دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی
گرایش نرم‌افزار

عنوان

پیاده سازی و مقایسه عملکرد الگوریتم‌های ژنتیک و سردکردن تدریجی برای حل مساله‌ی تعمیم‌یافته‌ی فروشنده‌ی دوره‌گرد

نگارش
علی مرتضوی

استاد راهنما
دکتر محمد رضا رزازی

آبان ۹۶

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی
گرایش نرم‌افزار

عنوان

پیاده سازی و مقایسه عملکرد الگوریتم‌های ژنتیک و سردکردن تدریجی برای حل مساله‌ی تعمیم‌یافته‌ی فروشنده‌ی دوره‌گرد

نگارش
علی مرتضوی

استاد راهنما
دکتر محمد رضا رزازی

آبان ۹۶

به نام خدا

تعهدنامه اصالت اثر

تاریخ:

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

اینجانب علی مرتضوی متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

علی مرتضوی

امضا

تقدیر و تشکر

خدا را شاکرم که به من توفیق داد تا بتوانم در راه شناخت جهان پیرامونم تلاش کنم.

از استاد گرامی جناب آقای دکتر محمدرضا رزازی که در انتخاب و پیش‌برد این پروژه به عنوان استاد پروژه، کمک‌های فراوانی به این جانب داشتند، کمال تشکر را دارم.

همچنین از قبول مسئولیت داوری این گزارش توسط جناب آقای دکتر سیدرسول موسوی و جناب آقای دکتر علیرضا باقری بسیار سپاس‌گزارم.

همچنین از جناب آقای مهندس مدرسی که در درک و فهم بهتر مساله و همچنین پیدا کردن راه‌حل‌های مناسب، به من کمک کردند کمال سپاس را دارم.

چکیده

در نظریه پیچیدگی محاسباتی، یک مساله تصمیم‌گیری ، ان‌پی تمام است؛ هرگاه در مجموعه ان‌پی باشد و همچنین در مجموعه ان‌پی‌سخت باشد. تا کنون برای حل هیچ کدام از این مسائل، راه حل با زمان چند جمله‌ای پیدا نشده‌است. همچنین اثبات شده‌است، در صورتی که برای هرکدام از این مسائل راه حل با زمان چند جمله‌ای پیدا شود، برای بقیه‌ی این مسائل نیز، راه‌حل چند جمله‌ای خواهیم داشت. با توجه به کاربردی بودن این‌گونه از مسائل، حل آن‌ها اهمیت فراوانی دارد. بدلیل سخت بودن این دسته از مسائل، برای پیدا کردن جواب، از رویکردهای متنوعی برای حل استفاده شده‌است.

یکی از رویکردهای حل این‌گونه مسائل استفاده از الگوریتم‌های اکتشافی است. در این پروژه، ما به حل یکی از این مسائل با استفاده از دو روش الگوریتم ژنتیک و سردکردن تدریجی پرداختیم و در انتها نتایج این دو روش را با هم مقایسه کردیم. مساله مورد بررسی در این پروژه، مساله فروشنده‌ی دوره‌گرد در محیط آسیب‌دیده است که حل آن از لحاظ کاربردی دارای اهمیت است.

در این پروژه، ابتدا ویژگی‌های مساله مورد نظر و ارتباط این مساله، با مسائل مشابه مورد بررسی قرار گرفت. سپس الگوریتم ژنتیک و سردکردن تدریجی را متناسب با این ویژگی‌ها طراحی کردیم. همچنین از روش برنامه ریزی خطی صحیح، برای مقایسه جواب بدست آمده توسط روش الگوریتم ژنتیک و سردکردن تدریجی با جواب بهینه استفاده کردیم. در نهایت عملکرد این دو روش را بررسی کردیم.

واژه‌های کلیدی:

الگوریتم ژنتیک، الگوریتم سردکردن تدریجی، فروشنده‌ی دوره‌گرد در محیط آسیب دیده، الگوریتم‌های اکتشافی، مسائل ان‌پی تمام

فهرست مطالب

صفحه

فهرست اشکال

صفحه

فهرست جداول

صفحه

فصل اول

مقدمه

۱- مقدمه

با بوجود آمدن کامپیوترهای دیجیتال و صنعتی شدن آن، امکان محاسبات به روش کامپیوتری فراهم شد. در نتیجه محققین به تلاش در استفاده از کامپیوترها برای بهبود بازدهی سیستم‌های مختلف پرداختند. در ابتدا مسائل اولیه کاربردی به روش های الگوریتمی حل شدند و این امر باعث بوجود آمدن تحولات بزرگی در زمینه‌های مختلفی همچون صنعت شد. این تحولات باعث افزایش محبوبیت کامپیوترها شد. محققین سعی در حل کردن مسائل کاربردی به روش‌های الگوریتمی کردند. در این شرایط، محققین با مسائلی مواجه شدند که حل کردن آن‌ها به روش های متعارف نیازمند زمان بسیار زیاد بود. به همین علت، مفهوم پیچیدگی مسائل مطرح شد. و مشخص شد دسته ای از مسائل دارای پیچیدگی بیشتری هستند.

به بیان دقیق‌تر در نظریه پیچیدگی محاسباتی[۱]، یک مساله تصمیم‌گیری[۲] ، ان‌پی تمام[۳] است؛ هرگاه در مجموعه ان‌پی[۴] باشد و همچنین در مجموعه ان‌پی‌سخت[۵] باشد. تا کنون برای حل هیچ کدام از این مسائل، راه حل با زمان چند جمله‌ای پیدا نشده است. همچنین اثبات شده است که در صورتی که برای هرکدام از این مسائل راه حل با زمان چند جمله‌ای پیدا شود، برای بقیه‌ی این مسائل نیز راه‌حل چند جمله‌ای خواهیم داشت.

