۱۲	۴۰۵/۴۲	۳۲	-۴۹۰/۹۹۷	۱۱/۷۶۵
۳	۵۱۲	۴۰۵/۴۲	۳۳	۲۷۴/۱۸۴	-۲۰۱/۱۱۵۴
۴	۵۱۲	۴۰۵/۶۰۷	۳۴	۵۰۱/۰۵۲۴	۷۳/۴۶۲۴
۵	۵۱۲	۴۰۵/۶۰۷	۳۵	-۱۸۴/۴۶۴	۲۵۸/۷۱۴
۶	۵۱۱/۴۵	۴۰۲/۳۵۵	۳۶	-۸۴/۵۸۱۱	۱۸۴/۷۱۸
۷	۵۱۰/۵۶۸	۴۰۳/۳۰۶	۳۷	-۱۴۳/۴۰۸	۳۹۲/۲۳۱
۸	۵۱۲	۴۰۱/۹۶۷	۳۸	-۳۳۱/۳۷۸	-۲۴۳/۹۲۲
۹	۵۱۰/۳۱۴	۴۰۳/۹۲	۳۹	-۱۸۹/۵۰۳	-۲۲۲/۱۴۳
۱۰	۵۱۰/۳۱۴	۴۰۳/۹۲	۴۰	-۲۷/۴۶۶۴	-۳۱۰/۰۰۵
۱۱	۵۱۲	۴۰۱/۲۸۷	۴۱	۲۶۸/۱۸۱	۲۱۰/۷۲۳
۱۲	۵۱۲	۴۰۱/۲۸۷	۴۲	-۱۹۸/۲۷۳	-۱۰۸/۳۱۸
۱۳	۵۰۹/۵۲۱	۴۰۳/۲۶۸	۴۳	-۱۲۷/۷۸۸	۴۸۷/۱۵۶
۱۴	۵۰۹/۰۷۳	۴۰۲/۱۴۹	۴۴	-۴/۵۱۲۵۷	-۲۹۹/۳۲۲
۱۵	۵۰۹/۰۷۳	۴۰۲/۱۴۹	۴۵	۳۷۱/۹۶۳	۴۳۱/۲۰۷
۱۶	۵۱۲	۴۰۷/۴۱۲	۴۶	-۲۸/۵۳۷۷	۲۱۷/۹۲۸
۱۷	۵۰۹/۲۹۳	۴۰۳/۱۵۹	۴۷	-۱۰۱/۷۴۸	-۱۶۸/۳۵۷
۱۸	۵۰۸/۱۷۷۴	۴۰۰/۷۲۵	۴۸	۲۰/۷۰۷۴	۴۰/۴۶۳۷
۱۹	۵۱۱/۱۷	۴۰۶/۵۹۷	۴۹	-۳۶۰/۷۲	۲۲۳/۹۱۶
۲۰	۵۰۸/۷۴۹	۳۹۹/۱۱۵	۵۰	-۵۰/۹۱۰۱	-۴۴/۰۸۹۴
۲۱	۵۰۸/۱۲۴	۴۰۲/۲۷۵	۵۱	-۳۷۹/۴۹۷	۵۱/۷۹۲۲
۲۲	۵۰۸/۱۲۴	۴۰۲/۲۷۵	۵۲	۱۱۴/۱۰۳	-۱۸۴/۱۰۵
۲۳	۵۱۱/۸۱	۴۰۸/۱۸۹	۵۳	-۴۲۲/۳۱۸	۴/۳۸۴۷۸
۲۴	۵۰۹/۳۴۱	۳۹۸/۵۹۲	۵۴	-۳۲۷/۲۳۸	۸۴/۰۵۸۳
۲۵	۵۰۹/۳۴۱	۳۹۸/۵۹۲	۵۵	-۲۹۰/۹۲۶	۱۲۲/۲۹۹

-۴۹۰/۹۹۳	۱۳۶/۳۳۹	۵۶	۴۰۲/۳۴۶	۵۰۷/۶۶۸	۲۶
-۲۹۰/۵۶۷	-۳۱۸/۶۸۱	۵۷	۳۹۹/۴۲۷	۵۰۵/۳۵۳	۲۷
-۳۸۱/۱۸۱	-۱۳۵/۰۱۶	۵۸	۴۰۹/۴۷۳	۵۱۲	۲۸
۴۹۰/۷۰۹	۱۹/۶۴۷۲	۵۹	۴۰۵/۴۵۹	۵۰۸/۷۶۶	۲۹
۴۰۱/۴۸۷	-۳۱۳/۴	۶۰	۴۰۵/۴۵۹	۵۰۸/۷۶۶	۳۰

پ-۲ جستجوی فاخته

جدول پ - ۲: موقعیت لانه‌های الگوریتم جستجوی فاخته

شماره	موقعیت X	شماره	موقعیت Y	موقعیت X	شماره
۱	۵۱۲	۳۱۳/۹۷۲	۳۱	۴۰۴/۲۷۱۷	۵۱۲
۲	۵۱۲	-۳۱۳/۹۶	۳۲	۴۰۴/۴۲۲۵	۵۱۲
۳	۵۱۲	-۳۱۴	۳۳	۴۰۴/۵۰۸۴	۵۱۲
۴	۵۱۲	-۳۱۳/۹۷۶	۳۴	۴۰۳/۸۳۳۷	۵۱۲
۵	۵۱۲	-۳۱۴/۱۰۷	۳۵	۴۰۳/۳۴۶۷	۵۱۲
۶	۵۱۲	-۳۱۳/۸۱۲	۳۶	۴۰۲/۹۹۹۵	۵۱۲
۷	۵۱۲	-۳۱۳/۶۲۸	۳۷	۴۰۲/۲۲۶۴	۵۱۲
۸	۵۱۲	-۳۱۳/۴۷۳	۳۸	۴۰۲/۰۹۷۳	۵۱۱/۱۰۶۷
۹	-۳۸۲/۲۰۸	-۴۵۶/۳۷۴	۳۹	۴۰۶/۳۷۵۱	۵۱۲
۱۰	-۳۸۲/۳۹۲	-۴۵۶/۰۵۷	۴۰	۴۰۰/۷۲۱۴	۵۱۲
۱۱	-۳۸۰/۴۹	-۴۰۰/۶۹۴	۴۱	۴۰۰/۸۱۶۹	۴۳۶/۸۰۱۵
۱۲	-۴۱۸/۵۱۱	-۳۹۳/۵۷۴	۴۲	۴۶۲/۲۵۸۷	۴۴۷/۴۸۴۵
۱۳	-۴۸۷/۳۱۶	۲۸۲/۸۸۹۸	۴۳	۳۸۰/۴۳۹۱	-۴۶۰/۹۶۹
۱۴	-۴۸۷/۴۲۱	۲۸۳/۳۱۰۲	۴۴	۳۸۰/۲۸۱	-۴۶۰/۳۰۴
۱۵	-۴۸۷/۹۰۳	۲۸۲/۷۷۴۰	۴۵	۳۸۴/۹۱۶۹	-۴۶۰/۰۰۲
۱۶	-۳۶۷/۶۴۹	۳۹۹/۰۰۶۱	۴۶	۳۸۷/۳۴۷۵	-۴۶۶/۹۰۵
۱۷	-۳۶۷/۶۳۲	۳۹۹/۳۲۲۱	۴۷	۳۸۷/۳۹۵۴	-۴۶۷/۹۳۷
۱۸	-۳۶۷/۰۹۱	۳۹۹/۷۷۳۶	۴۸	۳۸۳/۶۶۲۳	-۴۶۰/۰۳۷
۱۹	-۲۷۷/۶۸۷	۴۹۶/۰۰۰۲	۴۹	۳۸۹/۶۱۷۲	-۴۷۰/۳۴۸
۲۰	-۳۰۰/۱۹۴	۴۶۱/۷۰۴۱	۵۰	۴۹۹/۴۴۸۵	۳۴۷/۳۲۰۶
۲۱	-۴۶۱/۹۲۴	-۲۹۴/۷۳۰	۵۱	۴۹۹/۳۲۲۳	۳۴۷/۳۰۰۶
۲۲	-۴۶۲/۳۷	-۲۹۴/۸۹۱	۵۲	۴۹۹/۲۸۲۸	۳۴۷/۲۲۶۸
۲۳	-۴۶۱/۷۰۳	-۲۹۴/۶۷۸	۵۳	۴۹۹/۰۰۳۴	۳۴۷/۰۸۴۹
۲۴	-۵۱۲	-۱۷۴/۹۲۰	۵۴	۴۹۹/۲۱۸۱	۳۴۷/۳۱۹۰
۲۵	-۵۱۲	-۱۷۴/۹۲۶	۵۵	۴۹۹/۷۱۹۱	۳۴۷/۶۲۱۱
۲۶	-۵۱۲	-۱۷۴/۹۳	۵۶	۴۹۸/۸۷۰۳	۳۴۵/۹۷۲۰
۲۷	-۵۱۲	-۱۷۴/۹۷۱	۵۷	۳۹۳/۰۲۷	-۴۶۹/۳۱۴
۲۸	۲۷۷/۴۸۲۳	-۵۱۲	۵۸	۳۹۶/۰۸۱۰	-۴۷۲/۷۰۷
۲۹	۶/۷۰۸۷۸	-۵۰۵/۹۱۴	۵۹	۵۱۲	۳۵۲/۶۰۳۰
۳۰	۵۷/۲۳۴۴۰	۵۱۲	۶۰	۳۷۱/۲۹۰۹	-۴۶۲/۰۷۴

