حل مسئله فروشنده دوره گرد با استفاده از ترکیب الگوریتم های ازدحام ذرات و شبیه سازی تبریدی

یکی ازمسائل مهم درتئوری گراف ها مسئله فروشنده دوره گرد tspمی باشد اکثرمسائلی که میتوان آنها را با مسئله فروشنده دوره گرد مدل کرد دارای مقیاس خیلی بزرگ هستند که الگوریتم های موجود قادر به حل آنها دریک زمان قابل قبول نیستند. درحل مسئله TSP باتوجه به اهمیت و کاربرد بسیارزیاد آن درحوزه های مختلف مهندسی به دنبال جایگشتی مناسب از n شهرمی باشیم این مسئله به عنوان مسئله NP-Hard شناخته شده است بررسی تمام راه حل ها باافزایش ابعاد مسئله عملاامکان پذیر نیست و نیاز داریم که ازروشهای سریع و کارا مانند روشهای هوش جمعی استفاده کنیم دراین تحقیق حل مسئله فروشنده دوره گرد با استفاده از الگوریتم ازدحام جمعیت و شبیه سازی تبریدی ارایه شده است

مقدمه :

مساله فرو شنده دوره گرد به عنوان یکی از مسائل مهم بهینه سازی با کاربردهای زیاد در دنیای واقعی از جمله مسائل مسیریابی و سایل نقلیه، طراحی تولید، زمان سرویس دهی به مشتریان و غیره همواره مورد توجه پژوهشگران قرار گرفته است و الگوریتم های زیادی برای آنالیز عملکرد خود در حل مسائل پیچیده، به حل این مساله در ادبیات موضوع پرداخته اند. هدف از حل این مسأله ، پیدا کردن کوتاهترین مسیری است که از مجموعه ای از شهرها (گره ها) عبور کرده ، بطوریکه هر شهر فقط یکبار ملاقات شود و سپس به شهر اولیه که از آن حرکت را شروع کرده است، برگردد. چالش اصلی در ارتباط با این مساله تعداد شهرها میباشد و در واقع با افزایش تعداد گره ها و در نتیجه ایجاد زیر تورها، عملا روشهای دقیق بهینه سازی قادر به حل این مساله در زمان منطقی نمی باشند و الگوریتمهای ابتکاری و فراابتکاری راهکارهای جهت رسیدن به جوابهای بهینه و یا قابل قبول در زمان منطقی ارائه میدهند .در مطالعات گذشته روشهای گوناگونی برای حل مساله TSP معرفی شده است که در این میان الگوریتم ژنتیک، بهینه سازی کلونی مورچگان و بهینه سازی ازدحام ذرات بیشترین تعداد مقالات را داشته اند. دو پژوهشگر به نام های رامان و گیل، عملکرد این سه الگوریتم را در حل مساله TSP با یکدیگر مقایسه نموده اند. اما در سال

2019 دو پژوهشـــگر دیگر به نام های حلیم و اســـماعیل در مقاله ای شـــش الگوریتم فراابتکاری ژنتیک، جســـتجوی ممنوعه، بهینه ســـازی کلونی مورچگان، بهینه ســـازی فیزیولوژی درخت، روش نزدیکترین هم سایگی و تبرید تدریجی را از لحاظ دقت، همگرایی و نتایج به د ست آمده با یکدیگر مقای سه کردند با توجه به کاربرد فراوان الگوریتم های الهام گرفته شده از طبیعت در حل م سائل بهی نه ســـــازی، محق قان همواره در حال معرفی الگوریتم های جد ید و یا بهبود الگوریتم های بهینهســـازی موجود هســـتند. در این گزارش یک الگوریتم جدید با اســـتفاده از ترکیب دو الگوریتم ازدحام ذرات و شبیه سازی تبریدی معرفی و کارایی آن در حل TSP بررسی میشود.

مساله فروشنده دوره گرد

با در نظر گرفتن گراف وزن دار کامل G ،مساله فروشنده دوره گرد شامل یافتن گراف همیلتونی با کمترین هزینه میبا شد .در واقع در این م ساله هر جایگ شت از گرههای گراف یک جواب م ساله و یک دور همیلتونی میباشد. فرمول ریاضی این مساله به صورت معادله زیر میباشد :

$$G = (V.E).whereas$$
$$c: E \to N\ and\ V = \{v_1.v_2.\ldots.v_n\}\ for\ n \in N - \{0\}$$

در معادله بالا برای هر ورودی (G.c) ، {vi1, vi2....vin} = M(G.c) است که { i1....in} یک جایگ شت از 1 تا (n دور همیلتونی) میبا شد. برای هر دور همیلتونی، تابع هدف به صورت کمینه کردن وزنهای یال های گراف به صورت معادله زیر تعریف میشود :

$$H = v_{i1}.v_{i2}.\ldots.v_{in} \in M(G.c)$$
$$Cost\{(v_1.v_2.\ldots.v_n).(G.c) = \sum_{j=1}^{n} c\left(v_{i_j}.v_{i_{(j\ mod\ n)+1}}\right)$$

در ادامه ابتدا به تشریح هر کدام از الگوریتم های ازدحام ذرات و شبیه سازی تبریدی پرداخته، و سپس به ساختار پروژه و پیاده سازی مسئله میپردازیم.

در ریاضیات و علوم کامپیوتر، مساله بهینه سازی(Optimization Problem) ، در واقع مساله پیدا کردن «بهترین» راه حل، از میان کلیه راه حل‌های «ممکن» برای مساله است. نوع خاصی از مسائل بهینه‌سازی وجود دارند که در آن‌ها، به دلیل زیاد شدن تعداد اشیا (Objects) منظور تعداد مشاهدات است، مدیریت کردن آن‌ها با استفاده از روش‌های ترکیبیاتی (Combinatorial Methods)امکان‌پذیر نیست. یک مثال شناخته شده از چنین مسائلی، مساله فروشنده دوره‌گرد (Traveling Salesman Problem)است که از دسته مسائل ان‌پی کامل NP-Complete میباشد.

برای حل چنین مسائلی، یک الگوریتم بسیار کاربردی به نام « شبیه سازی تبرید (Simulated Annealing) به آن «تبرید شبیه سازی شده»، «بازپخت شبیه سازی شده» و یا به طور خلاصه، SAنیز گفته می‌شود ،وجود دارد. از الگوریتم شبیه سازی تبرید اغلب برای تخمین «بهینه سراسری (Global Optimization) در مسائل بهینه سازی که فضای جستجوی آن‌ها بزرگ است، استفاده می‌شود.

همچنین، این الگوریتم معمولا هنگامی که فضای جستجو گسسته باشد، مورد استفاده قرار می‌گیرد (همه سفرهایی که از مجموعه مشخصی از شهرها طی آن‌ها عبور می‌شود). پیش از پرداختن به الگوریتم، مفهوم فرایند بازپخت انیلینگ (Annealing | در متالورژی (Metallurgy)که الگوریتم تبرید شبیه سازی شده در واقع تقلیدی از آن است، بیان خواهد شد. سپس، الگوریتم شبیه سازی تبرید شرح داده می شود و در نهایت به حل مساله فرو شنده دوره‌گرد با آن پرداخته می‌شود.

فرایند بازپخت یا انیلینگ

در متالورژی و علم مواد، بازپخت یا انیلینگ، به عملیات حرارتی گفته می‌شود که طی آن مشخصات فیزیکی و گاهی، مشخصات شیمیایی ماده تغییر می‌کند. این کار با هدف افزایش شکل‌پذیری و کاهش سختی ماده انجام می‌شود. طی این فرایند، ابتدا فلز گرم شده، سپس در یک درجه حرارت خاص نگه داشته و در نهایت، به تدریج سرد می‌شود. با گرم کردن فلز، مولکول‌ها آزادانه به هر سوی حرکت می‌کنند. با سرد کردن تدریجی ماده، این آزادی کاهش پیدا می‌کند. اگر فرایند سرد کردن به اندازه کافی کند باشد که بتوان اطمینان حاصل کرد فلز در هر مرحله در تعادل ترمودینامیکی قرار دارد، می‌توان مطمئن شد که انرژی گرمایی به طور یکنواخت در جسم توزیع شده و بهترین ساختار بلوری در آن وجود دارد که متقارن و مقاوم است. در الگوریتم شبیه‌سازی تبرید، از فرایند مذکور الگوبرداری شده است.