با توجه به کاربرد این‌گونه از مسائل، حل آن‌ها اهمیت فراوانی دارد. از این رو در مواجهه با این مسائل از شیوه‌های زیر استفاده می‌کنند:

روش‌های قطعی: در این روش‌ها، ما به صورت دقیق به جستجو برای پاسخ بهینه می‌پردازیم. این‌گونه از راه‌حل‌ها برای ورودی‌های کوچک قابل انجام است. اما به ازای ورودی‌های بزرگ، حل آن‌ها بعلت نیاز به زمان زیاد ناممکن است.

روش‌های تقریبی: در این روش‌ها، به جای جستجو برای یافتن جواب بهینه، به جستجو برای جوابی که حداکثر یک نسبت ثابت از جواب بهینه است می‌پردازند.

---

[1] Computational Complexity Theory

[2] Decision Problem

[3] Nondeterministic Polynomial Time Complete

[4] NP (Nondeterministic Polynomial)

[5] NP-Hard

محدودکردن ورودی: با اضافه کردن فرض‌هایی به مساله اولیه، از روش های سریع‌تر برای حل مساله استفاده می‌کنند. (برای مثال در مساله کوله پشتی[۶]، با فرض اینکه اعداد طبیعی و در بازه محدودی هستند، راه‌حل با مرتبه چند جمله‌ای با استفاده از برنامه‌نویسی پویا[۷] وجود دارد.)

روش‌های اکتشافی: در این روش‌ها، الگوریتم‌هایی ارائه می‌شود که در بسیاری از حالت‌ها، نسبتا سریع هستند و به صورت آماری عملکرد نسبتا مطلوبی دارند اما اثباتی برای اینکه جواب‌هایشان لزوما بهینه و سرعت اجرایشان لزوما مطلوب است، وجود ندارد. در برخی از این جستجوها، برای یافتن جواب از اعداد تصادفی استفاده می‌کنند. در این روش‌ها، ممکن است به احتمال کمی، الگوریتم جواب بهینه را پیدا نکند. یکی از این روش‌ها استفاده از الگوریتم‌های تکاملی همچون ژنتیک[۸] است. روش دیگر، استفاده از سردکردن تدریجی[۹] برای یافتن جواب بهینه است.

در این پروژه ما قصد داریم به بررسی عملکرد دو روش اکتشافی ژنتیک و سردکردن تدریجی برای حل یکی از مسائل دشوار بپردازیم.

۱-۱ تعریف مساله

مساله مورد بررسی در این پروژه، مساله تعمیم یافته‌ای از مساله فروشنده‌ی دوره‌گرد[۱۰] است. ابتدا مساله فروشنده‌ی دوره‌گرد را تعریف می‌کنیم:

۱-۱-۱ مساله‌ی فروشنده‌ی دوره‌گرد:

مساله‌ی فروشنده دوره‌گرد در دسته مسائل ان‌پی‌سخت قرار می‌گیرد و به این صورت تعریف می‌شود:

ورودی: شامل یک مجموعه از شهرها و فاصله بین هر دو شهر است.

خروجی: دور همیلتنی[۱۱] که مجموع هزینه روی یال‌های آن کمینه باشد.

---

[6] Knapsack Problem

[7] Dynamic Programming

[8] Genetic Algorithms

[9] Simulated Annealing

[10] Travelling Salesman Problem

[11] Hamiltonian Cycle

در حالت تصمیم‌گیری این مساله از ما پرسیده می‌شود آیا دوری به صورت فوق وجود دارد که طول آن کم‌تر از L باشد، و در این حالت خروجی مساله، بله یا خیر خواهد بود.

۱-۱-۲ مساله‌ی تعمیم‌یافته فروشنده‌ی دوره‌گرد:

مساله‌ی فروشنده‌ی دوره‌گرد تعمیم‌یافته که می‌توان آن را به نام مساله‌ی مسیریابی خودرو در محیط آسیب‌دیده[۱۲] نامید به صورت زیر تعریف می‌شود:

فرض کنید برف جاده‌های بین شهرها را پوشانده‌است. و ما قصد داریم با شروع از یک شهر، با استفاده از دستگاه برف‌روب، تعدادی از مسیرهای بین شهرها را باز کنیم و سپس به شهر اولیه بازگردیم. به نحوی که بین هر دو شهر یک مسیر باز شده وجود داشته‌باشد. با توجه به اینکه در ابتدا مسیرها برفی هستند و پس از یک‌بار عبور دستگاه برف‌روب از این مسیر، مسیر بین دو شهر از برف خالی می‌شود؛ هزینه عبور از هر مسیر برای بار اول با هزینه عبور از همان مسیر برای بارهای بعدی متفاوت است. در نتیجه، به ازای هر جفت شهر، دو هزینه خواهیم داشت. با این فرض، مساله به صورت زیر تعریف می‌شود:

ورودی: شامل یک مجموعه از شهرها و هزینه‌های عبور در حالت برفی و غیربرفی بین هر جفت شهر است. هزینه عبور در هر دوجهت یکسان است.

خروجی: کوتاه ترین گشتی[۱۳] که با شروع از شهر اولیه، حداقل یک‌بار به تمام شهرها وارد شود و در نهایت به شهر اولیه بازگردد. در این حالت، گشت‌هایی که از یک راس چند بار عبور می‌کنند مجاز هستند. همچنین مجاز هستیم که از یک یال چند بار عبور کنیم.

در فصل بعدی به مفاهیم لازم برای درک بهتر این مساله می‌پردازیم. در فصل سوم سعی در پیدا کردن ویژگی‌های لازم برای حل بهتر مساله می‌پردازیم. در فصل چهارم به توضیح الگوریتم‌های مورد استفاده

دوری که با شروع از یک راس دقیقا یک بار به هر راس برود و در نهایت به راس اولیه بازگردد.

[12] Disaster Vehicle Routing Problem (DVRP)

[13] Walk

می‌پردازیم. در فصل پنج، به بررسی فرایند تولید و نتایج بدست آمده می‌پردازیم. و در انتها نتایج صورت گرفته را جمع‌بندی می‌کنیم.