پ-۳ ماهی الکتریکی

جدول پ - ۳: موقعیت ماهی‌های الکتریکی

شماره	موقعیت X	موقعیت Y	شماره	موقعیت X	موقعیت Y
۱	-۲۸۲/۹۴۲	-۴۸۷/۰۸۶	۲۱	-۱۷۵/۳۱۲	-۵۱۱/۹۴۳
۲	-۴۶۰/۶۸۱	۳۸۰/۶۹۹۵	۲۲	۰۰۸/۲۸۷۱	-۲۸۱/۲۷۷
۳	۰۱۲	۴۰۲/۷۷۶۸	۲۳	-۵۰۵/۸۴۱	۱۰/۳۸۲۲۷
۴	۴۷۴/۱۷۷۵	۴۲۹/۴۲۹۲	۲۴	۲۸۳/۱۶۴۰	-۴۸۷/۲۲۳۹
۵	۵۰۳/۲۳۶۸	۴۵۴/۱۴۶۳	۲۵	۰۱۲	۴۰۴/۵۸۷۵
۶	۴۹۷/۰۵۱۶	-۲۷۳/۴۱	۲۶	۴۹۴/۹۸۹۴	-۲۷۲/۲۰۷
۷	۰۱۲	۴۰۳/۹۹۰۹	۲۷	۴۴۵/۸۹۸۶	۴۶۰/۹۳۲۳
۸	-۱۷۴/۹۰۵۹	-۵۱۲	۲۸	۲۸۳/۱۰۰۱	-۴۸۷/۱۸۴
۹	-۵۱۲	۴۰۰/۵۶۳۵	۲۹	-۴۶۰/۰۱۸	۳۸۰/۸۰۲۰
۱۰	۳۵۳/۷۶۶۴	۵۰۸/۶۸۸۳	۳۰	-۵۱۲	۶/۲۱۹۷۱
۱۱	۲۸۳/۱۴۲۵	-۴۸۷/۲۸۷	۳۱	-۴۶۰/۶۸	۳۸۰/۷۱۸۶
۱۲	-۳۱۲/۴۸۱	۵۱۲	۳۲	-۵۱۲	۴۱۴/۱۸۱۴
۱۳	-۴۵۲/۸۶۲	-۳۷۸/۳۳۴	۳۳	۳۰۰/۸۴۰۸	۵۰۱/۳۴۹۱
۱۴	-۴۶۰/۶۷۶	۳۸۰/۶۴۴۷	۳۴	۴۰۲/۲۱۲۹	-۳۶۷/۲۰۴
۱۵	۲۸۲/۹۲۷۲	-۴۸۶/۸۹۱	۳۵	۴۱۰/۸۹۶۹	۱۷۰/۶۱۳۶
۱۶	۳۴۸/۶۶۳۱	۵۰۰/۹۸۳۸	۳۶	-۴۰۳/۷۶۳۴	۵۰۱۲
۱۷	-۴۱۶/۷۲۶	۹۹/۴۱۳۰۳	۳۷	۲۸۳/۲۶۸۳	-۴۸۷/۱۰۲
۱۸	-۴۶۰/۶۸۲	۳۸۰/۷۱۷	۳۸	-۱۰۸/۹۳۷	-۵۱۲
۱۹	-۳۸۴/۹۷۳	-۴۰۷/۸۳۳	۳۹	-۴۶۰/۴۹	۳۸۰/۶۲۶۶
۲۰	۰۱۲	۴۰۴/۲۶۸۴	۴۰	۳۴۷/۴۴۴۰	۰۰۰/۴۰۰۱
۲۱	۲۸۳/۱۸۴۷	-۴۸۷/۳۴۱	۴۱	۳۵۸/۸۷۳۳	۰۱۲
۲۲	-۲۹۸/۲۴۰	-۴۶۰/۱۷۰	۴۲	-۳۱۴/۰۱	۰۱۲
۲۳	۲۴۰/۸۰۷	۲۶۳/۳۸۰۳	۴۳	۴۴۰/۶۲۲۷	۴۰۰/۰۶۰۸
۲۴	۴۶۴/۵۱۰۳	-۳۰۱/۷۲۱	۴۴	۰۱۲	۴۰۵/۸۸۸۷
۲۵	-۱۷۸/۸۲۷	-۵۱۲	۴۵	-۴۰۰/۶۰۰	-۴۲۰/۱۰۶
۲۶	-۴۵۸/۳۹۴	-۳۸۴/۹۸۴	۴۶	-۴۶۰/۵۸۴	۳۸۰/۶۹۸۰
۲۷	-۳۱۴/۰۹۶	۰۱۲	۴۷	-۲۹۷/۳۵۶	-۴۶۰/۹۹۶
۲۸	۳۴۷/۰۵۷۹۸	۴۹۹/۰۱۳۱	۴۸	-۴۶۰/۴۵۰	۳۸۰/۶۶۶۶
۲۹	۳۴۶/۴۲۸۰	۴۹۸/۶۳۷۱	۴۹	-۱۷۴/۹۱	-۵۱۲
۳۰	-۳۱۳/۹۹۸	-۵۱۲	۵۰	۴۶۶/۴۲۲۱	-۳۰۲/۳۱۸

پ-۴ بهینه‌سازی کرم شب تاب

جدول پ - ۴: موقعیت کرم‌های الگوریتم بهینه‌سازی کرم شب تاب

شماره	موقعیت X	موقعیت Y	شماره	موقعیت X	موقعیت Y
۱	۵۱۲	۳۱	۴۰۴/۳۱۰۴	۵۱۲	۳۷۰/۰۴۳۵
۲	۵۱۲	۳۲	۵۰۱/۸۹۰۶	۳۴۱/۷۵۷۹	۵۱۲
۳	۵۱۲	۳۳	۵۱۲	-۳۰۵/۰۹۲	۵۱۲
۴	۵۱۲	۳۴	۵۱۲	۳۴۰/۳۱۵	۵۱۲
۵	۱۲۲/۰۵۸۹	۳۵	۵۱۲	-۳۳۸/۴۰۳	۲۵۱/۷۹۸۹
۶	۳۷۹/۶۸۴۶	۳۶	۵۱۲	۲۵۲/۹۸۴۱	۴۷۶/۴۵۸۸
۷	-۳۴۹/۳۸۶	۳۷	۵۱۲	-۱۵۸/۰۵۱	۵۱۲
۸	-۵۱۲	۳۸	۳۹۴/۰۱۹۱	۱۸۴/۰۶۵۲	-۵۱۲
۹	-۱۷۹/۹۷۳	۳۹	-۵۱۲	۱۷۰/۹۶۳۲	-۲۱۲/۹۴۹
۱۰	-۳۱۳/۰۶	۴۰	-۴۶۷/۰۵۶۳	-۴۳۱/۱۳۳۵	-۵۱۲
۱۱	-۵۱۲	۴۱	۴۱۳/۰۷۹۲	۳۶۲/۸۳۳۵	-۲۶۴/۶۱۸
۱۲	-۵۱۲	۴۲	۲۸۸/۷۳۷۶	۱۹/۷۰۳۶۳	-۵۱۲
۱۳	۲۶۳/۶۷۳	۴۳	-۵۱۲	۵۱۲/۷۵۲	-۴۵۰/۷۵۲
۱۴	۵۱۱/۲۲۶۴	۴۴	۵۹/۸۷۱۰۵	۵۱۲	۲۰۷/۷۱۶۵
۱۵	-۲۰۹/۸۹	۴۵	-۵۱۲	۵۱۲	۳۴۹/۸۰۹۶
۱۶	-۴۸۵/۰۷	۴۶	۱/۷۷۴۸۳۵	-۱۰۵/۷۷۲	۵۱۲
۱۷	-۴۲۶/۰۴۳	۴۷	-۱۸۰/۰۴	-۴۲۶/۲۹۱	-۵۱۲
۱۸	-۱۲۵/۷۹۴	۴۸	-۵۱۲	۵۶/۱۶۱۰۲	۵۱۲
۱۹	۵۱۲	۴۹	-۱۹۹/۲۰۱	-۵۱۲	-۱۷۹/۰۶۱
۲۰	-۵۱۲	۵۰	۳۴/۷۳۵۲۳	۵۱۲	-۵۱۲
۲۱	-۲۲۵/۱۳۱	۵۱	۵۱۲	-۱۶/۷۳۹۳	۵۱۲
۲۲	-۳۸۷/۰۹۸	۵۲	۱۳۵/۳۵۷۸	-۵۱۲/۰۸۵	-۵۱۲
۲۳	-۲۲۷/۴۳	۵۳	۴۷۲/۴۴۴	-۵۱۲	-۵۱۲
۲۴	۴۵۴/۶۶۰۸	۵۴	۴۴۲/۳۲۲۷	-۵۱۲	-۵۱۲
۲۵	-۳۹۴/۴۹۵	۵۵	۵۱۲	-۵۱۲/۷۲۳	-۵۱۲
۲۶	۳۲۹/۲۰۸۳	۵۶	-۱۰۲/۸۲۳	-۵۱۲/۱۱۹	-۵۰۴/۱۱۹
۲۷	-۱۷۸/۶۵۷	۵۷	-۲۶۱/۰۱۴	۵۱۲	-۵۱۲
۲۸	۳۶۹/۳۲۷۶	۵۸	-۵۱۲	-۵۱۲	۵۱۲
۲۹	-۳۱۳/۴۷۸	۵۹	-۵۱۲	-۵۱۲	۵۱۲
۳۰	-۵۱۲	۶۰	۳۲۱/۶۹۷۵	-۵۱۲	۵۱۲