مساله فروشنده دوره‌گرد

در مساله «فروشنده دوره‌گرد(Travelling Salesman Problem | TSP) ، پرسش زیر مطرح می‌شود:

لیستی از شهرها و فاصله بین هر جفت از شهرها موجود است. کوتاه‌ترین مسیر ممکنی که با استفاده از آن بتوان از یک نقطه شروع کرد و پس از عبور از همه شهرها به نقطه اول بازگشت، کدام است؟

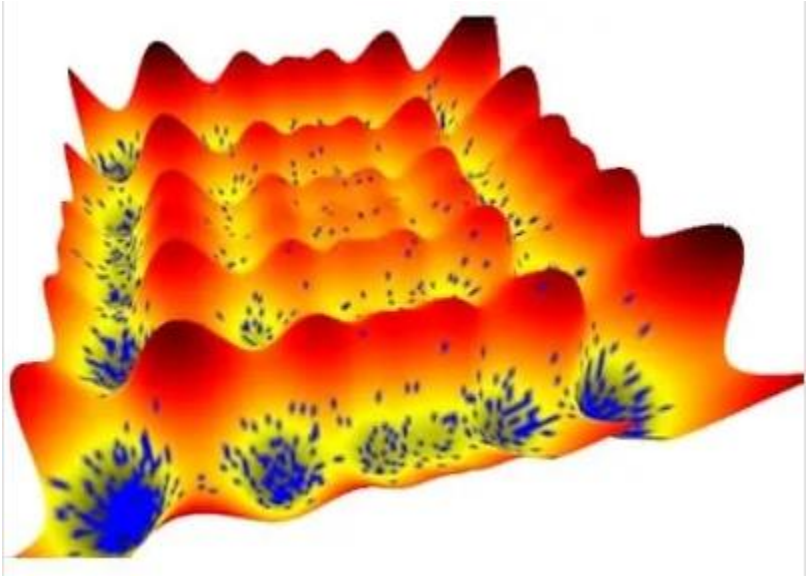
این مساله، یکی از مسائل «ان‌پی-سخت (NP-Hard) «در بهینه‌سازی کامپیوتری محسوب می شود که در حوزه «تحقیق در عملیات (Operational Research | OR) «و «علوم نظری رایانه (Theoretical Computer Science | TCS) حائز اهمیت ا ست. در نظریه پیچیدگی محا سباتی(Theory of Computational Complexity) ، نسخه تصمیم م ساله فرو شنده دوره‌گرد که در آن یک طول L داده شده، وظیفه تصمیم‌گیری پیرامون آن است که آیا گراف دارای دوری کمتر از L است یا خیر (از جمله مسائل ان‌پی-کامل محسوب می شود) بنابراین، این امکان وجود دارد که بدترین زمان اجرا برای هر الگوریتم برایTSP ، با افزایش تعداد شهرها به صورت «فوق چند جمله‌ای (Superpolynomially) «افزایش پیدا کند (اما نه بیش‌تر از نمایی).

این مساله برای اولین بار در سال ۱۹۳۰ فرموله شد و یکی از مسائلی است که بیشترین مطالعات در حوزه بهینه‌سازی روی آن انجام شده است. از این مساله، به عنوان بنچمارک (Benchmark) برای بسیاری از روش‌های بهینه‌سازی استفاده می‌شود. هرچند حل مساله فروشنده دوره‌گرد حتی به لحاظ کامپیوتری نیز دشوار است، تعداد زیادی از روش‌های تکاملی (Evolutionary Algorithms)و الگوریتم‌های دقیق (Exact Algorithm) برای حل این مساله شناخته شده هستند که می‌توانند TSP را با تعداد هزاران و حتی میلیون‌ها شهر حل کنند. مساله فروشنده دوره‌گرد را می‌توان به صورت یک مساله گراف و یا برنامه‌نویسی خطی صحیح (Integer Linear Programming)فرموله کرد. تعداد کل پاسخ‌های ممکن برای مساله برابر با $1/2*(n-1)!$ برای $n>2$ است که در آن، nتعداد کل شهرها است. در واقع، این عدد تعداد دورهای همیلتونی در یک گراف کامل با n راس را نشان می‌دهد.

الگوریتم شبیه سازی تبرید

همانطور که پیش‌تر بیان شد، در الگوریتم شبیه سازی تبرید (یا تبرید شبیه سازی شده) از فرایند بازپخت که از مباحث رشته متالورژی و مواد محسوب می شود، الگو گرفته شده است. انتخاب نام شبیه سازی تبرید برای این الگوریتم، ریشه در فرایند دارد که از آن تقلید می‌کند. در بهینه سازی

نیز مانند فرایند انیلینگ، آنچه در بخش پیشین پیرامون بازپخت مواد بیان شد، برای حل مسائل قابل انجام است. یعنی در واقع، جواب‌های یک مساله به خوبی گرم می شوند و با نوسانات زیادی تغییر می‌کنند؛ سپس، به تدریج دامنه تغییرات کم می شود و در واقع یک سری شیار به سمت جواب بهینه ساخته می شوند. الگوریتم شبیه سازی تبرید برای اولین بار در سال ۱۹۸۳، توسط «کریک‌پاتریک (Kirkpatrick) «و همکاران معرفی شد. شایان ذکر است، الگوریتم شبیه سازی تبرید از جمله الگوریتم‌های فراابتکاری) فراتکاملی | فرااکتشافی (Metaheuristic | محسوب می شود. در الگوریتم شبیه سازی تبرید، از روش احتمالاتی برای حل مساله بهینه سازی استفاده می‌شود.



در الگوریتم شبیه سازی تبرید(SA) ، نقطه s یک حالت از سیستم فیزیکی محسوب می شود و تابع E(s) مشابه با انرژی داخلی سیستم در حالت s است. هدف آن است که با شروع سیستم از یک حالت اولیه دلخواه (یک s_0 دلخواه)، به حالتی رسیده شود (S_n) که تابع E(s) در آن کمینه است. در واقع، با شروع از یک حالت دلخواه از سیستم فیزیکی، به حالتی رسیده می‌شود که انرژی داخلی سیستم در آن حالت کمینه است (سیستم کمترین انرژی را در آن حالت خواهد داشت). برای انجام این کار، الگوریتم از یک نقطه دلخواه آغاز به کار و سپس، یک حالت همسایه را انتخاب می‌کند. پس از آن، به طور احتمالی تصمیم می‌گیرد که در حالت کنونی بماند و یا به حالت همسایه جا به جا شود. مجموع این جا به جایی‌های احتمالی، سیستم را به سوی حالتی با انرژی

داخلی کمتر هدایت می‌کند. این کار تا جایی انجام می‌شود که سیستم به یک حالت عقلانی برسد یا اینکه میزان محاسبات، از یک آستانه مشخص بیشتر شود. برای درک بهتر وجه تمایز الگوریتم شبیه سازی تبرید با برخی از دیگر روش‌های ابتکاری مانند تپه‌نوری، مثال زیر قابل توجه است.

مساله فروشنده دوره‌گرد را می‌توان به عنوان مثالی در نظر گرفت که الگوریتم شبیه سازی تبرید در حل آن کاربرد دارد. همانطور که پیش‌تر بیان شد، در این مساله، فروشنده باید از تعداد زیادی از شهرها در حالی عبور کند که مسافت کل پیموده شده کمینه باشد. اگر فروشنده از یک مسیر تصادفی حرکت خود را آغاز کند، بعدا می‌تواند با این امید که در هر تغییر شهر مسافت پیموده شده را کاهش دهد، به ترتیب از کلیه شهرها عبور کند.

چالش اصلی در حل مساله فروشنده دوره گرد با استفاده از روشی مانند الگوریتم تپه‌نوردی (Hill Climbing)، از آنجا نشات می‌گیرد که این الگوریتم با جا به جایی بین همسایه‌ها معمولا به حالت کمینه می‌رسد و در همان نقطه متوقف می‌شود (در واقع در حالتی که نسبت به دیگر همسایگی‌های بررسی شده کمینه باشد متوقف می شود؛ در صورتی که امکان دارد یک کمینه سراسری وجود داشته باشد). در واقع، الگوریتم تپه‌نوردی «کمینه محلی (Local Minimum)» را معمولا به سرعت پیدا می‌کند، اما ممکن است در همان جا متوقف شود و بنابراین از آنجا به کمینه سراسری (Global Minimum) نمی‌رسد. این در حالی است که شبیه سازی تبرید می‌تواند راه حل خیلی خوبی را حتی در حضور «نویز (Noise)» برای چنین مساله‌ای پیدا کند.