فصل دوم
مفاهیم پایه

۲- مفاهیم پایه

در این فصل به بررسی مفاهیم پایه در خصوص مساله‌ی فروشنده‌ی دوره‌گرد و مساله‌ی فروشنده‌ی دوره‌گرد تعمیم‌یافته می‌پردازیم.

۲-۱ مساله‌ی فروشنده‌ی دوره‌گرد

مساله مورد بررسی در این پروژه، مساله‌ی تعمیم یافته‌ای از مساله‌ی فروشنده‌ی دوره‌گرد است. ابتدا مساله‌ی فروشنده‌ی دوره‌گرد را تعریف می‌کنیم:

مساله‌ی فروشنده دوره‌گرد در دسته مسائل ان‌پی‌سخت قرار می‌گیرد و به این صورت تعریف می‌شود:

- **ورودی:** شامل یک مجموعه از شهرها و فاصله بین هر دو شهر است.
- **خروجی:** دور همیلتنی که مجموع هزینه روی یال‌های آن کمینه باشد.

در حالت تصمیم‌گیری این مساله از ما پرسیده می‌شود آیا دوری به صورت فوق وجود دارد که طول آن کم‌تر از L باشد، و در این حالت خروجی مساله، بله یا خیر خواهد بود.

این مساله اولین بار در سال ۱۹۳۰ فرمول‌بندی شده است. و یکی از مسائلی است که به صورت گسترده در مسائل بهینه‌سازی[۱۴] مورد مطالعه قرار گرفته‌است. این مساله بعنوان محک[۱۵] برای بررسی کارایی روش‌های مختلف بهینه‌سازی استفاده شده است. با اینکه این مساله، مساله دشواری محسوب می‌شود، تعداد زیادی از روش‌های شهودی و دقیق برای این مساله معرفی شده‌اند.

این مساله کاربرد های زیادی در بخش‌های مختلف از جمله برنامه‌ریزی[۱۶]و علم منطق[۱۷] و تولید چیپ‌های کوچک[۱۸] دارد. و با تغییراتی اندک، به یک زیرمساله مهم در حوزه‌های مختلف تبدیل می‌شود. بعنوان مثال یکی از کاربردهای آن در ترتیب‌دهی دی‌ان‌ای [۱۹] می‌باشد.

---

[14] Optimization Problems

[15] Benchmark

[16] Planning

[17] Logistics

[18] Microchips

[19] DNA Sequencing

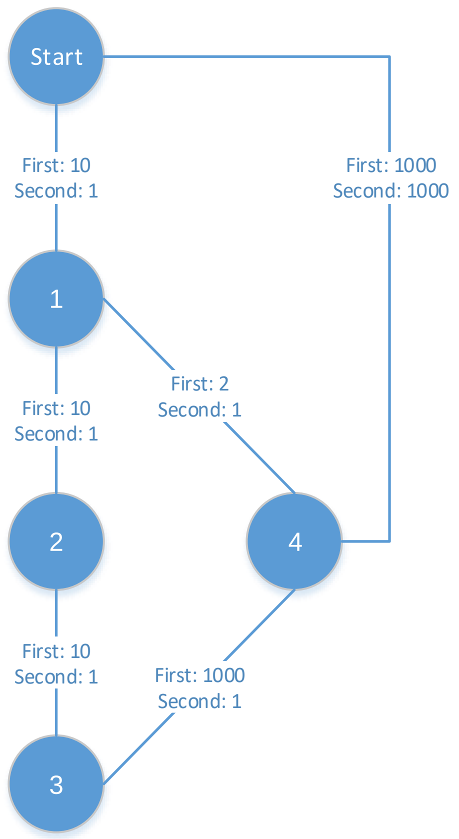
۲-۲ مساله‌ی فروشنده‌ی دوره‌گرد تعمیم‌یافته

فرض کنید برف جاده‌های بین شهرها را پوشانده‌است. و ما قصد داریم با شروع از یک شهر، با استفاده از دستگاه برف‌روب، تعدادی از مسیرهای بین شهرها را باز کنیم و سپس به شهر اولیه بازگردیم. به نحوی که بین هر دو شهر یک مسیر باز شده وجود داشته‌باشد. با توجه به اینکه در ابتدا مسیرها برفی هستند و پس از یک‌بار عبور دستگاه برف‌روب از این مسیر، مسیر بین دو شهر از برف خالی می‌شود؛ هزینه عبور از هر مسیر برای بار اول با هزینه عبور از همان مسیر برای بارهای بعدی متفاوت است. در نتیجه، به ازای هر جفت شهر، دو هزینه خواهیم داشت. با این فرض، مساله به صورت زیر تعریف می‌شود:

ورودی: شامل یک مجموعه از شهرها و هزینه‌های عبور در حالت برفی و غیربرفی بین هر جفت شهر است. هزینه عبور در هر دوجهت یکسان است.

خروجی: کوتاه ترین گشتی که با شروع از شهر اولیه، حداقل یک‌بار به تمام شهرها وارد شود و در نهایت به شهر اولیه بازگردد. (در این حالت، گشت‌هایی که از یک راس چند بار عبور می‌کنند مجاز هستند. همچنین مجاز هستیم که از یک یال چند بار عبور کنیم)

شکل ۲-۱ حالتی را نشان می‌دهد که جواب بهینه حالتی است که از یک یال چند بار عبور می‌کنیم.



شکل ۲-۱ مثالی از یک گراف که نیاز به استفاده چندباره از یک یال وجود دارد.

در شکل ۲-۱، هزینه عبور از یال هایی که در مشخص نشده است، برای بار اول و دوم بی‌نهایت فرض شده‌است. به ازای ما بقی یال ها، هزینه عبور به ازای اولین عبور و مابقی عبورها مشخص شده‌است.

در شکل ۲-۱، در حالتی که اجازه عبور مجدد از یک یال را نداشته باشیم بهترین گشت، گشت زیر خواهد بود[۲۰]:

Path = [Start, 1, 2, 3, 4, Start]

که هزینه آن برابر با ۲۰۳۰ می‌شود.