پ-۵ بهینه‌سازی امپریالیست رقابتی

جدول پ - ۵: موقعیت کشورهای الگوریتم بهینه سازی امپریالیست رقابتی

شماره	موقعیت X	موقعیت Y	شماره	موقعیت X	موقعیت Y
۱	-۴۶۳/۳۰۳	-۳۸۷/۸۷۴	۳۱	۵۱۲	۳۴۵/۴۴۴
۲	-۴۶۵/۶۷۸	۳۸۵/۶۹۱۴	۳۲	-۵۱۲	۴۰۳/۶۰۴۴
۳	۵۱۲	۴۰۴/۲۰۵۲	۳۳	-۵۱۲	۵۱۲
۴	-۳۱۳/۷۲۷	۵۱۲	۳۴	-۲۹۷/۷۸۹	۵۱۲
۵	۴۸۲/۰۶۲۹	۴۳۲/۶۱۲۲	۳۵	-۴۷۶/۵۳	۵۱۲
۶	-۴۵۹/۷۵۳	-۳۸۵/۱۱۶	۳۶	-۴۷۳/۵۸۹	-۵۱۲
۷	-۴۶۱/۳۹۸	-۲۷۹/۶۳۷	۳۷	-۵۱۲	-۳۳۳/۵۸۹
۸	-۴۶۹/۱۴۷	-۴۴۹/۰۶۴	۳۸	-۳۱۲/۲۲۳	۵۱۲
۹	-۴۴۸/۸۲۷	۴۶۸/۱۱۲۸	۳۹	-۵۱۲	۴۹۲/۷۸۴۳
۱۰	-۲۹۶/۶۲	۲۹۹/۰۶۲۱	۴۰	-۳۹۷/۴۴۱	۵۱۲
۱۱	-۴۶۷/۰۲۶	۳۸۵/۷۳۱۴	۴۱	-۲۰۴/۴۲	۵۱۲
۱۲	-۴۶۷/۱۸۱	۳۴۰/۷۳۱۴	۴۲	-۵۱۲	-۵۱۲
۱۳	-۴۶۷/۲۸۴	۳۸۵/۹۵۱۶	۴۳	۴۲۴/۴۹۴۵	۲۸۷/۵۴۲۶
۱۴	-۴۶۴/۷۴۶	۳۸۲/۴۰۹	۴۴	۵۱۲	۵۱۲
۱۵	-۴۶۱/۵۶۷	۳۷۳/۴۵۲۹	۴۵	۴۸۲/۰۶۳۶	۴۳۲/۶۱۴۲
۱۶	۳۳۳/۵۲۹۱	-۱۷۲/۴۵۷	۴۶	۵۱۲	۱۷۱/۸۰۴۸
۱۷	-۲۵۳/۴۲۹	۴۷۷/۶۶۰۸	۴۷	۴۸۷/۴۶۶۵	۴۵۳/۸۰۲۵
۱۸	-۴۶۱/۷۷۲	۴۰۴/۲۷۰۳	۴۸	۱۸۰/۰۸۳۵	۵۱۲
۱۹	-۴۷۳/۵۰۷	۴۲۵/۴۲۸	۴۹	۵۰۱/۶۲۸۹	۵۰۸/۹۲۹
۲۰	-۴۶۵/۶۷۹	۳۸۵/۶۷۸	۵۰	۵۰۹/۹۴۷۶	۳۸۱/۲۰۵۶
۲۱	۵۱۲	۵۱۲	۵۱	۷۰/۱۸۰۴۲	-۳۱۴/۸۶۷
۲۲	۵۱۲	۴۰۳/۳۷۸۴	۵۲	-۳۰۴/۲۱۸	۵۹/۸۸۹۰۸
۲۳	۵۱۲	۴۰۳/۴۶۴۵	۵۳	-۴۷۳/۱۲۹	-۳۷۱/۵۴۵
۲۴	۵۱۲	۴۰۳/۴۹۳۲	۵۴	-۱۰/۳۲۴۹	-۵۱۲
۲۵	۵۱۲	۴۱۱/۳۵۶۷	۵۵	-۴۳۸/۸۰۶	-۳۸۷/۹۵۰۲
۲۶	۴۸۱/۰۲۲۹	۳۹۸/۹۲۸۲	۵۶	-۴۰۹/۶۵۴	-۳۸۲/۹۶۱
۲۷	۵۱۲	۴۴۱/۰۹۹۵	۵۷	-۴۶۸/۰۳۳	-۳۵۸
۲۸	۵۱۲	-۸۰/۲۴۷	۵۸	-۴۲۵/۳۳۳	-۳۶۲/۷۷۸
۲۹	۵۱۲	۴۰۳/۶۵۴	۵۹	-۴۰۹/۲۹۲	-۳۷۷/۴۸۶
۳۰	۵۱۲	۵۱۲	۶۰	-۱۴۰/۳۳۷	-۳۷۸/۶۴۲

پیوست ج: موقعیت ذرات در بررسی الگوریتم با ورودی‌های تعیین شده

برای بررسی نتایج همگرایی و قدرت هر الگوریتم، در صحه‌گذاری انجام شده بروی تابع شانه تخم مرغی با مقدار کمینه -959.6407 ، موقعیت تمام ذرات هر جمعیت را استخراج کرده و در جداول این پیوست آورده‌ایم تا بتوان از این نتایج برای جمع‌بندی نهایی در مورد عملکرد هر الگوریتم استفاده کرد. از این داده‌ها برای مقایسه این الگوریتم‌ها در بخش نتایج داده در فصل دوم استفاده شده است. تا بتوانیم از طریق بررسی میانگین، ماکزیمم، مینیمم و بطورکلی روند این داده‌ها به مقایسه خوبی از عملکرد الگوریتم‌ها برسیم.

ج-۱ بهینه‌سازی غذایابی باکتریایی

جدول ج - ۱: داده‌های خروجی از اجرای الگوریتم غذایابی باکتریایی با ورودی تعیین شده

اجرای کد اول					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مستقله
۵/۶۳۰۸۹۲	۴۰۴/۳۲۴۷	۵۱۲	-۹۵۹/۶۳۰۹	۵۳	top right
۱/۶۳۱۱۰۵	۴۰۴/۲۴۵۸	۵۱۲	-۹۵۹/۶۴۰۴	۱۸	top left
۱۴/۱۳۴۱۳۹	۴۰۴/۳۱۹۶	۵۱۲	-۹۵۹/۶۳۱۹	۱۳۲	Secondary_Axis
۴۲/۹۷۸۱۴۱	۴۰۴/۱۴۰۶	۵۱۲	-۹۵۹/۶۳۱۲	۴۴۵	right
۵/۲۳۵۴۶۶	۴۰۴/۱۶۳۲	۵۱۲	-۹۵۹/۶۳۵۳	۴۸	main axis
۱۳/۰۱۶۰۰۷	۴۰۴/۱۴۲۳	۵۱۲	-۹۵۹/۶۳۱۸	۱۳۵	left
۲/۵۲۵۲۸۵	۴۰۴/۲۹۵۹	۵۱۲	-۹۵۹/۶۳۶۰	۲۷	down_right
۲۰/۸۱۸۱۸۵	۴۰۴/۲۲۲۱	۵۱۲	-۹۵۹/۶۴۰۶	۲۰۷	down_left
اجرای کد دوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مستقله
۳/۲۹۸۷۰۸	۴۰۴/۲۲۲۳	۵۱۲	-۹۵۹/۶۴۰۶	۱۸	top right
۱/۴۷۹۴۱۴	۴۰۴/۱۸۲۶	۵۱۲	-۹۵۹/۶۳۸۰	۹	top left
۸/۹۰۹۸۴۴	۴۰۴/۲۸۹۱	۵۱۲	-۹۵۹/۶۳۶۹	۸۰	Secondary_Axis
۲۱/۰۴۲۰۹	۴۰۴/۱۸۰۱	۵۱۲	-۹۵۹/۶۳۷۶	۱۹۵	right
۵/۱۴۹۱۱۸	۴۰۴/۱۹۵	۵۱۲	-۹۵۹/۶۳۹۱	۴۴	main axis
۱/۷۴۰۸۵۸	۴۰۴/۳۲۰۴	۵۱۲	-۹۵۹/۶۳۱۷	۱۳	left
۷/۵۳۵۸۲۳	۴۰۴/۲۳۵۹	۵۱۲	-۹۵۹/۶۴۰۶	۶۴	down_right
۸۲/۶۲۰۳۳۲	۴۰۴/۳۱۰۴	۵۱۲	-۹۵۹/۶۳۳۶	۶۸۲	down_left
اجرای کد سوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مستقله
۷/۹۲۸۲۲۵	۴۰۴/۱۶۷۳	۵۱۲	-۹۵۹/۶۳۵۹	۶۴	top right

۱/۲۵۶۰۶۷	۴۰۴/۳۲۱۱	۵۱۲	-۹۵۹/۶۳۱۶	۱۲	<i>top left</i>
۶/۷۱۷۴۱۲	۴۰۴/۱۴۴۴	۵۱۲	-۹۵۹/۶۳۲۰	۵۸	<i>Secondary_Axis</i>
۴۵/۳۱۱۰۹۹	۴۰۴/۱۸۹۹	۵۱۲	-۹۵۹/۶۳۸۷	۳۷۵	<i>right</i>
۰/۷۴۴۸۰۲	۴۰۴/۱۹۰۲	۵۱۲	-۹۵۹/۶۳۸۷	۶	<i>main axis</i>
۲/۸۰۶۲۴۲	۴۰۴/۱۴۷۴	۵۱۲	-۹۵۹/۶۳۲۶	۲۳	<i>left</i>
۲/۸۶۸۹۱۹	۴۰۴/۲۶۷۱	۵۱۲	-۹۵۹/۶۳۹۲	۲۴	<i>down_right</i>
۰/۸۷۸۴۴۶	۴۰۴/۱۸۰۱	۵۱۲	-۹۵۹/۶۳۷۶	۷	<i>down_left</i>

اجرای کد چهارم

موقعیت داده ورودی در فضای مسئله	تعداد تکرار چرخه	مقدار هزینه	موقعیت نهایی ذره(X)	موقعیت نهایی ذره(Y)	زمان اجرا(ثانیه)
۱۵	-۹۵۹/۶۴۰۴	۵۱۲	۴۰۴/۲۴۶۳	۴۰۴/۲۲۱۷۹۶	۲/۳۲۱۷۹۶
۲۲۳	-۹۵۹/۶۳۹۵	۵۱۲	۴۰۴/۲۶۴۳	۴۰۴/۹۵۱۳۵۲	۲۵/۹۵۱۳۵۲
۱۰۴	-۹۵۹/۶۳۹۷	۵۱۲	۴۰۴/۲۶۱	۴۰۴/۱۳۷۰۵۵	۱۳/۹۳۷۰۵۵
۶۸	-۹۵۹/۶۴۰۱	۵۱۲	۴۰۴/۲۵۴۷	۴۰۴/۸۶۰۷۴۳	۷/۸۶۰۷۴۳
۲۲۸	-۹۵۹/۶۳۳۹	۵۱۲	۴۰۴/۳۰۸۹	۴۰۴/۶۳۳۶۵۴	۲۶/۶۳۳۶۵۴
۱۵۲	-۹۵۹/۶۳۹۸	۵۱۲	۴۰۴/۲۰۳۸	۴۰۴/۱۸۰۸۵	۱۸/۰۸۰۸۵
۱۴۵	-۹۵۹/۶۳۳۹	۵۱۲	۴۰۴/۳۰۸۸	۴۰۴/۱۲۸۲۱۳	۱۷/۱۲۸۲۱۳
۳۱۷	-۹۵۹/۶۳۷۵	۵۱۲	۴۰۴/۲۸۴۲	۴۰۴/۲۷۱۸۲۸	۳۹/۲۷۱۸۲۸