راهکار الگوریتم شبیه سازی تبرید برای غلبه بر این مشکل و بهبود استراتژی مذکور، استفاده از دو ترفند است. ترفند اول، از الگوریتم متروپولیس (Metropolis Algorithm) گرفته شده که در سال ۱۹۵۳ توسط متروپولیس و همکاران اون کشف شده است. در این الگوریتم، گاهی مسیرهای کوتاه پذیرفته نمی‌شوند، زیرا این عدم پذیرش منجر به آن می‌شود که الگوریتم فضای راه حل ممکن بزرگ‌تری را اکتشاف کند. ترفند دوم، مجددا با تقلید از فرایند بازپخت فلز و کاهش دما به دمای پایین‌تر اتفاق می‌افتد. پس از آنکه بررسی‌ها و جا به جایی‌های زیادی انجام شد و مشاهده شد که تابع E(s) این تابع در واقع همان تابع هزینه یا Cost Function است به آرامی کاهش

پیدا می‌کند (دما کاهش پیدا می‌کند)، اندازه انجام جا به جایی‌های «بد» کنترل می‌شـــود. پس از چندین بار کاهش دما به مقدار کم‌تر، فرایند «فرونشانی (Quench) «با پذیرش جا به جایی‌های خوب به منظور پیدا کردن کمینه محلی تابع هزینه اتفاق می‌افتد. زمان‌بندی های بازپخت (Annealing Schedules)متعددی برای کاهش درجه حرارت وجود دارد، اما نتایج به طور کلی خیلی به جزئیات حساس نیستند.

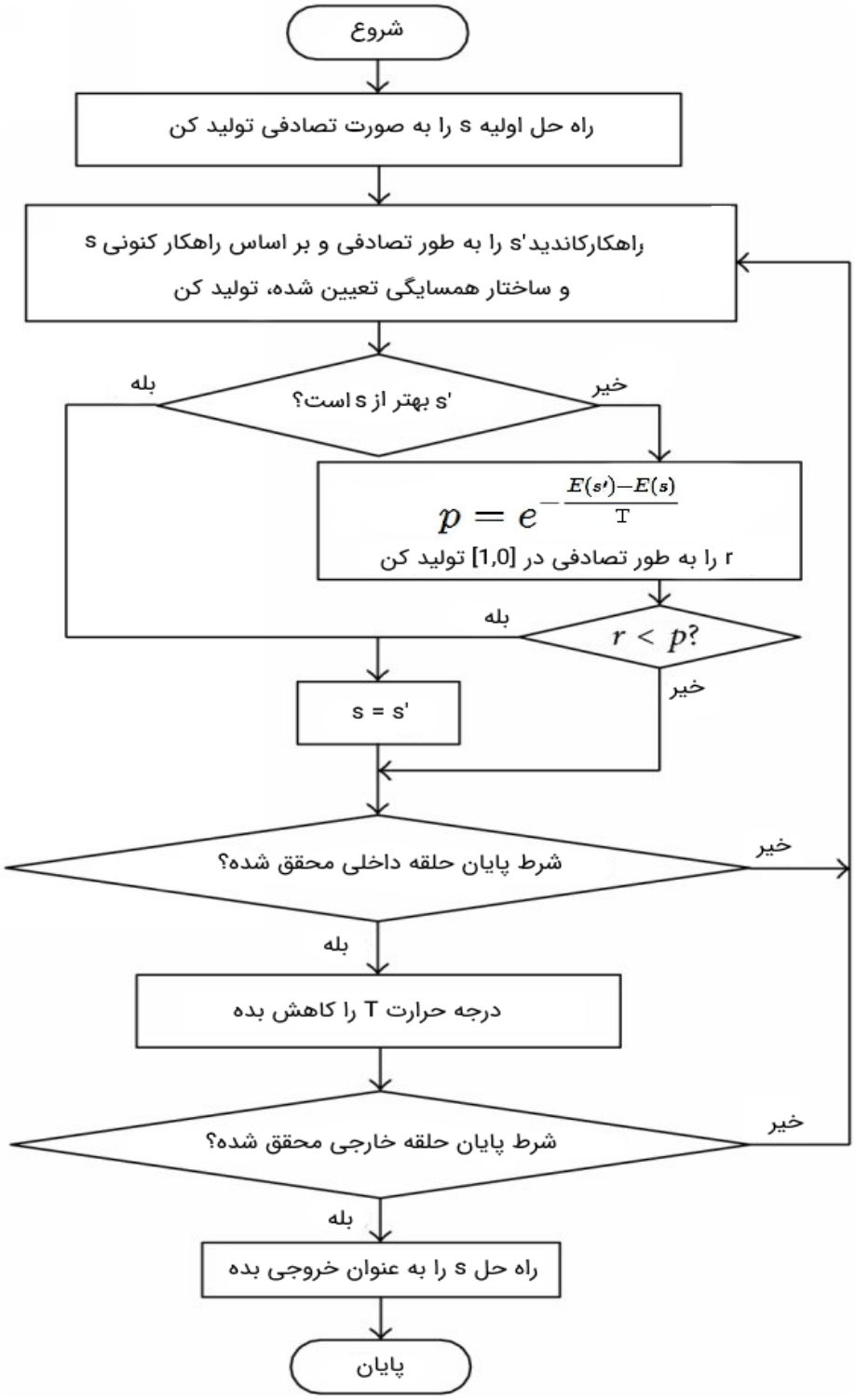
اکنون، الگوریتم شبیه سازی تبرید به صورت ریاضیاتی و دقیق ارائه می‌شود. احتمال انتقال از یک حالت فعلی، مثلا s به حالت کاندید جدید مانند s′ با یک تابع احتمال پذیرش P(e,e′,T)مشخص می‌شود که در آن، e=E(s)و e′=E(s′) است) چنانچه پیش‌تر بیان شد، تابع E در فضای حالت نشان‌گر انرژی داخلی سیستم و T نشان‌گر دما است. دمای T بر اساس زمان تغییر می‌کند . بنابراین، با توجه به آنکه پیش‌تر گفته شـد هدف آن اسـت که انرژی سـیسـتم کمینه باشـد، پس حالتی که منظور s است در آن E(s) کمتر باشد، بهتر از حالتی است که در آن مقدار E(s)بیشتر باشد. نکته قابل توجه در الگوریتم تبرید شبیه سازی شده آن است که تابع احتمال P همواره و حتی در شرایطی که e کوچک‌تر از e′ است، باید مثبت باشد. این خصوصیت موجب می شود الگوریتم در بهینه محلی که نسـبت به بهینه سـراسـری «بدتر» اسـت، متوقف نشـود. در واقع، در صـــورتی که s′بهتر از E(s)≥E(s′) باشـــد،s′ پذیرفته می‌شـــود و اگر s′ بدتر از s باشـــد و موجب E(s)<E(s′)شود، s′با یک احتمال پذیرفته می‌شود. تابع احتمال P به صورت زیر است:

$$p = e^{-\frac{E(s\prime)-E(s)}{T}} = e^{-\frac{\Delta f}{T}}$$

در صورت کاهش دمای T و میل کردن آن به سمت صفر، احتمال P نیز باید کاهش پیدا کند و یا به صـــفر و یا یک عدد مثبت میل کند. در این شـــرایط، Pهنگامی به ســـمت صـــفر میل می‌کند که e<e′ باشد و در صورتی به یک عدد مثبت میل می‌کند که e′<e باشد. همانطور که مشهود

است، دما در کنترل تغییرات سیستم نقش اساسی دارد. همانطور که پیش‌تر هم بیان شد، دما در شبیه‌سازی به صورت تدریجی کاهش پیدا می‌کند. بنابراین، الگوریتم از $T=\infty$ و به عبارتی، از یک دمای بسیار بزرگ کار خود را آغاز می‌کند و در هر مرحله با توجه به یک زمان‌بندی تبرید از پیش مشخص شده، کاهش پیدا می‌کند. زمان‌بندی تبرید با توجه به این مو ضوع انجام می شود که اگر منابع مورد استفاده (مثلا میزان محاسبات) به پایان برسند، زمان انجام فرایند نیز به پایان می‌رسد. از همین رو، الگوریتم ابتدا در فضای بزرگی از راه حل‌ها، صرف نظر از انرژی داخلی سیستم به دنبال پاسخ می‌گردد و به تدریج، به سمت مناطق دارای انرژی کمتر جا به جا می‌شود. این منطقه، به تدریج کوچک‌تر می‌شود و این کار، تا زمانی که بهینه سراسری یافته شود، ادامه پیدا می‌کند. برای یک مساله دارای فضای متناهی، افزایش زمان موجب می‌شود تا با احتمال یک، الگوریتم در نقطه بهینه سراسری متوقف شود؛ هر چند زمان لازم برای انجام این کار، بیش‌تر از زمان لازم برای جستجو در کل فضا است و بنابراین، این موضوع در عمل کاربردی ندارد.