[۲۰] این گشت به ترتیب رویت رئوس نوشته شده‌است.

$$10 + 10 + 10 + 1000 + 1000 = 2030$$

اما در صورتی که مجاز باشیم از یک یال بیش از یک بار استفاده کنیم؛ بهترین گشت، گشت زیر خواهدشد:

Path= [Start, 1, 2, 3, 2, 1, 4, 1, Start]

که هزینه آن برابر با ۳۶ می‌شود.

$$10 + 10 + 10 + 1 + 1 + 2 + 1 + 1 = 36$$

در نتیجه عبور بیش از یک بار از چند یال، هزینه مسیر را به شکل چشم‌گیری کاهش داد.

پس از معرفی مساله، به بررسی کارهای پیشین می‌پردازیم.

۲-۳ کارهای پیشین

تعمیم‌های مختلفی از مساله‌ی فروشنده‌ی دوره‌گرد وجود دارد. مطالعه و بررسی مدل‌های گوناگون مساله، به فهم بهتر مساله کمک می‌کند. همچنین می‌توان از ایده‌های استفاده‌شده در کارهای دیگر نیز استفاده کرد.

۲-۳-۱ مساله‌ی جی‌تی‌اس‌پی [۲۱]

هدف از طرح این مساله، بیان مساله‌ی تی‌اس‌پی[۲۲] با قیود کمتر بوده است. در این مساله، از ما خواسته‌می‌شود که با گرفتن یک گراف دلخواه (که یال‌های آن مقادیر غیرمنفی هستند) کوتاه‌ترین گردشی بسته[۲۳] را به بعنوان خروجی مشخص کنیم که در آن تمام رئوس حداقل یک‌بار رویت شده‌اند. در این مساله، فروشنده‌ی دوره‌گرد می‌تواند از یک راس بیش از یک‌بار عبور کند. همچنین می‌تواند از یک یال بیش از یک‌بار عبور کند. [1]

[21] GTSP (Graphical Traveling Salesman Problem)

[22] TSP (Traveling Salesman Problem)

[23] Closed Walk

تفاوت این مساله با مساله‌ی مورد نظر ما این است که در مساله‌ی جی‌تی‌اس‌پی هزینه عبور از یال‌ها برای بار اول و بقیه بارها ثابت است.

۲-۳-۱-۱ تبدیل مساله‌ی جی‌تی‌اس‌پی به مساله‌ی تی‌اس‌پی معادل آن

مساله‌ی جی‌تی‌اس‌پی را می‌توان به مساله‌ی تی‌اس‌پی معادل آن تبدیل کرد. این بدین معنا است که در صورتی که بتوانیم مساله‌ی تی‌اس‌پی را به ازای تمام ورودی‌ها حل کنیم، می‌توانیم از آن برای حل مساله‌ی جی‌تی‌اس‌پی با ورودی مشخص استفاده‌کنیم. برای این‌کار باید ورودی مساله‌ی جی‌تی‌اس‌پی را به ورودی مناسب برای مساله‌ی تی‌اس‌پی تبدیل‌کنیم. همچنین اثبات‌کنیم که به ازای جواب بهینه در مساله‌ی جی‌تی‌اس‌پی یک جواب در مساله‌ی تی‌اس‌پی معادل وجود دارد. همچنین به ازای هر جواب بهینه در مساله‌ی تی‌اس‌پی یک جواب متناظر برای مساله جی‌تی‌اس‌پی وجود دارد.

فرض‌کنید مساله‌ی جی‌تی‌اس‌پی با گراف $G_1 = (V_1, E_1)$ داریم، حال می‌خواهیم با استفاده از این گراف، ورودی متناظر آن برای مساله‌ی تی‌اس‌پی را بدست بیاوریم. ورودی متناظر را گراف کامل $G_2 = (V_2, E_2)$ می‌نامیم. و آن را طوری تعریف می‌کنیم که در آن $V_1 = V_2$ و مقدار هر یال از i به j در گراف G_2 برابر با کوتاه‌ترین مسیر بین i به j در گراف G_1 است.

تاکنون گراف متناظر برای ورودی مساله‌ی تی‌اس‌پی را ساخته‌ایم. حال اثبات می‌کنیم که جواب‌های این دو مساله با هم تناظر یک به یک دارند.

برهان:

در صورتی که جواب بهینه در مساله جی‌تی‌اس‌پی را در نظر بگیریم و آن را به ترتیب ملاقات رئوس جدید نشان‌دهیم، در آن صورت، جواب ما به صورت شبیه به یک دور همیلتونی می‌شود. این نوع را فرم دو بنامیم.

برای مثال فرض کنید ترتیب رئوس در جواب بهینه مساله جی‌تی‌اس‌پی به شکل زیر باشد.

(فرم عادی) $1, 2, 3, 4, 2, 3, 5, 3, 6, 1$

در این حالت، ترتیب ملاقات رئوس جدید به شکل زیر خواهدبود.

(فرم دوم) 1, 2, 3, 4, 5, 6

می‌توان ادعا کرد در صورتی که فرم دوم جواب بهینه در م ساله جی‌تی‌اس‌پی را دا شته با شیم، باید برای عبور از هر راس به راس بعدی در فرم ۲، از کوتاه‌ترین مسیر بین آن‌ها عبور کنیم.

برهان خلف: در صورتی که برای عبور از دو راس در فرم دوم، از مسیری غیر از کوتاه‌ترین مسیر استفاده کرده‌باشیم، می‌توانیم آن مسیر را با کوتاه‌ترین مسیر جایگزین کنیم. در این صورت جواب جدید یک جواب درست برای مساله‌ی جی‌تی‌اس‌پی است. زیرا ما مطمئن هستیم که به تمام رئوس رفته‌ایم. علاوه بر این، این جواب از جواب قبل بهتر شده است. در نتیجه به تناقض می‌رسیم.