اجرای کد پنجم

موقعیت داده ورودی در فضای مسئله	تعداد تکرار چرخه	مقدار هزینه	موقعیت نهایی ذره(X)	موقعیت نهایی ذره(Y)	زمان اجرا(ثانیه)
۳۵۳	-۹۵۹/۶۳۲۵	۵۱۲	۴۰۴/۱۴۷۳	۴۰۴/۹۵۸۰۷۷	۴۲/۹۵۸۰۷۷
۱۹	-۹۵۹/۶۳۴۸	۵۱۲	۴۰۴/۱۶۰۲	۴۰۴/۱۳۹۰۱۲	۲/۱۳۹۰۱۲
۲۱۹	-۹۵۹/۶۳۱۱	۵۱۲	۴۰۴/۳۲۳۴	۴۰۴/۵۱۴۵۳۵	۲۸/۵۱۴۵۳۵
۱۳۶	-۹۵۹/۶۳۷۸	۵۱۲	۴۰۴/۱۸۱۶	۴۰۴/۹۵۲۷۱۳	۱۵/۹۵۲۷۱۳
۸۹۲	-۹۵۹/۶۳۷۲	۵۱۲	۴۰۴/۲۸۷۱	۴۰۴/۱۰۶۰۵۹۱۷۶۴	۱۰/۰۵۹۱۷۶۴
۳۶	-۹۵۹/۶۳۷۵	۵۱۲	۴۰۴/۱۷۹۱	۴۰۴/۸۱۸۷۰۳	۴/۸۱۸۷۰۳
۱۲	-۹۵۹/۶۳۵۲	۵۱۲	۴۰۴/۱۶۲۵	۴۰۴/۴۵۹۸۸۹	۱/۴۵۹۸۸۹
۱۰۴	-۹۵۹/۶۳۶۹	۵۱۲	۴۰۴/۲۸۹۲	۴۰۴/۱۲۷۱۱۳۳	۱۲/۲۷۱۱۳۳

ج-۲ بهینه‌سازی جستجوی فاخته

جدول ج - ۲: داده‌های خروجی از اجرای الگوریتم جستجوی فاخته با ورودی تعیین شده

اجرای کد اول					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۱۵۶۱۸	۴۰۴/۲۴۷۲	۵۱۲	-۹۵۹/۶۴۰۴	۱۹۳	top right
۰/۶۸۴۳	۴۰۴/۲۴۷۷	۵۱۲	-۹۵۹/۶۴۰۴	۱۳۱	top left
۰/۱۸۰۷۶	۴۰۴/۱۴۲۴	۵۱۲	-۹۵۹/۶۳۱۶	۲۸۲	Secondary_Axis
۰/۲۴۹۴۴	۴۰۴/۳۲۴۶	۵۱۲	-۹۵۹/۶۳۰۹	۵۴۰	right
۰/۱۵۴۴۹	۴۰۴/۲۸۹۵	۵۱۲	-۹۵۹/۶۳۶۹	۳۲۷	main axis
۱/۰۲۳۶	۴۰۴/۱۷۳۱	۵۱۲	-۹۵۹/۶۳۶۷	۲۰۰۲	left
۰/۱۵۷۸۳	۴۰۴/۲۱۲۳	۵۱۲	-۹۵۹/۶۴۰۲	۳۰۴	down_right
۰/۸۹۰۴۵	۴۰۴/۱۷۸۴	۵۱۲	-۹۵۹/۶۳۷۴	۱۸۰۲	down_left
اجرای کد دوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۱۸۱۱۱	۴۰۴/۱۸۴۱	۵۱۲	-۹۵۹/۶۳۸۱	۳۱۳	top right
۰/۱۱۷۲۵	۴۰۴/۳۰۷۵	۵۱۲	-۹۵۹/۶۳۴۱	۲۶۲	top left
۰/۱۴۴۰۹	۴۰۴/۳۱۷۲	۵۱۲	-۹۵۹/۶۳۲۴	۳۰۰	Secondary_Axis
۰/۲۴۲۷۱	۴۰۴/۱۷۷۹	۵۱۲	-۹۵۹/۶۳۷۴	۵۰۰	right
۰/۲۵۴۷	۴۰۴/۳۰۴۷	۵۱۲	-۹۵۹/۶۳۴۶	۵۶۰	main axis
۱/۲۴۵۴	۴۰۴/۱۶۱۴	۵۱۲	-۹۵۹/۶۳۵۰	۲۷۵۴	left
۰/۴۲۰۰۸	۴۰۴/۱۸۱	۵۱۲	-۹۵۹/۶۳۷۷	۹۲۱	down_right
۰/۳۶۱۲۱	۴۰۴/۱۸۴۱	۵۱۲	-۹۵۹/۶۳۸۱	۶۶۷	down_left
اجرای کد سوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۱۷۶۰۹	۴۰۴/۲۶۹۷	۵۱۲	-۹۵۹/۶۳۹۰	۲۱۳	top right
۰/۴۴۹۴۱	۴۰۴/۲۷۷۶	۵۱۲	-۹۵۹/۶۳۸۳	۹۶۹	top left
۰/۴۱۵۲	۴۰۴/۲۷۳۹	۵۱۲	-۹۵۹/۶۳۸۶	۲۶۰	Secondary_Axis
۰/۶۲۸۸۱	۴۰۴/۱۶۱۳	۵۱۲	-۹۵۹/۶۳۵۰	۷۱۳	right
۰/۱۸۶۰۳	۴۰۴/۱۹۸۳	۵۱۲	-۹۵۹/۶۳۹۴	۲۴۰	main axis
۰/۳۴۸۵۳	۴۰۴/۱۶۲۹	۵۱۲	-۹۵۹/۶۳۵۳	۵۴۳	left
۰/۲۲۷۹۹۲	۴۰۴/۳۰۷۱	۵۱۲	-۹۵۹/۶۳۴۲	۳۲۸	down_right
۰/۵۰۹۷۶	۴۰۴/۳۲۴۴	۵۱۲	-۹۵۹/۶۳۰۹	۷۸۹	down_left
اجرای کد چهارم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۳۳۲۶۹	۴۰۴/۳۲۴۸	۵۱۲	-۹۵۹/۶۳۰۸	۴۶۹	top right
۰/۳۹۷۳۲	۴۰۴/۱۴۱۱	۵۱۲	-۹۵۹/۶۳۱۳	۸۲۰	top left
۰/۲۹۴۷۴	۴۰۴/۱۴۱	۵۱۲	-۹۵۹/۶۳۱۳	۴۶۶	Secondary_Axis

۰/۳۰۹۱۹	۴۰۴/۳۱۵۶	۵۱۲	-۹۵۹/۶۳۲۷	۴۹۷	<i>right</i>
۰/۱۵۵۳۲	۴۰۴/۲۰۳۷	۵۱۲	-۹۵۹/۶۳۹۸	۲۲۰	<i>main axis</i>
۰/۵۶۱۲۲	۴۰۴/۱۴۴۶	۵۱۲	-۹۵۹/۶۳۲۰	۹۱۰	<i>left</i>
۰/۴۵۷۳	۴۰۴/۳۱۹۷	۵۱۲	-۹۵۹/۶۳۱۹	۶۸۸	<i>down_right</i>
۰/۹۷۸۱۷	۴۰۴/۱۳۹۶	۵۱۲	-۹۵۹/۶۳۱۰	۱۶۲۳	<i>down_left</i>
اجرای کد پنجم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۳۳۰۳۶	۴۰۴/۱۷۵۳	۵۱۲	-۹۵۹/۶۳۷۰	۴۳۴	<i>top right</i>
۰/۵۵۲۵۷	۴۰۴/۲۷۶۵	۵۱۲	-۹۵۹/۶۳۸۴	۹۳۱	<i>top left</i>
۰/۰۴۲۵۵۹	۴۰۴/۲۲۴۷	۵۱۲	-۹۵۹/۶۴۰۷	۵۳	<i>Secondary_Axis</i>
۰/۱۶۸۲۱	۴۰۴/۲۷۷۸	۵۱۲	-۹۵۹/۶۳۸۳	۲۳۲	<i>right</i>
۰/۱۴۴۱۵	۴۰۴/۲۹۲۹	۵۱۲	-۹۵۹/۶۳۶۴	۱۹۸	<i>main axis</i>
۰/۵۷۳۳۶	۴۰۴/۱۴۲۷	۵۱۲	-۹۵۹/۶۳۱۷	۸۹۳	<i>left</i>
۰/۶۴۵۰۸	۴۰۴/۳۱۹۸	۵۱۲	-۹۵۹/۶۳۱۸	۹۰۸	<i>down_right</i>
۰/۵۴۶۷۶	۴۰۴/۱۳۹۲	۵۱۲	-۹۵۹/۶۳۰۹	۷۸۰	<i>down_left</i>