فلوچارت الگوریتم شبیه سازی تبرید



الگوریتم بهینه سازی ازدحام ذرات(PSO)

در اوایل سال ۱۹۹۰ میلادی، پژوهش‌های گوناگونی پیرامون رفتار اجتماعی گروه‌های حیوانات انجام شد. این پژوهش‌ها حاکی از آن بودند که برخی از حیوانات که به یک گروه خاص متعلق هستند، مانند پرندگان، ماهی‌ها و دیگر موارد، قادر به آن هستند که اطلاعات را در گروه‌های (دسته‌های | گله‌های) خود شان به اشتراک بگذارند و چنین قابلیتی به این حیوانات مزایای قابل توجهی برای بقا اعطا می‌کرد.

با الهام گرفتن از این مطالعات، «کندی (Kennedy) و ابِرهارت (Eberhart) در سال ۱۹۹۵ الگوریتم بهینه سازی ازدحام ذرات (Particle Swarm Optimization | PSO) یا الگوریتم PSOرا در یک مقاله معرفی کردند. الگوریتم بهینه سازی ازدحام ذرات یا الگوریتم PSO یک الگوریتم فراابتکاری (Metaheuristic) است که برای بهینه‌سازی توابع پیوسته غیر خطی منا سب محسوب می شود. نویسندگان مقاله مذکور، الگوریتم بهینه سازی ازدحام ذرات یا الگوریتم PSO را از مفهوم هوش ذرات (Swarm Intelligence) که معمولا در گروه های حیوانات مانند گله‌ها و دسته‌های حیوانات وجود دارد الهام گرفته و ساخته‌اند.

برای شفاف شدن هر چه بیشتر ساز و کار کلی الگوریتم بهینه سازی ازدحام ذرات و دیگر الگوریتم‌هایی که از رفتار گروهی حیوانات الهام گرفته شده‌اند، توضیحاتی پیرامون رفتار گروهی (گله‌ای) حیوانات ارائه می‌شود. این توضیحات می‌تواند به درک چگونگی ساخت الگوریتم بهینه سازی ازدحام ذرات (و دیگر الگوریتم‌های دارای رویکرد مشابه) برای حل مسائل پیچیده ریاضی کمک کند.

الگوریتم بهینه سازی ازدحام ذرات و رفتار گروهی حیوانات

دسته پرندگانی (گروه پرندگان | ازدحام پرندگان) که بر فراز یک منطقه در حال حرکت هستند، باید یک نقطه را برای فرود پیدا کنند. در این حالت، تعریف اینکه همه پرندگان در کدام نقطه باید فرود بیایند، مسئله پیچیده‌ای است. زیرا پاسخ این مسئله، وابسته به موضوعات مختلفی یعنی بیشینه کردن منابع غذایی در دسترس و کمینه کردن خطر وجود شکارچیان است در نقطه محل

فرود است. در این شرایط، ناظر می‌تواند حرکت پرندگان را به صورت رقص‌پردازی ببیند. پرندگان به طور هم‌زمان در یک برهه از زمان حرکت می‌کنند تا بهترین محل برای فرود آمدن تعیین شود و همه دسته (گروه) به طور هم‌زمان فرود بیایند.

در مثال بیان شده پیرامون حرکت ازدحامی پرندگان و فرود هم‌زمان آن‌ها، اعضای دسته پرندگان (گروه پرندگان) یا همان ازدحام پرندگان، امکان به اشتراک‌گذاری اطلاعات با یکدیگر را دارند. در صورتی که پرندگان امکان به اشتراک‌گذاری اطلاعات با یکدیگر را در گروه‌های خود شان نداشته باشند، هر پرنده‌ای از گروه (دسته) در محل (نقطه) و در زمان متفاوتی فرود می‌آید.

پژوهش‌هایی که از سال ۱۹۹۰ پیرامون رفتار پرندگان انجام شد، حاکی از آن است که همه پرندگان یک ازدحام (گروه | دسته) که به دنبال نقطه خوبی برای فرود هستند، قادر به آن هستند که از بهترین نقطه برای فرود در هنگامی که آن نقطه توسط یکی از اعضای ازدحام پیدا شد، آگاه شوند. با استفاده از این آگاهی، هر یک از اعضای این ازدحام، تجربه دانش شخصی و ازدحامی خود را متوازن می‌کنند که با عنوان دانش اجتماعی (Social Knowledge) شناخته شده است.

شایان ذکر است که معیارهایی که برای ارزیابی خوب یا نامناسب بودن یک نقطه برای فرود مورد بررسی قرار می‌گیرند، شرایط بقایی هستند که در یک نقطه، برای بقا وجود خواهند داشت. از جمله این موارد، بیشینه بودن منابع غذایی و کمینه بودن خطر وجود شکارچیان است که پیش‌تر نیز به آن‌ها اشاره شد. مسئله پیدا کردن بهترین نقطه برای فرود، یک مسئله بهینه‌سازی محسوب می‌شود. گروه، ازدحام یا گله باید بهترین نقطه فرود، برای مثال طول و عرض جغرافیایی را، به منظور بیشینه کردن شرایط بقای اعضای خود تعیین کند.

برای انجام این کار، هر پرنده‌ای ضمن پرواز، به جستجوی نقطه مناسب فرود می‌پردازد و نقاط مختلف را از جهت معیارهای بقای گوناگون مورد ارزیابی قرار می‌دهد تا بهترین منطقه برای فرود

را پیدا کند و این کار تا زمانی انجام می‌شـــود که بهترین منطقه برای فرود، توســـط کل ازدحام مشخص شود.

کندی و اِبرهارت، از رفتار جمعی پرندگان الهام گرفتند؛ رفتاری که مزایای بقای قابل توجهی را برای پرندگان در هنگام جســـتجو برای یک نقطه امن برای فرود تضـــمین می‌کرد. آن‌ها بر همین اســـــاس، الگوریتمی را ارائـه کردنـد کـه الگوریتم ازدحـام ذرات (Particle Swarm Optimization)نامیده می‌شـــود. الگوریتم PSO می‌تواند رفتاری به مثابه آنچه برای دســـته پرندگان گفته شد را تقلید کند.

الگوریتم بهینه سازی ازدحام ذرات کلاسیک

نسخه اولیه الگوریتم ازدحام ذرات یا الگوریتم PSO که با عنوان نسخه کلاسیک این الگوریتم نیز شناخته شده است، در سال ۱۹۹۵ ارائه شد. از آن زمان تاکنون، انواع دیگری از این الگوریتم به عنوان نسخه‌های دیگر الگوریتم کلاسیک ارائه شده‌اند که از جمله آن‌ها می‌توان به کاهش خطی وزن اینرســـی(Linear-Decreasing Inertia Weight) ، وزن عـامـل انقبـاض (The Constriction Factor Weight)و اینرســـی پویا (Dynamic Inertia) در کنار مدل‌های ترکیبی یا حتی روش‌های بهینه سازی الهام گرفته شده از کوانتوم که روی الگوریتم PSO اعمال شـــده‌اند اشـــاره کرد .هدف از مســـائل بهینه‌ســـازی، تعیین متغیری اســـت که با بردار $X=[x_1x_2x_3...x_n]$نشان داده می‌شود و بسته به فرمول بهینه‌سازی ارائه شده توسط تابع$f(X)$ ، بیشینه یا کمینه می‌شود. بردار متغیرX ، به عنوان یک بردار مثبت شناخته شده است. این بردار، یک مدل متغیر و بردار n بُعدی آن را نمایش می‌دهد که در آن، nنشانگر تعداد متغیرهایی است که ممکن است در مسئله تعیین شوند n .در مسئله پیدا کردن بهترین نقطه برای فرود دسته پرندگان، طول و عرض جغرافیایی است.