با توجه به استدلال بالا، فرم دوم هر جواب بهینه در مساله جی‌تی‌اس‌پی، به فرم یک دور درمی‌آید که برای هزینه بین هر دو راس متوالی، هزینه‌ی کوتاه‌ترین مسیر بین آن‌ها خواهد بود. در نتیجه می‌توان گفت این جواب معادل با یک دور در مساله‌ی تی‌اس‌پی در G_2 می‌باشد.

در نتیجه برای یافتن جواب مساله‌ی جی‌تی‌اس‌پی می‌توان به دنبال کوتاه‌ترین دور در مساله‌ی تی‌اس‌پی در G_2 رفت.

همچنین می‌توان نشان داد به ازای هر جواب بهینه در مساله‌ی تی‌اس‌پی می‌توان یک جواب بهینه در مساله‌ی جی‌تی‌اس‌پی یافت.

در [2] ارتباط بین مساله‌ی جی‌تی‌اس‌پی و مساله‌ی تی‌اس‌پی به صورت کامل‌تر توضیح‌داده‌شده‌است.

این مساله شباهت زیادی به مساله مطرح‌شده در این پروژه دارد. در صورتی که هزینه برگشت از هریال تغییر نکند، مساله‌ی ما به یک مساله‌ی جی‌تی‌اس‌پی تبدیل خواهدشد.

۲-۳-۲ مساله‌ی درخت پوشای کمینه[۲۴]

یکی از مفاهیم مفید در مسائل مربوط به مساله‌ی تی‌اس‌پی، مساله‌ی درخت پوشای کمینه است.

[24] Minimum Spanning Tree (MST)

۲-۳-۲-۱ تعریف مساله‌ی درخت پوشای کمینه

به زیرمجموعه‌ای از یال‌های یک گراف بی جهت که باعث می‌شوند تمام رئوس به هم راه‌داشته باشند و دور نداشته‌باشد، درخت پوشا گفته می‌شود.

به درخت پوشایی که مجموع وزن یال‌هایش، کمینه باشد، درخت پوشای کمینه گفته می‌شود.

۲-۳-۲-۲ شباهت درخت پوشای کمینه با فروشنده دوره‌گرد

شباهت این مساله، این است که در درخت پوشای کمینه سعی داریم تمام رئوس را ببینیم و جمع هزینه یال‌ها کمینه شود. حال آنکه در درخت پوشای کمینه نیز، می‌خواهیم تمام رئوس دیده شوند. همچنین در صورتی که یال آخر یک درخت پوشای کمینه را حذف کنیم، یک درخت پوشا با درجه حداکثر ۲ خواهیم داشت. البته این درخت لزوما درخت پوشای کمینه نیست.

۲-۳-۲-۳ شـــباهت درخت پوشـــای کمینه با مســـاله‌ی تعمیم یافته فروشـــنده دوره‌گرد (دی‌وی‌آرپی)

در صورتی که هزینه برگشت ثانویه به ازای تمام یال‌ها برابر با صفر باشد، در این صورت مساله‌ی دی‌وی‌آرپی تبدیل به درخت پوشای کمینه خواهدشد. زیرا ابتدا عبور از درخت درخت پوشای کمینه و برگشت از آن، یک جواب قابل قبول برای مساله‌ی دی‌وی‌آرپی خواهدبود. همچنین می‌توانیم به ازای هر مسیر قابل قبول در مساله‌ی دی‌وی‌آرپی یک درخت بدست بیاوریم که یال‌هایش زیرمجموعه ای از یال‌های مورد استفاده در مسیر مساله‌ی دی‌وی‌آرپی است. برای تولید این درخت کافیست یک مجموعه خالی در نظر گرفته و با شروع از راس اولیه، اولین یال مسیر را بررسی می‌کنیم. در صورتی که یال اول دارای راسی باشد که برای تا کنون دیده نشده است، آن یال را به مجموعه یال‌ها اضافه می‌کنیم. سپس به یال دوم از مسیر رفته و همین روال را انجام می‌دهیم. با توجه به اینکه تمام رئوس دیده شده اند و دور نداریم، این مجموعه مربوط به یک درخت پوشا خواهد بود. می‌دانیم که هزینه این درخت، از هزینه جواب در مساله‌ی دی‌وی‌آرپی کمتر نیست. همچنین می‌دانیم که هزینه این درخت نیز از هزینه درخت پوشای کمینه کمتر نیست. در نتیجه، هیچ جوابی در مساله‌ی دی‌وی‌آرپی بهتر از جواب درخت پوشای کمینه نخواهیم‌داشت.

با توجه به این شباهت و با توجه به اینکه در مسائل واقعی، احتمال اینکه هزینه ثانویه بسیار ناچیز باشد، مساله‌ی درخت پوشای کمینه اهمیت پیدا می‌کند.

فصل سوم
ویژگی‌های مساله‌ی فروشنده‌ی دوره‌گرد تعمیم‌یافته

۳- ویژگی‌های مساله‌ی فروشنده‌ی دوره‌گرد تعمیم‌یافته

در این فصل به بررسی ویژگی‌های مساله‌ی اصلی این پروژه یعنی مساله‌ی فروشنده‌ی دوره‌گرد تعمیم‌یافته می‌پردازیم.