ج-۳ بهینه‌سازی ماهی الکتریکی

جدول ج - ۳: داده‌های خروجی از اجرای الگوریتم ماهی الکتریکی با ورودی تعیین شده

اجرای کد اول					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۲۹۹۵۸	۴۰۴/۲۰۷۲	۵۱۲	-۹۵۹/۶۴۰۰	۲۲	<i>top right</i>
۰/۶۶۷۵۹	۴۰۴/۲۶۱۳	۵۱۲	-۹۵۹/۶۳۹۷	۱۳۱	<i>top left</i>
۰/۷۵۳۴۳	۴۰۴/۳۱۷۴	۵۱۲	-۹۵۹/۶۳۲۳	۱۳۷	<i>Secondary_Axis</i>
۰/۸۹۸۱۲	۴۰۴/۲۶۲۴	۵۱۲	-۹۵۹/۶۳۹۶	۱۷۸	<i>right</i>
۱/۰۵۰۳	۴۰۴/۲۶۳۴	۵۱۲	-۹۵۹/۶۳۹۵	۱۹۲	<i>main axis</i>
۰/۷۵۱۱۵	۴۰۴/۲۰۴۴	۵۱۲	-۹۵۹/۶۳۹۸	۱۳۴	<i>left</i>
۱/۴۱۶۲	۴۰۴/۱۴۲۴	۵۱۲	-۹۵۹/۶۳۱۶	۲۴۶	<i>down_right</i>
۰/۵۶۲۴۹	۴۰۴/۱۷۱	۵۱۲	-۹۵۹/۶۳۶۵	۹۵	<i>down_left</i>
اجرای کد دوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۱/۷۲۸۴	۴۰۴/۱۸۵۵	۵۱۲	-۹۵۹/۶۳۸۲	۳۳۸	<i>top right</i>
۱/۳۲۸۹	۴۰۴/۲۰۸۲	۵۱۲	-۹۵۹/۶۴۰۰	۲۶۷	<i>top left</i>
۰/۹۸۷۸۷	۴۰۴/۲۱۲۷	۵۱۲	-۹۵۹/۶۴۰۲	۱۸۶	<i>Secondary_Axis</i>
۰/۴۶۳۰۶	۴۰۴/۱۸۶۷	۵۱۲	-۹۵۹/۶۳۸۴	۸۴	<i>right</i>
۱/۱۴۴۹	۴۰۴/۱۹۲۸	۵۱۲	-۹۵۹/۶۳۸۹	۲۳۵	<i>main axis</i>
۱/۳۹۵۲	۴۰۴/۳۱۰۹	۵۱۲	-۹۵۹/۶۳۳۵	۲۵۴	<i>left</i>

۱/۰۲۷	۴۰۴/۳۱۲۸	۵۱۲	-۹۵۹/۶۳۳۲	۲۱۹	<i>down_right</i>
۰/۵۰۰۶۶	۴۰۴/۲۹۴	۵۱۲	-۹۵۹/۶۳۶۳	۱۰۲	<i>down_left</i>
اجرای کد سوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۷۶۲۲	۴۰۴/۱۴۰۴	۵۱۲	-۹۵۹/۶۳۱۲	۱۳۴	<i>top_right</i>
۰/۴۱۱۰۵	۴۰۴/۱۸۸۹	۵۱۲	-۹۵۹/۶۳۸۶	۸۷	<i>top_left</i>
۲/۸۷۳۸	۴۰۴/۲۹۶۲	۵۱۲	-۹۵۹/۶۳۵۹	۶۳۹	<i>Secondary_Axis</i>
۰/۴۶۰۶۱	۴۰۴/۲۱۶۲	۵۱۲	-۹۵۹/۶۴۰۴	۸۷	<i>right</i>
۱/۲۳۴۴	۴۰۴/۲۴۸۵	۵۱۲	-۹۵۹/۶۴۰۳	۲۷۱	<i>main_axis</i>
۰/۴۲۰۴۲	۴۰۴/۲۹۰۶	۵۱۲	-۹۵۹/۶۳۶۷	۸۶	<i>left</i>
۰/۶۰۶۸۴	۴۰۴/۲۰۵۲	۵۱۲	-۹۵۹/۶۳۹۹	۱۲۷	<i>down_right</i>
۰/۴۴۷۹	۴۰۴/۲۵۰۷	۵۱۲	-۹۵۹/۶۴۰۳	۹۳	<i>down_left</i>
اجرای کد چهارم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۴۹۹۳۹	۴۰۴/۲۹۶۲	۵۱۲	-۹۵۹/۶۳۵۹	۶۹	<i>top_right</i>
۱/۰۵۷۱	۴۰۴/۱۴۶۲	۵۱۲	-۹۵۹/۶۳۳۳	۱۹۴	<i>top_left</i>
۲/۰۲۹۴	۴۰۴/۲۵۵۶	۵۱۲	-۹۵۹/۶۴۰۰	۴۴۶	<i>Secondary_Axis</i>
۰/۵۴۷۷۲	۴۰۴/۲۶۰۲	۵۱۲	-۹۵۹/۶۳۹۷	۹۹	<i>right</i>
۱/۲۳۶۲	۴۰۴/۳۱۷۸	۵۱۲	-۹۵۹/۶۳۲۲	۲۲۰	<i>main_axis</i>
۰/۳۹۹۲۷	۴۰۴/۲۲۱۱	۵۱۲	-۹۵۹/۶۴۰۵	۸۳	<i>left</i>
۱/۳۰۰۵۵	۴۰۴/۲۵۵۸	۵۱۲	-۹۵۹/۶۴۰۰	۲۶۵	<i>down_right</i>
۰/۹۱۷۵۳	۴۰۴/۲۱۱	۵۱۲	-۹۵۹/۶۴۰۲	۱۹۱	<i>down_left</i>
اجرای کد پنجم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۸۳۲۶۸	۴۰۴/۳۲۴۷	۵۱۲	-۹۵۹/۶۳۰۹	۱۵۳	<i>top_right</i>
۰/۵۷۶۴۶	۴۰۴/۱۵۲۱	۵۱۲	-۹۵۹/۶۳۳۶	۱۱۷	<i>top_left</i>
۰/۲۰۱۰۱	۴۰۴/۲۰۷۶	۵۱۲	-۹۵۹/۶۴۰۰	۳۸	<i>Secondary_Axis</i>
۰/۷۱۶۲۶	۴۰۴/۱۳۸۲	۵۱۲	-۹۵۹/۶۳۰۷	۱۴۶	<i>right</i>
۰/۵۲۱۰۵	۴۰۴/۲۷۵	۵۱۲	-۹۵۹/۶۳۸۵	۱۰۷	<i>main_axis</i>
۱/۳۳۳۵	۴۰۴/۳۱۲۶	۵۱۲	-۹۵۹/۶۳۳۱	۲۷۲	<i>left</i>
۱/۳۰۰۵۲	۴۰۴/۲۲۵۶	۵۱۲	-۹۵۹/۶۴۰۶	۲۶۸	<i>down_right</i>
۰/۸۸۴۷	۴۰۴/۲۱۱۳	۵۱۲	-۹۵۹/۶۴۰۲	۱۸۵	<i>down_left</i>

ج-۴ بهینه‌سازی کرم شبتاب

جدول ج - ۴: داده‌های خروجی از اجرای الگوریتم کرم شبتاب با ورودی تعیین شده

اجرای کد اول					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
-۰/۳۶۲۲۷	۴۰۴/۲۳۲۶	۵۱۲	-۹۵۹/۶۴۰۷	۲۲۹	top right
-۰/۳۵۴۴۳	۴۰۴/۲۷۲۵	۵۱۲	-۹۵۹/۶۳۸۸	۲۷۳	top left
-۰/۴۰۴۹۷	۴۰۴/۲۹۵۲	۵۱۲	-۹۵۹/۶۳۶۱	۳۲۴	Secondary_Axis
-۰/۶۷۵۰۲	۴۰۴/۲۱۷۲	۵۱۲	-۹۵۹/۶۴۰۴	۴۹۷	right
-۰/۰۵۸۶۹۶	۴۰۴/۲۲۴۸	۵۱۲	-۹۵۹/۶۴۰۶	۴۴	main axis
-۰/۳۷۸۵۶	۴۰۴/۲۵۲۲	۵۱۲	-۹۵۹/۶۴۰۲	۳۰۹	left
-۰/۰۳۹۲۸۷	۴۰۴/۲۰۵۵	۵۱۲	-۹۵۹/۶۳۹۹	۳۲	down_right
۱/۴۳۹۷	۴۰۴/۲۶۳۸	۵۱۲	-۹۵۹/۶۳۹۵	۱۱۷۷	down_left
اجرای کد دوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
-۰/۰۸۷۷۵۹	۴۰۴/۳۱۲	۵۱۲	-۹۵۹/۶۳۳۳	۶۳	top right
-۰/۴۰۴۷۳	۴۰۴/۲۴۱۹	۵۱۲	-۹۵۹/۶۴۰۵	۲۸۱	top left
-۰/۶۹۹۱۳	۴۰۴/۱۷۷۹	۵۱۲	-۹۵۹/۶۳۷۴	۵۸۰	Secondary_Axis
-۰/۰۶۹۳۳۵	۴۰۴/۱۸۹۲	۵۱۲	-۹۵۹/۶۳۸۶	۵۴	right
۱/۸۱۳۶	۴۰۴/۲۱۴۲	۵۱۲	-۹۵۹/۶۴۰۳	۱۵۴۶	main axis
۱/۳۹۶۸	۴۰۴/۲۱۸۹	۵۱۲	-۹۵۹/۶۴۰۵	۱۱۹۸	left
-۰/۶۶۵۹۳	۴۰۴/۱۵۶۸	۵۱۲	-۹۵۹/۶۳۴۳	۵۶۴	down_right
-۰/۰۴۳۳۳۹	۴۰۴/۲۲۶۷	۵۱۲	-۹۵۹/۶۴۰۶	۳۰	down_left
اجرای کد سوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
-۰/۴۹۳۵۴	۴۰۴/۲۴۶۳	۵۱۲	-۹۵۹/۶۴۰۴	۳۲۳	top right
-۰/۱۴۴۴۴	۴۰۴/۲۶۲۷	۵۱۲	-۹۵۹/۶۳۹۶	۱۰۹	top left
-۰/۴۵۴۹	۴۰۴/۱۹۹۴	۵۱۲	-۹۵۹/۶۳۹۵	۳۸۳	Secondary_Axis
-۰/۹۱۱۰۲	۴۰۴/۲۴۸۹	۵۱۲	-۹۵۹/۶۴۰۳	۷۶۰	right
۱/۱۲۶۲	۴۰۴/۱۹۰۹	۵۱۲	-۹۵۹/۶۳۸۸	۹۴۸	main axis
-۰/۳۸۴۳۳	۴۰۴/۱۴۳۴	۵۱۲	-۹۵۹/۶۳۱۸	۳۱۹	left
-۰/۸۸۲۱۴	۴۰۴/۱۵۶۶	۵۱۲	-۹۵۹/۶۳۴۲	۷۴۰	down_right
-۰/۴۸۸۲۲	۴۰۴/۲۳۳	۵۱۲	-۹۵۹/۶۴۰۷	۴۰۰	down_left
اجرای کد چهارم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۱/۶۰۵۲	۴۰۴/۲۵۳۶	۵۱۲	-۹۵۹/۶۴۰۱	۱۱۶۲	top right
-۰/۷۸۹۳۳	۴۰۴/۲۶۷۲	۵۱۲	-۹۵۹/۶۳۹۲	۶۶۸	top left
-۰/۲۸۴۹	۴۰۴/۲۲۶۹	۵۱۲	-۹۵۹/۶۴۰۶	۲۳۰	Secondary_Axis
-۰/۹۷۰۵۱	۴۰۴/۲۶۷	۵۱۲	-۹۵۹/۶۳۹۳	۸۰۲	right
-۰/۹۴۸۸	۴۰۴/۱۹۸	۵۱۲	-۹۵۹/۶۳۹۴	۸۰۲	main axis
-۰/۵۱۲۳۹	۴۰۴/۲۰۸۵	۵۱۲	-۹۵۹/۶۴۰۰	۴۲۳	left
-۰/۰۳۷۷۴۵	۴۰۴/۲۹۹۶	۵۱۲	-۹۵۹/۶۳۵۴	۲۸	down_right