از ســـوی دیگر، تابع f(X) تابع برازش (Fitness Function) یا تابع هدف (Objective Function)نامیده می‌شـود و تابعی اسـت که میزان خوب یا بد بودن یک موقعیت X را ارزیابی می‌کند. این تابع برای مسئله دسته پرندگان، میزان خوب بودن یک نقطه برای فرود است که پرنده پس از پیدا کردن یک نقطه به آن فکر می‌کند. چنین ارزیابی برای مســـئله فرود گروه پرندگان، برا ساس معیارهای بقای گوناگون انجام می شود. اکنون، ازدحامی با P ذره در نظر گرفته می شود؛ یک بردار مکان Xitو یک بردار ســرعت Vitدر هر تکرار برای هر یک از i ذره‌ای این ســرعت را ایجاد می‌کنند، به صورت زیر وجود دارد:

$$X_i^t = (x_{i1} x_{i2} x_{i3} \dots x_{in})^T$$

$$V_i^t = (v_{i1} v_{i2} v_{i3} \dots v_{in})^T$$

این بردارها بر اساس بُعد j مطابق با معادله‌ای که در ادامه آمده است، به روز رسانی می‌شون

$$V_{ij}^{t+1} = wV_{ij}^t + c_1 r_1^t (pbest_{ij} - X_{ij}^t) + c_2 r_2^t (gbest_j - X_{ij}^t)$$

و

$$X_{ij}^{t+1} = X_{ij}^t + V_{ij}^{t+1}$$

که در آن‌ها، داریم:

$i = ۱, ۲, \dots, P$ و $j = ۱, ۲, \dots, n.$

معادله اول نشانگر آن است که سه عامل مختلف در حرکت ذرات در یک تکرار، نقش‌آفرین هستند. بنابراین، سه عبارت در این رابطه وجود دارد که بعدا مورد بررسی قرار خواهند گرفت. در عین حال، معادله دوم، موقعیت ذرات را به‌روزرسانی می‌کند. پارامتر w ثابت وزن اینرسی است و برای نسخه کلاسیکPSO ، این مقدار یک مقدار مثبت ثابت است. در نسخه کلاسیکPSO ، مقدار پارامتر w مثبت است. این پارامتر برای متوازن کردن جستجوی سراسری حائز اهمیت است که به آن اکتشاف (هنگامی که مقادیر بالاتری تنظیم شده‌اند) و جستجوی محلی (وقتی مقادیر کم‌تری تنظیم شده‌اند) نیز گفته می‌شود. یکی از مهم‌ترین تفاوت‌های الگوریتم PSO کلاسیک با دیگر نسخه‌های مشتق شده از این الگوریتم، پارامتر w است. سرعتی که اولین عبارت در معادله را به روزرسانی می‌کند، ضرب داخلی پارامتر w و سرعت پیشین ذره است. به همین دلیل است که حرکت پیشین ذره به حرکت کنونی نمایش داده می‌شود. از همین رو، برای مثال، اگر $w = 1$ بود، حرکت ذره به طور کامل به وسیله حرکت قبلی خودش تحت تاثیر قرار گرفته است؛ بنابراین، ذره ممکن است به حرکت خود در همان جهت ادامه دهد.

از سوی دیگر، اگر $0 \leq w < 1$، این تاثیر کاهش پیدا می‌کند و این یعنی ذرات به منطقه دیگری در ناحیه جستجو می‌روند. بنابراین، با توجه به کاهش پارامتر وزن اینرسی، ازدحام (گروه | دسته) ممکن است نواحی بیشتری را در ناحیه جستجو مورد اکتشاف قرار دهد و این یعنی شانس پیدا کردن بهینه سراسری افزایش پیدا می‌کند. اگرچه، در حالاتی که از مقادیر w کم‌تر استفاده می‌شود نیز هزینه‌ای وجود دارد که شبیه‌سازی‌ها را زمان‌برتر خواهد کرد.

عبارت درک فردی که دومین عبارت در معادله یک است، به وسیله تفاضل بین بهترین موقعیت خود ذره، برای مثال pbestij(ij)و موقعیت کنونی آن Xtijمحاسبه می شود. شایان توجه است که ایده نهفته در پس این ایده آن است که هر چه فعالیت ها فاصله بیشتری از موقعیت pbestij(ij)بگیرند، تفاضل (pbestijij−Xtij) باید افزایش پیدا کند. بنابراین، این عبارت افزایش پیدا کرده و ذره را به بهترین موقعیت آن جذب می‌کند. پارامتر c1که به صورت

حاصل‌ضرب در این رابطه وجود دارد، یک ثابت مثبت و یک پارامتر شناخت فردی محسوب می‌شود و به اهمیت تجربیات پیشین خود ذره وزن می‌دهد.

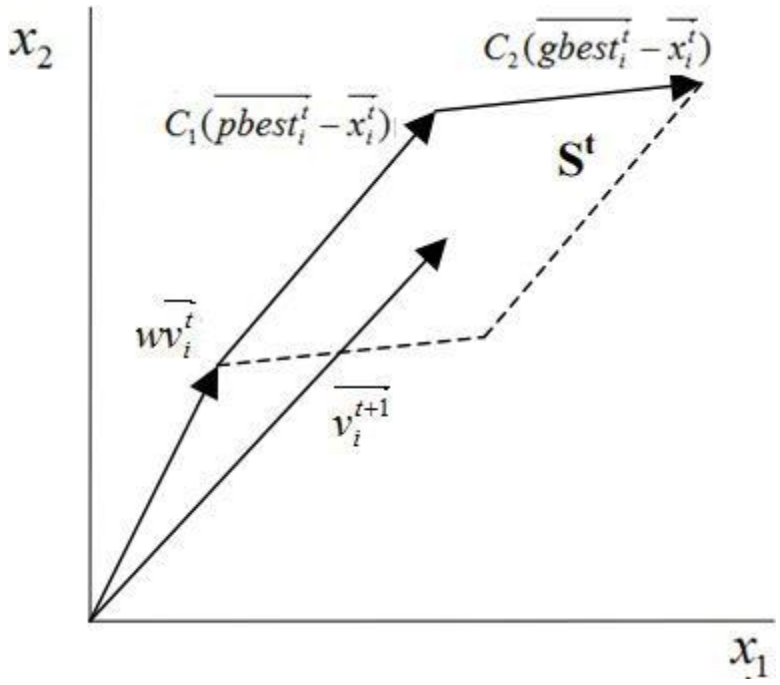
دیگر پارامتری که ضرب عبارت دوم را شکل می‌دهد، عبارت r1است .r1. یک پارامتر مقدار تصادفی با طیف [0,1] است. این پارامتر تصادفی، نقش مهمی را بازی می‌کند، زیرا از همگرایی پارامترها ممانعت و بهینه سراسری احتمالی را بیشینه می‌کند. در نهایت، سومین عبارت مربوط به یادگیری اجتماعی است. به دلیل وجود این پارامتر، همه ذرات در ازدحام قادر به آن هستند که اطلاعات پیرامون بهترین نقطه به دست آمده را صرف نظر از اینکه کدام ذره آن را پیدا کرده است، با یکدیگر به اشتراک بگذارند؛ برای مثال gbestijij . فرمت این عبارت نیز درست مانند دومین عبارت است که مربوط به یادگیری فردی می‌شود. بنابراین، تفاضل (gbestijij−X(t)ij) مانند یک جاذبه برای ذرات برای بهترین نقطه تا هنگام پیدا شدن نقطه در تکرار t عمل می‌کند. به طور مشابه، c2پارامتر یادگیری اجتماعی و وزن آن، اهمیت یادگیری سراسری ذرات است. همچنین، r2نیز نقشی مشابه با r1دارد.

در ادامه، الگوریتم PSO ارائه شده است و افراد ممکن است متوجه منطق بهینه سازی موجود در جستجوهای آن برای کمینه‌ها شوند و همه بردارهای مکانی که توسط تابع f(X) ارزیابی می‌شوند. تابع f(X) با عنوان «تابع برازش (Fitness Function) شناخته شده است. در تصاویر زیر نیز به روز رسانی‌هایی در سرعت ذرات و موقعیت آن در تکرار t با در نظر داشتن مسئله دوبُعدی با متغیرهای x1و x2انجام شده است.

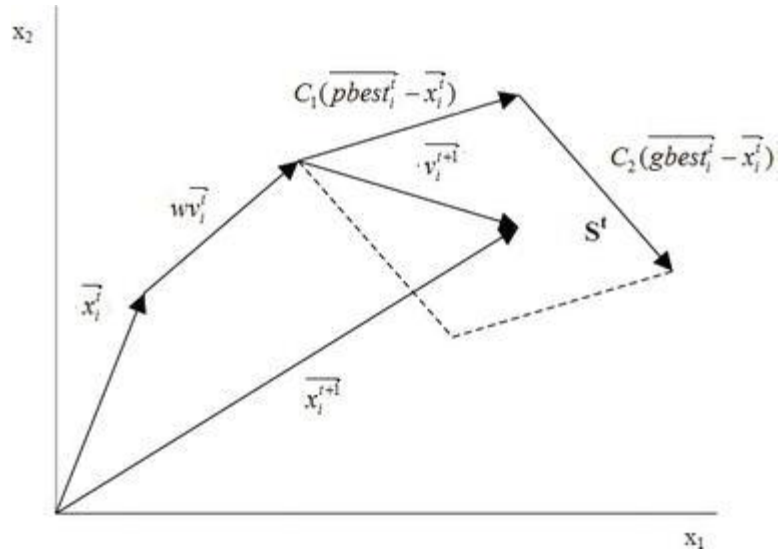
1. مقداردهی اولیه
   - برای هر i در جمعیت ازدحام با اندازه:p
     - Xiرا به طور تصادفی مقداردهی اولیه کن.
     - $$$$x_{i}$V_{i}$$را به طور تصادفی مقداردهی اولیه کن.
     - تابع برازش f(Xi) را ارزیابی کن.
     - Pbestij(ij)را با یک کپی از Xiمقداردهی اولیه کن.