۳-۱ هزینه مسیر

مسیر دلخواه P را در نظر بگیرید، هزینه این مسیر برابر با هزینه مجموع یال‌ها است. هزینه‌ی هر یال نیز بسته به تعداد عبور از آن یال محاسبه می‌گردد. [3]

فرض کنید مسیری به شکل زیر داشته‌باشیم:

$$\boldsymbol{P = \{e_1, \dots, e_n\}} \tag{۳-۱}$$

$$\boldsymbol{s(i) = count\ (e_i)\ in\ P} \tag{۳-۲}$$

$$\boldsymbol{Cost(p) = \sum_{i=1}^{n} Cost(e_i) = \sum_{e_i \in P} c_1(e_i) + (s(i) - 1) * c_2(e_i)} \tag{۳-۳}$$

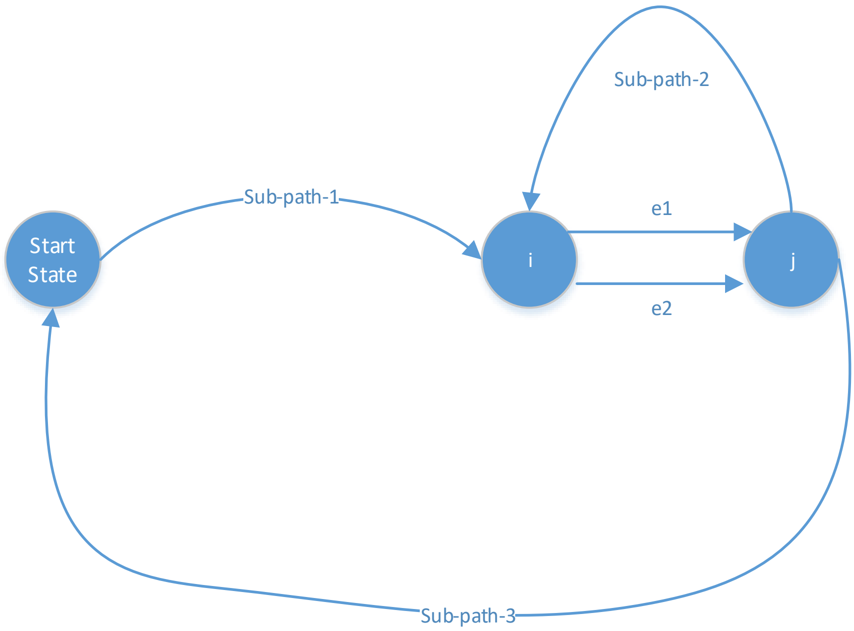
در نتیجه استفاده کمتر از هر یال باعث کاهش مجموع هزینه‌ها می‌شود.

۳-۲ حداکثر تعداد عبور از یک یال در یک جهت

برای بررسی حداکثر تعداد عبور از یک یال در یک جهت در جواب بهینه، ابتدا قضیه ۳-۱ را معرفی می‌کنیم.

قضیه ۳-۱: در مسیر بهینه، یک یال حداکثر یک بار در هرجهت پیموده خواهدشد.

برهان خلف: فرض‌کنید در مسیر بهینه از یک یال در یک جهت، بیش از یک بار عبور کرده باشیم. می‌توان مسیر را به صورت شکل ۳-۱ نشان داد.



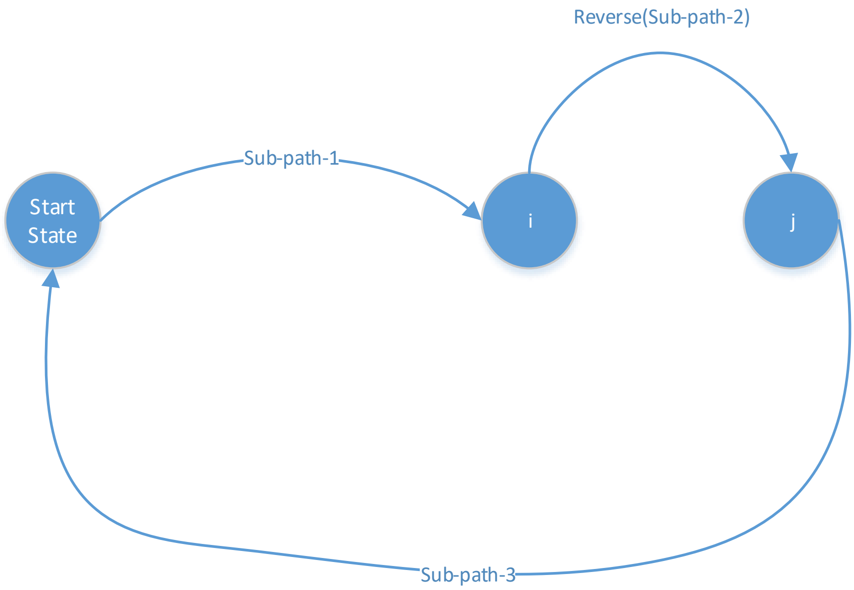
شکل ۳-۱ مسیر اولیه که دارای دو یال در یک جهت است.

در صورتی که بیش از دوبار از یال e_{ij} استفاده کرده‌باشیم، و این دوبار را e1 و e2 بنامیم، در صورتی که این مسیر را به ترتیب دیدن رئوس نشان دهیم، به شکل زیر خواهدشد.

Path1= [start-state, [sub-path-1], e1, [sub-path-2], e2, [sub-path-3], start-state]

می‌توان از روی Path1 مسیر جدیدی به نام Path2 را ساخت که در شرایط مساله صدق کند و از مسیر قبلی بهتر باشد.

در شکل ۳-۲ مسیر جدید Path2 را می‌بینید.



شکل ۳-۲ مسیر جدید پس از حذف دو یال اضافی

Path2= [start-state, [sub-path-1], Reverse (Sub-path-2), [sub-path-3], start-state]

که در آن Reverse (Sub-path-2) ترتیب پیمایش برعکس مسیر Sub-path-2 می‌باشد.

با توجه به اینکه یال e1 و e2 حذف شده‌است، طبق رابطه (۳-۳) هزینه کلی کاهش پیدا می‌کند. در نتیجه به تناقض می‌رسیم. ■

با توجه به قضیه فوق، در مسیر بهینه، از هر یال در هر جهت حداکثر یک‌بار عبور خواهدشد.

۳-۳ اثبات ان‌پی تمام بودن مساله‌ی فروشنده‌ی دوره‌گرد تعمیم‌یافته

توجه شود که اثبات ما برای نسخه‌ی تصمیم‌گیری مساله است.

طبق تعریف برای اثبات ان‌پی تمام بودن هر مساله نیاز است که دو ویژگی در مورد مساله اثبات شود.