۰/۴۹۴۰۸	۴۰۴/۲۰۴۳	۵۱۲	-۹۵۹/۶۳۹۸	۴۱۳	down_left
اجرای کد پنجم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۱۶۶۹	۴۰۴/۲۳۱۵	۵۱۲	-۹۵۹/۶۴۰۷	۴۸۰	top right
۰/۸۰۱۳۱	۴۰۴/۱۴۵۳	۵۱۲	-۹۵۹/۶۳۲۲	۶۶۴	top left
۰/۲۸۷۶۲	۴۰۴/۳۱۱۴	۵۱۲	-۹۵۹/۶۳۳۵	۲۳۸	Secondary_Axis
۰/۰۶۱۹۵	۴۰۴/۲۷۰۶	۵۱۲	-۹۵۹/۶۳۳۵	۳۷	right
۰/۰۶۰۶۰۷	۴۰۴/۱۴۳۹	۵۱۲	-۹۵۹/۶۳۱۹	۵۱	main axis
۰/۱۳۸۰۳	۴۰۴/۲۸۳۲	۵۱۲	-۹۵۹/۶۳۷۷	۱۱۱	left
۰/۶۱۴۴۲	۴۰۴/۱۵۷۵	۵۱۲	-۹۵۹/۶۳۴۴	۴۱۱	down_right
۰/۹۵۸۹	۴۰۴/۳۰۰۳	۵۱۲	-۹۵۹/۶۳۵۳	۶۸۰	down_left

ج-۵ بهینه‌سازی امپریالیست رقابتی

جدول ج - ۵: داده‌های خروجی از اجرای الگوریتم امپریالیست رقابتی با ورودی تعیین شده

اجرای کد اول					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۱۲۷۰۱	۴۰۴/۲۹۸۵	۵۱۲	-۹۵۹/۶۳۵۶	۱۷	top right
۰/۲۰۱۸۶	۴۰۴/۳۱۸۴	۵۱۲	-۹۵۹/۶۳۲۱	۵۶	top left
۰/۱۰۴۲	۴۰۴/۲۸۱۷	۵۱۲	-۹۵۹/۶۳۷۸	۳۵	Secondary_Axis
۰/۰۵۶۶۱	۴۰۴/۲۸۸۳	۵۱۲	-۹۵۹/۶۳۷۰	۲۰	right
۰/۲۲۷۴۴	۴۰۴/۲۲۳۸	۵۱۲	-۹۵۹/۶۴۰۶	۶۰	main axis
۰/۰۷۲۷۳۷	۴۰۴/۳۰۰۷	۵۱۲	-۹۵۹/۶۳۵۳	۱۸	left
۰/۱۰۱۳۲	۴۰۴/۲۳۸۹	۵۱۲	-۹۵۹/۶۴۰۶	۳۴	down_right
۰/۲۴۰۰۵	۴۰۴/۱۸۷۹	۵۱۲	-۹۵۹/۶۳۸۵	۵۸	down_left
اجرای کد دوم					
زمان اجرا(ثانیه)	موقعیت نهایی ذره(Y)	موقعیت نهایی ذره(X)	مقدار هزینه	تعداد تکرار چرخه	موقعیت داده ورودی در فضای مسئله
۰/۰۹۴۳۷۳	۴۰۴/۳۰۱۷	۵۱۲	-۹۵۹/۶۳۵۱	۲۸	top right
۰/۲۳۷۴	۴۰۴/۱۸۵۴	۵۱۲	-۹۵۹/۶۳۸۲	۱۰۷	top left
۰/۰۴۸۸۸۳	۴۰۴/۲۴۴۶	۵۱۲	-۹۵۹/۶۴۰۵	۱۲	Secondary_Axis
۰/۰۴۶۸۵۷	۴۰۴/۳۰۴۲	۵۱۲	-۹۵۹/۶۳۴۷	۱۷	right
۰/۱۳۱۰۵	۴۰۴/۲۶۵۹	۵۱۲	-۹۵۹/۶۳۹۳	۴۸	main axis
۰/۰۹۹۶۸۲	۴۰۴/۱۷۷۹	۵۱۲	-۹۵۹/۶۳۷۴	۳۸	left
۰/۰۷۲۸۸۸	۴۰۴/۲۳۹۲	۵۱۲	-۹۵۹/۶۴۰۶	۲۸	down_right
۰/۰۷۷۵۰۵	۴۰۴/۱۳۹۸	۵۱۲	-۹۵۹/۶۳۱۰	۲۹	down_left
اجرای کد سوم					

موقعیت داده ورودی در فضای مسئله	تعداد تکرار چرخه	مقدار هزینه	موقعیت نهایی ذره (X)	موقعیت نهایی ذره (Y)	زمان اجرا(ثانیه)
top right	۲۴	-۹۵۹/۶۳۹۶	۰۱۲	۴۰۴/۲۶۳	۰/۰۹۳۶۱۸
top left	۲۰	-۹۵۹/۶۳۷۰	۰۱۲	۴۰۴/۲۸۸۳	۰/۰۵۲۹۰۵
Secondary_Axis	۱۰	-۹۵۹/۶۴۰۴	۰۱۲	۴۰۴/۲۴۸۱	۰/۰۲۶۳۵۹
right	۵۸	-۹۵۹/۶۴۰۱	۰۱۲	۴۰۴/۲۵۳۲	۰/۱۶۰۶۲
main axis	۵۸	-۹۵۹/۶۳۸۶	۰۱۲	۴۰۴/۲۷۳۹	۰/۲۴۳۲۲
left	۵۴	-۹۵۹/۶۳۹۱	۰۱۲	۴۰۴/۲۶۹۳	۰/۱۵۸۹۶
down_right	۵۲	-۹۵۹/۶۳۹۳	۰۱۲	۴۰۴/۱۹۷۸	۰/۱۳۰۸۴
down_left	۲۷۲	-۹۵۹/۶۳۵۰	۰۱۲	۴۰۴/۳۰۲۴	۱/۲۱۷۵
اجرای کد چهارم					
موقعیت داده ورودی در فضای مسئله	تعداد تکرار چرخه	مقدار هزینه	موقعیت نهایی ذره (X)	موقعیت نهایی ذره (Y)	زمان اجرا(ثانیه)
top right	۹	-۹۵۹/۶۳۳۴	۰۱۲	۴۰۴/۳۱۱۶	۰/۰۲۹۴۶۸
top left	۷۵۲	-۹۵۹/۶۳۴۵	۰۱۲	۴۰۴/۳۰۰۵	۴/۱۶۵۸
Secondary_Axis	۱۲	-۹۵۹/۶۴۰۴	۰۱۲	۴۰۴/۲۱۷۷	۰/۰۲۹۸۶۸
right	۲۰	-۹۵۹/۶۴۰۰	۰۱۲	۴۰۴/۲۵۵۱	۰/۰۴۹۵۸۲
main axis	۵۷	-۹۵۹/۶۴۰۷	۰۱۲	۴۰۴/۲۲۹۲	۰/۱۷۳۲۷
left	۳۹	-۹۵۹/۶۳۸۱	۰۱۲	۴۰۴/۱۸۴۶	۰/۰۹۶۷۵۹
down_right	۳۹	-۹۵۹/۶۳۸۳	۰۱۲	۴۰۴/۲۷۷	۰/۰۹۷۴۳۳
down_left	۴۴	-۹۵۹/۶۴۰۴	۰۱۲	۴۰۴/۲۴۷۵	۰/۱۰۷۸۲
اجرای کد پنجم					
موقعیت داده ورودی در فضای مسئله	تعداد تکرار چرخه	مقدار هزینه	موقعیت نهایی ذره (X)	موقعیت نهایی ذره (Y)	زمان اجرا(ثانیه)
top right	۱۳	-۹۵۹/۶۳۴۸	۰۱۲	۴۰۴/۱۵۹۸	۰/۰۴۴۴۹۳
top left	۱۹۳	-۹۵۹/۶۴۰۵	۰۱۲	۴۰۴/۲۱۸۹	۰/۶۰۱۱۳
Secondary_Axis	۳۴	-۹۵۹/۶۳۹۳	۰۱۲	۴۰۴/۲۶۵۹	۰/۰۹۷۴۷۸
right	۱۶	-۹۵۹/۶۳۳۳	۰۱۲	۴۰۴/۳۱۲۳	۰/۰۴۳۱۷۶
main axis	۳۳۰	-۹۵۹/۶۴۰۶	۰۱۲	۴۰۴/۲۲۲۹	۱/۲۰۰۲
left	۳۶۱	-۹۵۹/۶۳۹۴	۰۱۲	۴۰۴/۱۹۸۵	۱/۲۴۰۹
down_right	۴۳	-۹۵۹/۶۳۳۶	۰۱۲	۴۰۴/۱۵۲۸	۰/۱۱۲۸۹
down_left	۴۵	-۹۵۹/۶۳۱۸	۰۱۲	۴۰۴/۳۲۰۲	۰/۱۲۲۳۶

پیوست ژ

در این پیوست موقعیت نقاط تشکیل دهنده منحنی مسیر نهایی، زمان اجرا، طول مسیر در هر اجرای الگوریتم‌ها به تفکیک در دو بخش قرار داده شده است. بخش اول مربوط به داده‌های حاصل از صد تکرار و بخش دوم داده‌های حاصل از بیست تکرار می‌باشد.