- gbestرا با یک نسخه از Xiبا بهترین برازش مقداردهی اولیه کن.

2. مراحل را تا هنگامی که یک معیار توقف ارضا شود، تکرار کن:
   - برای هر ذره:i
     - Vtiو Xtiرا مطابق با معادلات ۱ و ۲ مقداردهی اولیه کن.
     - تابع برازش f(Xti)را ارزیابی کن.
     - اگر f(pbest_{i})<f(X_i^t)pbesti←Xit
     - اگر f(gbest)<f(X_i^t)gbest←Xit h’v



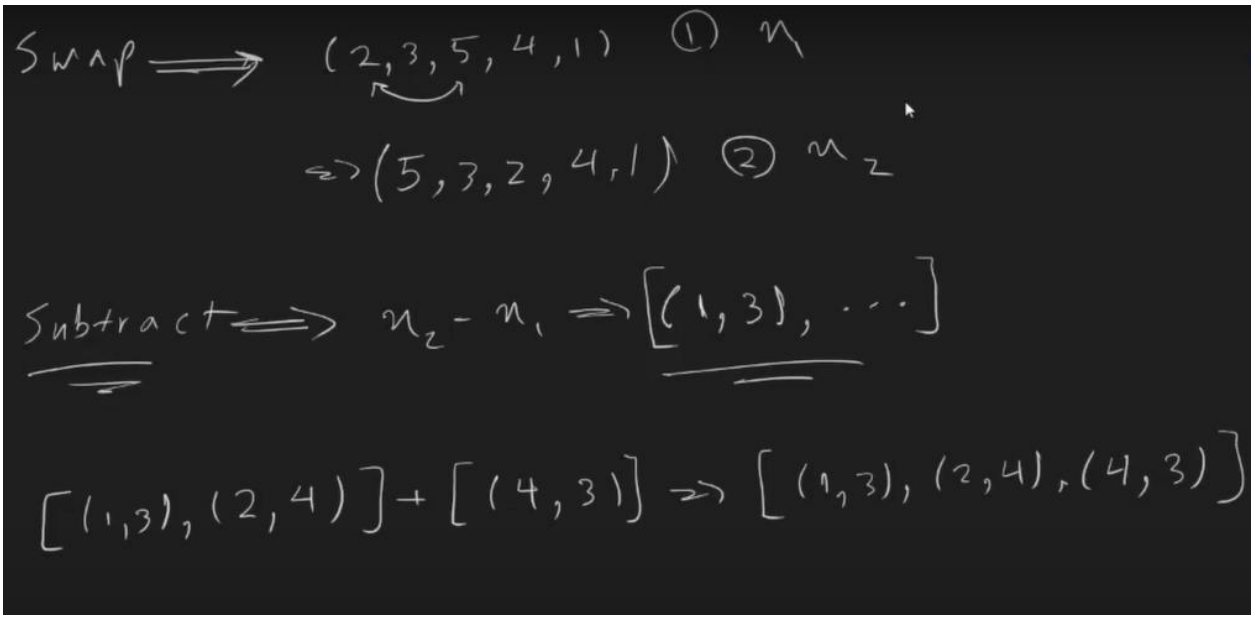
بردار سرعت در تکرار t به صورتی که به وسیله دو مولفه ترکیب شده با ارجاع به یک مسئله دوبُعدی است.



این بردار مکانی در تکرار t به روز ر سانی شده، در حالیکه به و سیله دو مولفه با ارجاع به م سئله دوبُعدی ترکیب شده است.

در مورد روش کار :

گفتیم که الگوریتم pso از دســـته الگوریتم های هوش جمعی میباشـــد که در آن، یک جمعیت اولیه از جواب ها را داریم که با روش های رندم یا یک الگوریتمی، مقدار دهی اولیه شـــده اند و با ارسال سیگنال ها به یکدیگر و با اشتراک یک هوش اجتماعی، به جواب بهینه نزدیک میشوند. این الگوریتم از ابتدا هدف آن ، بهینه سازی توابع در فضای پیوسته بوده و و برای مسئله فروشنده دوره گرد که دارای یک فضای گسسته از جواب ها میباشد، باید تغییر کند.



همانطور که در شکل بالا نیز نشان داده شده است :

در واقع علمگر swap به این صورت هست که در دنباله، جای دو آیتم را با یکدیگر عوض میکنیم.

عملگر subtract شامل لیستی از swap هایی هست که انجام شده تا x1 به x2 تبدیل شود.

علمگر add بین دو دنباله نیز، اجتماع لیست ها میباشد.

در الگوریتم زیر، که مربوط به اگلوریتم ازدحام ذرات میباشد،

$$v^i[t+1] = w v^i[t] + c_1 r_1 \left(x^{i,best}[t] - x^i[t]\right) + c_2 r_2 \left(x^{gbest}[t] - x^i[t]\right)$$

در واقع این W اضافه شده به این دلیل که V یک حرکت تصادفی هست، یعنی هر چه که جلوتر میرویم بهتر است این حرکت تصادفی کمتر شود و حرکت به سمت GBEST و PBEST بهتر شود. پس ما در اینجا یک ترکیبی بین PSO و SA داریم که با یک ضریب دمایی (W) ، به مرور زمان، تاثیر این حرکت تصادفی را کمتر میکنیم.

swap.m

```
function res = swap(node_sequence, swap)
    temp = node_sequence(swap(1));
    node_sequence(swap(1)) = node_sequence(swap(2));
    node_sequence(swap(2)) = temp;
    res = node_sequence;
    clear temp;
end

```

در فایل بالا، علمکر swap تعریف شده که یک دنباله به این تابع داده میشود(که یک ماتریس 1*2 میباشد) و دو ایندکس مشخص شده از این دنباله را با یکدیگر جابجا میکند.

subtract.m

```
function res = subtract(node_sequence1, node_sequence2)
    res = [];
    for i=1:length(node_sequence1)
        for j=i:length(node_sequence1)
            if node_sequence2(i) == node_sequence1(j) && i ~= j
                res = [res, [i; j]];
                node_sequence1 = swap(node_sequence1, res(:, end));
            end
        end
    end
end
```

و در کد بالا، تابع subtract را میبینیم. که با اســـتفاده از دو حلقه for تو در تو، دو دنباله را میگیرد و swap های مورد نیاز را پیدا میکند و در لیســتی به نام res ذخیره میکند و این لی ست را بر میگرداند.

TSPcostFunction.m

```
function cost = TSPcostFunction(node_sequence, node_positions)
    cost = 0;
    for i = 1:length(node_sequence) - 1
        cost = cost + measureDist(node_positions(node_sequence(i), :), ...
            node_positions(node_sequence(i + 1), :));
    end

end

function dist = measureDist(a, b)
    dist = sqrt((a(1) - b(1))^2 + (a(2) - b(2))^2);
end
```

در فایل بالا نیز، تابع cost دیده می شود که معیار م سئله ما میبا شد که با ا ستفاده از یک تابع به نام measuredist که برای اندازه گیری فاصله دو شهر میباشد، فاصله شهرهای متوالی را که در داخل دنباله مشــخص شــده ، اندازه میگیرد و با هم جمع میزند و به عنوان هزینه برمیگرداند و ما به این وسیله، یک ارزش گزاری برای آن particle که یک دنباله دارد، مشخص میکنیم.

usage.m

```
clc;
clear;
close all;

program.citiesNumber = 15;
program.PositionRange = [0 100];

params.MaxIt = 50;
params.nPop  = 20;
params.showPlot = 1;
params.showIters = 1;
% program
% params

out = pso(program, params);
figure;
plot(out.BestCost, 'LineWidth', 2)
% xlabel('iteration');
% ylabel('BestCost');

```

در فایل بالا نیز یک مثال استفاده از الگوریتم نوشته شده است که همانطور که دیده میشود متغیرهای تعداد شهرها، موقعیت جغرافیایی شهرها، و همچنین پارامتر های مسئله مانند تعداد کل گردش هایی که مسئله ما باید انجام بدهد و تعداد جمعیت نیز در این فایل تنظیم میشود. و پارامتر دیگری به نام showplot که میخواهیم در هر مرحله ، موقعیت شهر ها در یک گراف به ما نشان داده شود و پارامتر showiters که اطلاعات هر دوره گردش را برای ما چاپ میکند را با مقدار 1 مشخص کرده ام که به این معنی است این دو متغیر به همراه best cost را نیز در هر مرحله برای ما چاپ میکند.