- مساله در مجموعه مسائل ان‌پی باشد.
- مساله در مجموعه مسائل ان‌پی‌سخت باشد.

۳-۳-۱ اثبات ان‌پی بودن مساله

برای اثبات ان‌پی بودن، باید اثبات کنیم در صورتی که یک تصدیق‌کننده برای مساله داشته‌باشیم، می‌توانیم در زمان چندجمله‌ای درستی آن تصدیق‌کننده را بررسی‌کنیم.

توجه شود که ورودی در نسخه تصمیم‌گیری گراف G و مقدار l می‌باشد. و باید بگوییم با داشتن گراف G آیا قادر خواهیم‌بود دور بسته‌ای با حداکثر هزینه l داشته‌باشیم.

با توجه به اینکه در قضیه ۳-۱ اثبات کردیم که در مسیر بهینه حداکثر تعداد عبور از یک یال یک‌بار است، می‌توان گفت تعداد یال‌های در مسیر جواب کمتر در رابطه ۳-۴ صدق می‌کند.

$$path - length \leq 2 * \binom{n}{2} = O(n^2) \tag{۳-۴}$$

فرض کنید تصدیق کننده ما، مسیر بهینه است. در این صورت، برای بررسی اینکه تمام رئوس دیده شده است، باید تمام مسیر را بررسی کرد. پس به زمان $O(n^2)$ نیاز است و برای اینکه بررسی کنیم هزینه از l کمتر است نیز به $O(n^2)$ زمان نیاز است. در نتیجه، در زمان چندجمله‌ای، درستی جواب قابل بررسی خواهد بود. ■

۳-۳-۲ اثبات ان‌پی‌سخت بودن مساله

باید اثبات کنیم که مساله‌ی دیگری که آن مساله جزو مسائل ان‌پی‌سخت است، قابل تبدیل به مساله‌ی ما است. می‌دانیم مساله پیدا کردن دور همیلتنی یک مساله ان‌پی‌سخت است. می‌توان اثبات کرد که مساله‌ی جی‌تی‌اس‌پی نیز یک مساله‌ی ان‌پی‌سخت است. زیرا در صورتی که تمام یال‌ها در گراف اندازه یک داشته‌باشند، مساله‌ی پیدا کردن مسیر بسته با اندازه ثابت n در مساله‌ی جی‌تی‌اس‌پی در یک گراف با n راس، برابر با پیدا کردن دور همیلتنی در همان گراف خواهدشد. پس مساله پیدا کردن دور همیلتنی قابل تبدیل به مساله‌ی جی‌تی‌اس‌پی است. [4]

می‌توان به راحتی مساله جی‌تی‌اس‌پی را به مساله‌ی دی‌وی‌آرپی (مساله فروشنده دوره‌گرد تعمیم یافته) تبدیل‌کرد. به این ترتیب که به ازای ورودی $GTSP(G_1, l)$ مساله‌ی جایگزین $DVRP(G_1, G_1, l)$

را حل می‌کنیم. با توجه به اینکه مساله‌ی پیدا کردن دور همیلتنی یک مساله ان‌پی‌سخت است، روابط زیر برقرار است:

$$GTSP \ll_p DVRP \quad (۵-۳)$$

$$Hamiltonian\ Cycle \ll_p GTSP \quad (۶-۳)$$

با توجه به رابطه ۳-۵ و رابطه ۳-۶ به رابطه ۳-۷ می‌رسیم.

$$Hamiltonian\ Cycle \ll_p DVRP \quad (۷-۳)$$

با توجه به رابطه ۳-۷، مساله‌ی دی‌وی‌آرپی یک مساله‌ی ان‌پی‌سخت است. زیرا مساله پیدا کردن دور همیلتنی به مساله‌ی دی‌وی‌آرپی قابل تبدیل است. ■

با توجه به اینکه مساله فروشنده دوره‌گرد تعمیم‌یافته (دی‌وی‌آرپی) هم در مجموعه ان‌پی است و هم در مجموعه ان‌پی‌سخت، این مساله در مجموعه ان‌پی تمام قرار دارد. ■

۳-۴ تبدیل مساله به مساله‌ی برنامه‌ریزی خطی صحیح (آی‌ال‌پی)[۲۵]

با توجه به اینکه طبق قضیه ۳-۱ در مسیر بهینه از هر یال در هر جهت حداکثر یک‌بار عبور خواهیم‌کرد، می‌توان این مساله را به مساله‌ی برنامه ریزی خطی تبدیل کرد.

۳-۴-۱ تعریف برنامه ریزی خطی صحیح

برنامه ریزی خطی صحیح، یک مساله‌ی بهینه‌سازی ریاضیاتی است که در آن برخی از متغیرها باید عدد صحیح باشند و بقیه متغیرها اعداد حقیقی هستند. در این‌گونه مسائل تابع هدف و محدودیت‌ها خطی[۲۶] هستند. در این مسائل هدف کمینه کردن تابعی خطی از متغیرها است. شکل کلی این مسائل به صورت رابطه ۳-۸ است.

---

[25] Integer Linear Programming (ILP)

[26] Linear

$$maximize \sum a_i x_i \qquad (۸-۳)$$

همچنین شروط زیر برای متغیر های x_i وجود دارد. هر کدام از شروط می‌بایست به فرم رابطه ۳-۹ باشد.

$$subject\ to \sum b_i x_i \le c \qquad (۹-۳)$$

همچنین باید بعضی از متغیرها صحیح و بعضی دیگر اعداد حقیقی مثبت باشند.

$$x_i \ge 0 , \forall x_i \qquad (۱۰-۳)$$
$$xi \in \mathrm{Z} , some\ of\ i$$

۳-۴-۲ ان‌پی‌سخت بودن برنامه‌ریزی خطی صحیح

در صورتی که مساله برنامه‌ریزی خطی صحیح جزو مسائل ان‌پی‌سخت نباشد، از لحاظ نظری امکان ندارد بتوانیم مساله‌ی دی‌وی‌آرپی را به مساله برنامه‌ریزی خطی تبدیل کنیم. زیرا در صورتی که بتوانیم این تبدیل را انجام دهیم می‌توان نتیجه گرفت که برنامه ریزی خطی صحیح از یک مساله ان‌پی‌سخت، دشوار تر است. در نتیجه در دسته مسائل ان‌پی‌سخت قرار خواهد گرفت که با فرض بیان‌شده در تناقض است.