ژ-۱ نتایج با صد تکرار

جدول ژ - ۱: داده‌های الگوریتم غذایابی باکتریایی

موقعیت نقاط کنترل (n)							زمان اجرا (ثانیه)	طول مسیر	شماره
۱۹۳,۱۳۸۸	۱۸۴,۰۰۵۹	۱۲۲,۷۲۵۵	۴۶۱,۸۰۶۵	۳۲۰,۴۳۸۵	۲۴,۴۴۰۲۸	۱۸۳۸,۷	۱۱۶۵,۵	۱	
-۱۱,۴۹۶۸	-۳۰,۵,۶۸۹	-۳۹۱,۹۳۵	۳۲۸,۹۰۵۵	۱۰۴,۵۴۴۱	-۰,۱۶۷۸۵	۱۱۸۵,۳	۱۵۵۲	۲	
۷,۶۰۵۹۵۹	-۱۶۳,۶۱	-۲۷۶,۲۳۹	۲۸۳,۵۴۵	۱۰۰,۷۷۵۹	-۴۶,۶۸۵۰۲	۱۰۸۳,۶	۱۱۰۱,۳	۳	
۷۹,۵۴۱۵۴	-۸۴,۸۴۶۷	-۲۵۲,۶۰۰	۳۶۳,۵۷۸۵	۱۷۷,۷۸۳۷	-۸,۶۸۵۹۸	۱۰۹۵,۳	۱۱۰۲,۱	۴	
۱۰۰,۲۷۶۹	-۷۱,۹۰۹۵	-۲۴۶,۱۲۹	۳۸۳,۹۳۴۶	۲۰۰,۸۰۲۲	۳,۱۰۰۸۷۸	۱۰۷۸,۹	۱۱۰۲,۱	۵	

جدول ژ - ۲: داده‌های الگوریتم جستجوی فاخته

موقعیت نقاط کنترل (n)							زمان اجرا (ثانیه)	طول مسیر	شماره
۸۰,۹۷۹۲۳	-۳,۸۴۶۵	-۲۸۱,۱۷۶	۱۸۰,۳۴۸۶	۱۸۰,۱۳۰۲۶۳۸	-۳۸,۴۸۴۹۰۱۰۳	۶,۳۹۳۶۱۸۴	۱۱۷۹,۳	۱	
-۳۷,۸۱۶۴	-۲۲۶,۴۸۵	-۳۸۷,۲۰۱	۳۰۷,۲۴۲۴	۶۸,۴۳۱۲۳۱۰۱	-۱۷۳,۶۶۶۸۲۲۹	۶,۴۸۰۸	۱۱۳۸,۹	۲	
-۱۱۱,۳۷۸	-۱۷۷,۴۲۸	-۳۲۰,۴۴۹	۱۰۰,۷۰۰۹	۳۵,۴۷۱۴۲۴۸۴	-۸۲,۴۸۰۲۴۸۳۱	۶,۰۶۰۸۲۶۷	۱۱۲۲,۷	۳	
-۷۲,۰۹۲۷	-۲۷۶,۶۲۳	-۳۶۱,۹۲۶	۲۷۴,۶۹۳۴	۰,۴۰۶۳۹۷۴۴۴	-۲۶۵,۳۸۳۰۲۲۹	۶,۶۵۶۱۴۴۸	۱۱۱۹	۴	
-۱۳۴,۵۰۱	-۲۷۷,۰۴۱	-۳۲۷,۹۰۸	۱۶۱,۲۱۱۱	۳۴,۵۴۲۸۳۰۰۵	-۵۰,۶۱۱۸۷۲۲۶	۶,۶۲۲۱۳۸	۱۱۲۰,۳	۵	

جدول ژ - ۳: داده‌های الگوریتم ماهی الکتریکی

موقعیت نقاط کنترل (n)							زمان اجرا (ثانیه)	طول مسیر	شماره
۱۳۶,۰۱۵۷	-۱۰,۴۱۴۱	-۲۸۷,۷۲۸	۳۶۰,۳۰۸۷	۱۷۲,۱۱۳۴	-۰۷,۹۷۴۱	۳,۸۰۷۲۴۲۱	۱۱۲۸,۶	۱	
-۸۲,۶۲۱۷	-۱۰۷,۳۲۹	-۲۷۱,۸۴۹	۱۷۵,۴۷۴۰	۸۰,۳۱۴۴۰	-۳۸,۰۸۷۷	۳,۶۰۵۴۳۳	۱۱۰۳,۱	۲	
-۱۹۸,۸۰۲	-۳۱۱,۹۹۶	-۳۴۲,۷۰۸	۷۴,۲۰۰۹۷	-۱۰۷,۶۵۴	-۱۶۱,۶۵	۳,۶۶۱۸۵۴	۱۱۰۴,۸	۳	
-۱۱۰,۷۰۱	-۳۲۱,۳۴	-۴۰۹,۶۹۲	۳۱۰,۰۵۳	۱۰۷,۱۹۲۶	-۸۲,۳۴۳۷	۳,۶۱۰۵۷۶۴۰	۱۲۰۳,۴	۴	
-۲۶۲,۵۰۱	-۳۳۳,۲۰۸	-۴۰۲,۶۳۱	۵۷,۰۷۳۰۵	-۶۵,۱۱۴۴	-۱۹۰,۱۳۲	۳,۵۶۵۰۷۳۳	۱۱۱۴,۴	۵	

جدول ۷ - ۴: داده‌های الگوریتم کرم شبتاب

موقعیت نقاط کنترل (n)						زمان اجرا (ثانیه)	طول مسیر	شماره
-۹۸,۰,۱۴۶	-۲۵۰,۴۳۷	-۲۹۴,۱۹	۱۰۹,۹۲۶۱	-۱۷,۸۲۲۵	-۹۰,۴۷۷۸	۳۶,۲۵۸۳۱۱۲	۱۰۹,۵	۱
-۳۷,۸۸	-۲۳۸,۴۵۰	-۲۹۲,۵۶۶	۲۲۱,۶۷۶۶	-۴,۹۱۳۵۵	-۸۷,۵۴۱۹	۳۴,۷۴۴۴۱۷۹	۱۰۹,۵	۲
-۷۵,۳,۰۵۷	-۲۴۳,۵۰۰	-۲۹۳,۸۲۱	۱۹۴,۴۴۴۲	-۹,۳۷۷۴۴	-۸۹,۸۸۰۴	۴۱,۱۱۳۸۴۴۹	۱۰۹,۰	۳
۲,۱۳۴,۶۳۱	-۲۰,۴,۴۸۶	-۲۹۴,۲۷	۲۷۴,۱,۰۷۱	۳۷,۵,۸۴۳	-۸۲,۴۴۵۳	۴۷,۳۵۷۱۶۲	۱۱۰	۴
۷۰,۶۲۱۳۵	-۱۰,۸,۷۸۹	-۲۶۰,۷۵۱	۳۰,۱۳۰۴	۱۰۰,۲۴۰۲	-۳۱,۸۸۹۰	۵۰,۰,۸۷۱۸۶۱	۱۱۰,۲	۵

جدول ۷ - ۵: داده‌های الگوریتم امپریالیست رقابتی

موقعیت نقاط کنترل (n)						زمان اجرا (ثانیه)	طول مسیر	شماره
-۲,۴,۰,۸,۰,۴	-۱۴,۰,۳۷۰	-۲۶,۸,۶۲۷	۲۹۱,۶۸۹۲	۱۳۴,۰,۸۶۲	-۳۱,۹۷۸۶	۳,۰,۴,۰,۲۳۸۶	۱۱۰,۲,۳	۱
۱۱۷,۷۹,۰,۷	۱۹,۳۱۴۴۷	۱۱,۷,۷۳۱	۳۲۷,۷۴۰۴	۳,۷۵۷۱۱۱	-۱۱۲,۱	۴,۱۷۵۷۱۴۴	۱۱۶۰,۵	۲
۱۹۷,۷۸۲۵	۱۷۹,۲۱۲	۹۲,۴۴۷۶۷۲	۴۸۹,۹۷۹۷	۳۵۰,۱۰۷۲	-۷۹,۲,۹۹	۳,۷۶۱۲۶۴۶	۱۲۰,۷,۶	۳
۸۰,۱۲۵۴۱	-۸۴,۷۸۸۸	-۲۶۳,۹۱۱	۴۲۶,۲۴۹۴	۲۴۰,۸۸,۰۳	-۲۰,۰,۷۰۹	۳,۴۹۹۱۷۱۲	۱۱۱۱,۶	۴
۹۴,۴۹۷۰	-۵۶,۳۶۸	-۲۴۰,۴۱۶	۳۹۰,۰,۵۰۸۱	۲۲۹,۰,۵۹۴۲	۱۲,۳۶۳۹۳	۴,۵۳۶۱۳۵۰	۱۱۰,۲,۵	۵