در فایل pso.md نیز که الگوریتم ازدحام نوشته شده است. که تابع اصلی ما تابع pso میباشد که مسئله و پارامترها را میگیرد و یکسری پارامتر ها نیز داخل همین تابع مشخص میشود . در واقع پارامتر های زیر در داخل این فایل مشخص میشود :

```
% PSO Parameters
w = .9;
wdamp = .95;
alpha = .85; % Personal Learning Coefficient
beta = .85; % Global Learning Coefficient
```

پارامتر wdamp همان پارامتری ا ست که در واقع الگوریتم های pso را با SA ترکیب میکند. در واقع در هر تکرار، مقدار wdamp در w ضرب میشود و مقدار حاصل به عنوان w جدید به مسئله داده میشود.

سپس به بخش initialization مسئله میرسیم که ساختار کلی particle ها را مشخص میکنیم که به صورت زیر میباشد .

```
%% Initialization

empty_particle.citySequence = [];
empty_particle.Cost = [];
empty_particle.Velocity = [];
empty_particle.Best.citySequence = [];
empty_particle.Best.Cost = [];

particle = repmat(empty_particle, nPop, 1);

GlobalBest.Cost = inf;

% initialize city positions
cityPositions = unifrnd(PositionMin, PositionMax, PositionSize);

% initialize city positions (for test porpuse with 15 cities and PosiotionRange of 0 100)
```

که یک ر شته از توالی شهرها، هزینه آن ها، وکتور سرعت و یک ذره best هم خواهند دا شت که نشان دهنده بهترین حالتی است که آن ذره در طول مسیر الگوریتم مشاهده کرده.

در کد، مقدار اولیه globalbest را نیز بی نهایت میگذاریم، به این دلیل که مسئله بهینه سازی ما، هدفش یافتن مینیمم هست و این یعنی اولین باری که الگوریتم شروع به کار میکند، اولین ذره

ای که بیاید، با هر مقدار خطایی، انتخاب میشـود. به این دلیل که هر مقداری که داشـته باشـد از مقدار بینهایت کمتر است.

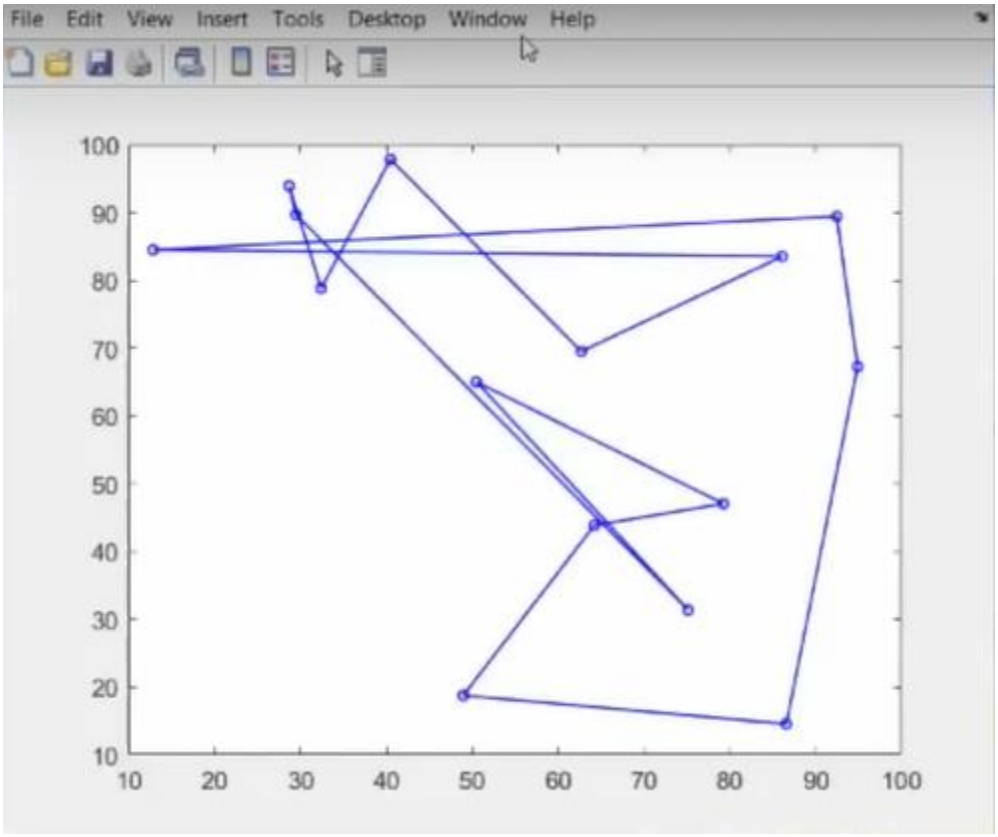
اگر مسـئله ما ماکزیمم میبود، باید این مقدار را منفی بینهایت میگذاشـتیم. در نهایت با تابع unifrnd یک مقدار اولیه برای موقعیت شـهرها در نظر میگیریم که در بازه positionmin و positionmax این اعداد را تولید میکند. که به عنوان مثال یک مورد را در کد قرار داده ام.

```
% initialize city positions (for test porpuse with 15 cities and PosiotionRange of 0 100)
  cityPositions = [
     9.5297   47.2452;
    58.4346   91.4046;
    39.8273   47.2617;
    96.5833   40.5779;
    11.8282   60.6019;
    13.0960   10.6280;
    20.5225   77.0525;
    93.3953   77.1152;
    60.8161   82.1863;
    83.0912   59.4440;
    95.1951    6.8667;
    73.4111   34.8132;
     9.1560   88.5590;
     2.6903   86.0422;
    69.6881   12.0456;
 ];
```

سپس یک مقدار اولیه برای سایر پارامترهای مسئله قرار میدهیم.

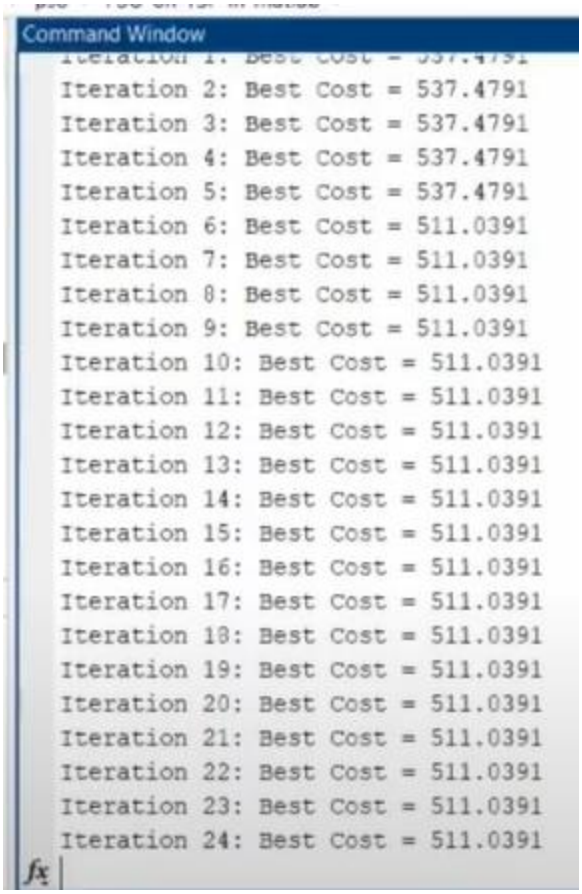
```
for i = 1:nPop

    % Initialize city sequence
    particle(i).citySequence = randperm(citiesNumber);

    % Initialize Velocity
    particle(i).Velocity = randi(citiesNumber, [2, ceil(citiesNumber/2)]);
```

در نهایت با اجرا کردن فایل usage.m ، چند تا از خروجی های کد را در این جا مشاهده میکنید