۷- نتایج با بیست تکرار

جدول ۷ - ۶: داده‌های الگوریتم غذاخانه باکتریایی

موقعیت نقاط کنترل (n)						زمان اجرا (ثانیه)	طول مسیر	شماره
۱۹۴,۹۹۲	۲۰,۱,۰۰۵	۱۴۷,۴۹۴۶	۳۹۹,۰,۹۷۵	۳۷۲,۲۱۲۴	۷۴,۸۸۰۵۳۶	۲۸۴,۲۸۷۸	۱۱۷۲,۸,۰,۳	۱
۱۷۳,۱۴۷۹	۱۵۱,۳۵۲	۴۲,۰,۹۳۵۲	۱۴۴,۷۸۴۸	-۳,۵,۰,۰,۲۷	-۲۰,۶,۴۴۱	۳۱۹,۶۰۳۱	۱۲۲۹,۳	۲
-۷۹,۷۵۳۱	-۱۴۰,۰,۳۹۵	-۲۴۸,۷۸۶	۲۰,۲,۹۱۴۷	۱۲۲,۹۷۶۹	-۰,۳۳,۰,۷۱	۳۲۷,۲۰۱۹	۱۱۰,۲,۵۷۹	۳
-۷۸,۰,۰,۸۴	-۲۶۳,۰,۶۲	-۳۴۴,۵۷۳	۱۹۷,۷۷۸۷	-۲۷,۷۴۱	-۱۹۸,۲۱۶	۳۲۴,۴۸۷۶	۱۱۰,۱,۷۲۴	۴
-۲۹,۷۷۱۸	-۳۴۴,۳۳۵	-۴۱۳,۶۷۴	۳۶۱,۰,۵۰۸۱	۱۰,۸,۳۸۴۸	-۱۷۲,۰,۵۱۳	۳۲۶,۱۷۴۲	۱۱۹۹,۵	۵

جدول ۷ - ۷: داده‌های الگوریتم جستجوی فاخته

موقعیت نقاط کنترل (n)						زمان اجرا (ثانیه)	طول مسیر	شماره
-۱۴۱,۱۳	-۲۸۰,۹,۰۴	-۲۲۴,۹,۰۳	۳۰,۰,۵۱۴۷	۱۳۴,۰,۰,۸۴۹۴۳	۲۲۱,۷۹۷۹۰۰۴	۱,۴۶۴۹	۱۳۸,۰,۳	۱
-۱۳۹,۷۹۷	-۴۵۱,۰,۸۸	-۴۵۶,۰,۶۲	۳۱۶,۰,۵۴۷۸	۲۱۹,۰,۰,۲۸۲۸۸۷	-۱۱۱,۶,۲۲۲۳,۰,۴	۱,۳۸۱۴	۱۳۶۹,۸	۲
۵,۰,۸۶۶۷۱	-۵۶,۴۳۹۸	-۱۸۰,۲۲۶	۲۰,۹,۹,۰,۴	۱۴۹,۷۴,۷۸۳۲۷	۲۶,۰,۹۴۳۶۲,۰,۵	۱,۴۵۹۱	۱۳۶,۰,۱۹۳۷	۳
۲۲۰,۸۶,۰,۲	-۲۲,۹,۷۳۸	-۲۹,۰,۸۴۷	۳۱,۰,۹,۷۲۲	۲۴۱,۶۳۲۹,۰,۹۲	۴۹,۰,۸,۰,۸۲۲۷۱	۱,۲,۰,۲۷	۱۲۷۹,۶۶۷۸	۴
۵۱,۰,۵۴۱۸۶	۸۹,۳۵۲۷۸	۲۱۶,۹۱۳۴	۳۸,۰,۳۱۷۶	۳۰,۰,۰,۲۸۳۳۹۲	-۱۰,۴,۸۲۱۸۹۹۸	۱,۲۲۶۷	۱۴۱۲,۰,۱۴۸	۵

جدول ۷ - ۸: داده‌های ماهی الکترونیکی

موقعیت نقاط کنترل (n)						زمان اجرا (ثانیه)	طول مسیر	شماره
۱۷,۹۱۰,۰۹	۹۱,۴۹۹۱۸	۱۰,۶,۰۲۲۸	۱۸۵,۸۲۱	-۷,۴۷۴۷۳	-۲۷۶,۱۵۷	۰,۹۸۱۸۱	۱۴۰,۱,۸	۱
۵۷,۷۴۸۷۴	-۱۲۵,۸۱۵	-۴۱۱,۵۷۱	۲۳۹,۲۰۹۴	۴۱,۲۴۲۷۹	-۱۳۹,۵۴	۰,۸۸۷۸۸	۱۲۱,۱,۴	۲
۱۲۹,۰۷۰۵۲	-۷۷,۶۲۳۹	-۱۲۵,۷۶۸	۹۳,۷۵۰۳	-۸۲,۴۲۰۷	-۳۰,۶,۴۹۲	۰,۸۸۷۹۴	۱۲۴,۲	۳
۱۹۵,۰۲۲۴	۱۲۳,۸۷۳۵	-۷۹,۹۷۹۴	۱۳۱,۸۲۷۱	-۱۳۶,۶۶	-۲۳۴,۳۰۵	۰,۷۶	۱۲۴,۳,۱	۴
-۱۷۹,۰۳۸	-۳۱۶,۴۳۶	-۴۰,۶,۱۰۸	۷۹,۸۷۶۵۶	-۳۸,۱۸۳۱	-۶۴,۴۰۰۶	۰,۸۰,۶۰۷	۱۱۸۷	۵

جدول ۷ - ۹: داده‌های الگوریتم کرم شب تاب

موقعیت نقاط کنترل (n)						زمان اجرا (ثانیه)	طول مسیر	شماره
-۱۷۹,۷۹۷	-۲۹۳,۱۴۳	-۳۰,۸,۹۲۲	۱۸۹,۲۱۰۳	۱۳۳,۹۶,۰۷	۷۲,۰,۷۴۴۶	۶,۹۷۴۵	۱۱۶,۷,۸	۱
-۲۰,۴۳۴۵	-۱۵۹,۰,۶۳	-۳۱۹,۱۲۹	۲۶۴,۰,۰۱۹	۱۳۶,۴۱۰۷	۱۲,۱۶۷۱۲	۷,۱۹۷۶	۱۱۴,۳	۲
۱۰,۶,۰,۸۲۹	-۸۶,۴۱۷	-۲۸۰,۳۴۷	۴۰۹,۳۵۷	۱۷۳,۹۳۵۱	-۱۰,۲۷۴۲	۷,۲۶۲۷	۱۲۰,۲,۷	۳
-۹۴,۷۷۱۳	-۲۲۴,۴۸۰	-۲۳۰,۰,۵۷۷	۱۷,۸۴۰۲۷	۶۸,۳۹۷۳۴	-۲۰,۳,۰,۹	۷,۹,۹۹	۱۲۳,۹,۱	۴
-۳,۴۸۹۲۵	-۱۸۰,۶۲۶	-۲۹۳,۹۳۷	۳۶۳,۸۱۰۱	۱۹۷,۳۱۸۳	-۶۸,۰,۷۴۶	۷,۰,۹۰۲	۱۱۲۷,۷	۵

جدول ۷ - ۱۰: داده‌های الگوریتم امپریالیست رقابتی

موقعیت نقاط کنترل (n)						زمان اجرا (ثانیه)	طول مسیر	شماره
۱۷۱,۰,۰۳۵	۴۱,۳۲۹۳۵	-۲۴۲,۶۳۱	۳۹۰,۲۰,۹۷	۳۱۰,۰,۸۳۱	۰,۱۳۳۱۲۷	۰,۶۲۵۴۸	۱۱۲۴,۰,۵	۱
۱۲۹,۲۵۸۲	۱۲۶,۶۸۲۹	۱۸,۹۲۳۱۹	۱۷۳,۸۱۴۸	۱۳۶,۹۸۹	-۱۰۴,۷۷۶	۰,۷۱۶۸۲	۱۱۸۹,۰,۲	۲
۲۸۳,۳۵۹۶	۸۹,۳۰,۶۲۲	-۰۳,۰,۵۴۱	۲۲,۰,۷۱۴	-۳۳۰,۰,۰۴۴	-۴,۹,۲۰۹	۰,۷۴۱۹۷	۱۴۳۳,۰,۹	۳
۷۹,۶۲۹۴۸	-۱۰۱,۳۲۵	-۲۰,۰,۹۸۷	۰۱۲	۴۷۵,۸۲۹۷	۲۲۳,۲۷۱۸	۰,۵۹۷	۱۲۷۹,۰,۷	۴
-۱۳۴,۲۳	-۲۰,۰,۷۳۱	-۴,۰,۳,۶۶۴	۱۳۴,۲۱۱۸	۱۹۴,۰,۲۶۴	-۲۰,۰,۱۱۷	۰,۷۸,۰,۹۷	۱۴۲۹,۰,۷	۵

Abstract:

The optimization algorithms are classified into two categories: exact and approximate. The exact algorithm is able to find the exact solution, which is not efficient enough in difficult problems, and depending on the dimensions of the problem, its execution time increases, but the approximate algorithm is able to find the solution. It is close to optimal in short time for hard problems. Meta-heuristic algorithms are a type of stochastic algorithms that are used to find the optimal answer. In fact, meta-heuristic algorithms are one of the types of approximate optimization algorithms that have solutions for deviating from local optimal points and can be used in a wide range of problems. These algorithms are classified by different criteria, one of which is based on an answer and based on the population, which based on an answer changes an answer during the search process, and in the algorithm based on the population during the search, a population of answers is considered. Examples of these algorithms are electric fish optimization algorithm, firefly algorithm, etc. To compare these algorithms, there are single-objective or multi-objective test functions that check the convergence speed of the algorithm, the accuracy of the algorithm, the strength and robustness of the algorithm, and the overall performance of the algorithm, and one of these functions, the egg holder function, is a single test function. It is an objective function with several local minima. In this thesis, approaches based on meta-heuristic optimization algorithms have been proposed for the problem of mobile robot path planning. For this purpose, optimization algorithms are evaluated with the egg holder test function, and by comparing the results, the algorithms can be ranked and used for robot routing problems.

Keywords: Metaheuristic algorithms, Population-based, Robot pathing, Test function, Modeling



**Iran University of Science and Technology
Mechanical Engineering Department**

**A Comparative Study of the Performance of
Intelligent Algorithms: Cuckoo Search, Electric
Fish, Firefly, Bacterial Foraging, Imperialist
Competitive Algorithm in Path Planning**

Bachelor of Science Thesis in Mechanical Engineering

**By:
Parham Porkhial**

**Supervisor:
Dr. Khanmirza**