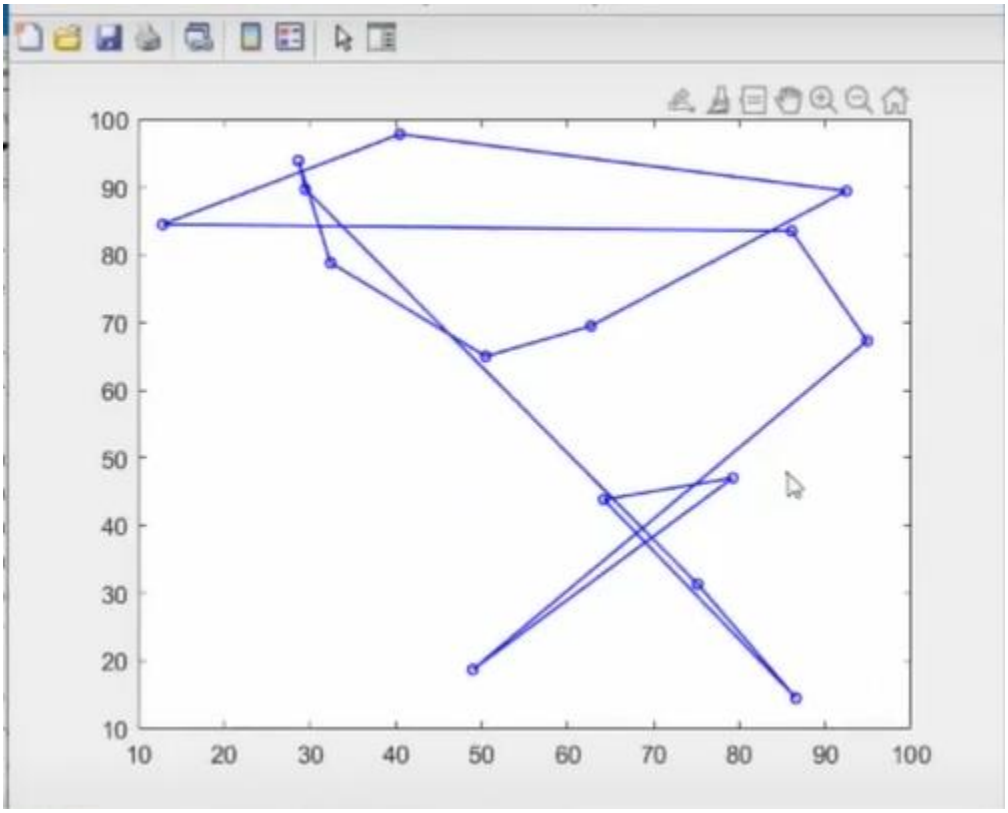

```
Command Window
  Iteration 1: Best Cost = 537.4791
  Iteration 2: Best Cost = 537.4791
  Iteration 3: Best Cost = 537.4791
fx
```

و به همین صورت کد در حال اجرا، ادامه پیدا میکند



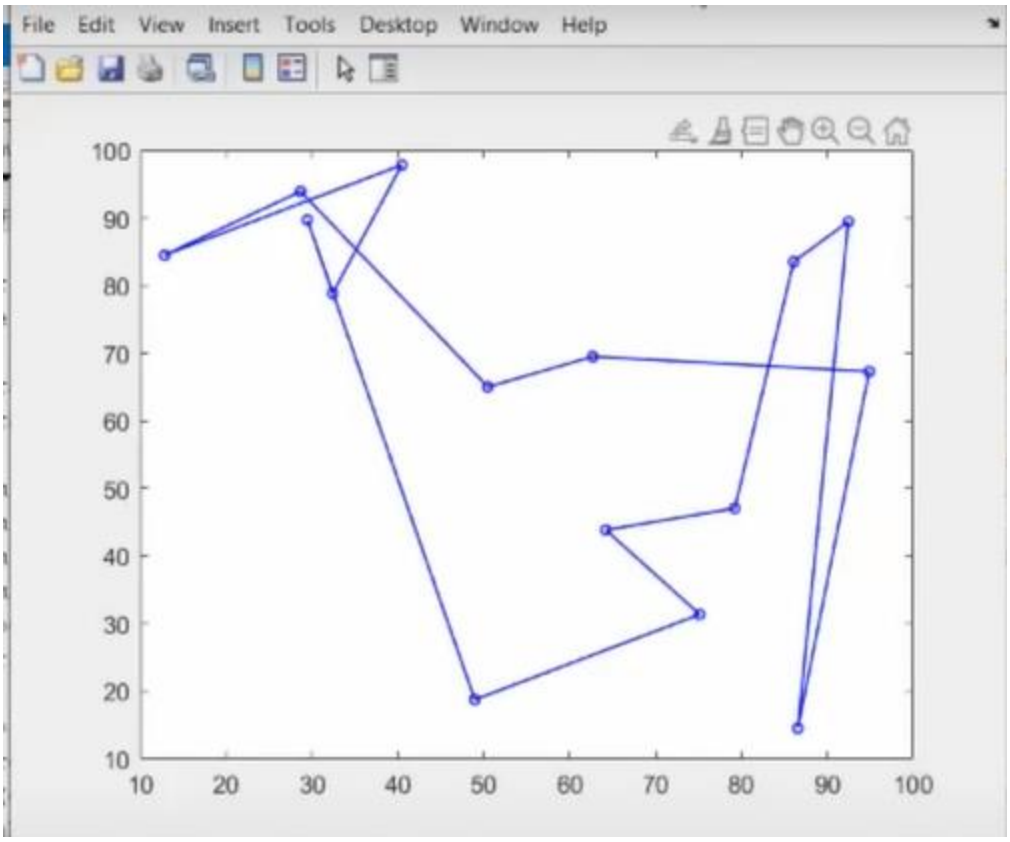
تا زمانی که iteration های ما به 50 برسد

و هر بار که مقدار بهتری به عنوان global best پیدا میکند، این مقدار را در نمودار ترسیم میکند



و در واقع مشاهده میکنیم که به مقدار بهینه نزدیک و نزدیک تر میشود.

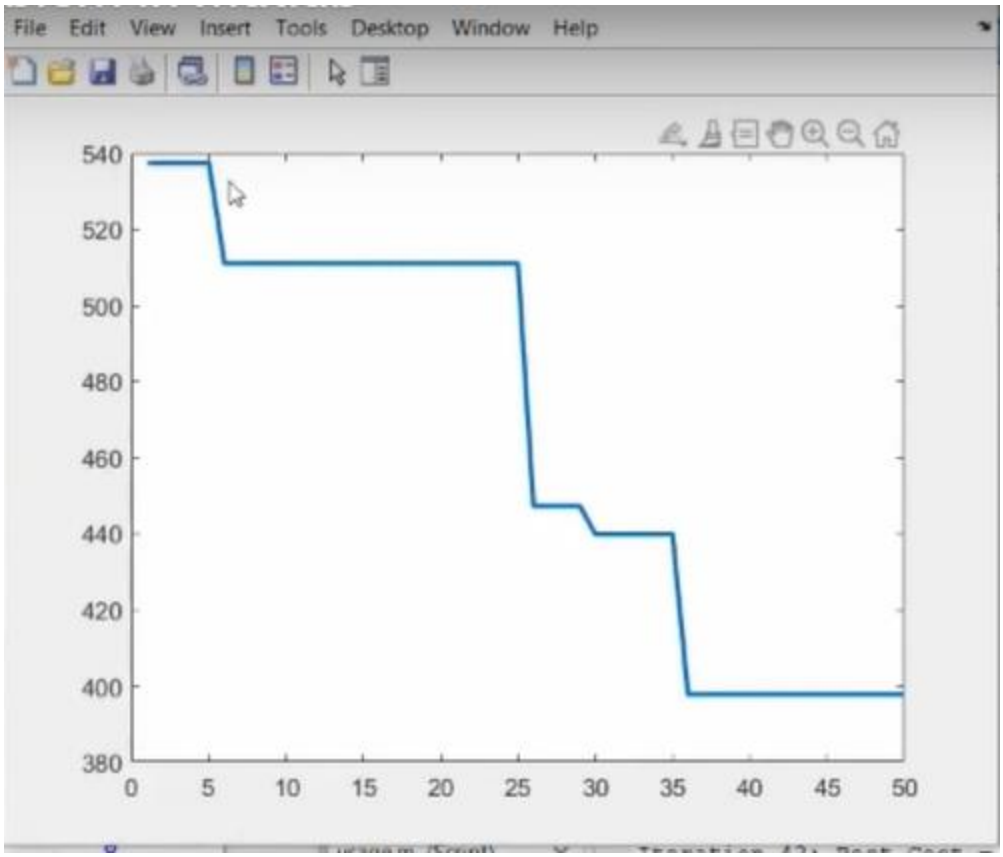
```
Iteration 45: Best Cost = 397.7646
Iteration 46: Best Cost = 397.7646
Iteration 47: Best Cost = 397.7646
Iteration 48: Best Cost = 397.7646
```



در واقع بهترین مقدار ما مقدار 397.7646 میباشد

```
Iteration 50: Best Cost = 397.7646
```

و تابع نزولی ارزش بهترین particle ما و یا global best ما به صورت زیر میباشد



و در نهایت مقاله ای که در ســـاخت این پروژه به من کمک کرد و ایده ی اولیه را از آن گرفتم، از لینک زیر قابل دانلود می باشد.

[Solving City Routing Issue with Particle Swarm Optimization]

با تشکر