در نتیجه قبل از بررسی امکان تبدیل مساله دی‌وی‌پی‌آر به مساله برنامه‌ریزی خطی صحیح، می‌بایست از ان‌پی‌سخت بودن مساله برنامه‌ریزی خطی صحیح اطمینان حاصل کرد. به همین دلیل ابتدا به بررسی ان‌پی‌سخت بودن مساله برنامه‌ریزی خطی صحیح می‌پردازیم.

برای اثبات می‌توان مساله‌ی ارضای گزاره‌های بولی[۲۷] را به مساله‌ی آی‌ال‌پی تبدیل کرد. هر متغیر در مساله‌ی ارضای گزاره‌های بولی به یک متغیر در مساله‌ی آی‌ال‌پی تبدیل می‌شود و قیود طوری قرار داده‌می‌شود که در صورتی که تابع هدف بیشینه شود، انگار تمام شرط‌ها در مساله‌ی ارضای گزاره‌های بولی برقرار شده است.

[27] SAT یا Boolean Satisfiability Problem

۳-۴-۳ نحوه تبدیل مساله به برنامه‌ریزی خطی صحیح

پس هم اکنون با توجه به اینکه مساله‌ی آی‌ال‌پی یک مساله‌ی ان‌پی‌سخت است و مساله‌ی دی‌وی‌آرپی یک مساله ان‌پی تمام، می‌توان مطمئن بود که از لحاظ نظری مساله‌ی دی‌وی‌آرپی با استفاده از برنامه‌ریزی خطی قابل حل است. در نتیجه به حل مساله می‌پردازیم.

رابطه‌های مربوطه در مساله به شکل زیر خواهد بود:

ابتدا متغیر های x_{ij} را تعریف می‌کنیم که دو مقدار می‌گیرد. اگر $x_{ij} = 0$ باشد یعنی در مسیر جواب از راس i به راس j نخواهیم‌رفت. اگر $x_{ij} = 1$ باشد یعنی این یال جهت‌دار در مسیر جواب وجود دارد. همچنین اگر متغیر $x'_{ij} = 1$ باشد، در این صورت از این یال برای بار دوم عبور کرده‌ایم. بدین ترتیب با توجه به اینکه مقدار c_{ij} و c'_{ij} هزینه عبور از یال برای بار اول و دوم می‌باشد، هزینه کلی یک مسیر جواب به صورت رابطه ۳-۹ خواهد شد. که در آن قصد داریم هزینه را کمینه کنیم.

$$minimize\ w = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} + c'_{ij}x'_{ij} \tag{۱۱-۳}$$

شرط های زیر نیز برای متغیر های ما وجود دارد:

$$x'_{ij} \le x_{ji}\ , \forall i,j: i \neq j, 1 \le i,j \le n \tag{۱۲-۳}$$

$$x_{ij} + x_{ji} \le 1, x_{ij} + x'_{ij} \le 1\ , \qquad \forall i,j: i \neq j, 1 \le i,j \le n \tag{۱۳-۳}$$

$$x_{ii} = 0\ x'_{ii} = 0\ , \forall i: 1 \le i \le n \tag{۱۴-۳}$$

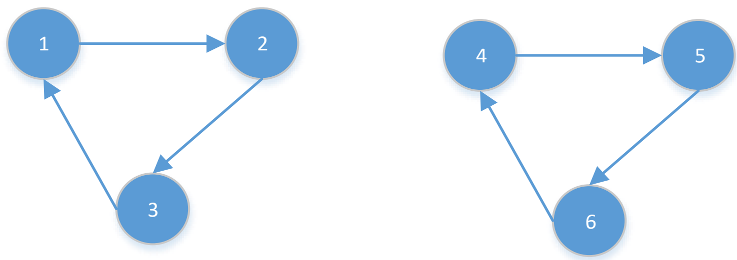
$$\sum\nolimits_{i=1}^{n} x_{ij} + x'_{ij} \ge 1\ , \forall j: 1 \le j \le n \tag{۱۵-۳}$$

$$\sum\nolimits_{j=1}^{n} x_{ij} + x'_{ij} = \sum\nolimits_{j=1}^{n} x_{ji} + x'_{ji}\ \ , \forall i: 1 \le i \le n \tag{۱۶-۳}$$

$$\sum_{i\in S}\sum_{j\in \bar{S}} x_{ij} + x'_{ij} \ge 1\ \ , for\ all\ non\ empty\ partition\ (S,\bar{S}) \tag{۱۷-۳}$$

$$x_{ij}, x'_{ij} \in \{0,1\} \tag{۱۸-۳}$$

شرط ۳-۱۲ برای این است که در صورتی از هزینه مربوط به x'_{ij} در محاسبه فرمول استفاده‌شود که یک بار در جهت مخالف از این یال عبور کرده‌باشیم. شرط ۳ -۱۳ برای این است که به ازای هر یال، حداکثر یک‌بار هزینه اولیه حساب شود و هزینه عبور بعدی، با هزینه کمتر محاسبه شود. شرط ۳-۱۴ باعث می‌شود از هر راسی به خود آن راس نرویم. شرط ۳-۱۵، تضمین می‌کند که به هر راسی حداقل یک ورود را داشته‌ایم. شرط ۳-۱۶ می‌گوید به ازای هر ورود به یک راس، باید از آن راس خارج شد. شرط ۳-۱۷ باعث می‌شود که چند مسیر جداگانه از هم بوجود نیاید. برای توضیح بیشتر به شکل ۳-۳ توجه کنید.



شکل ۳-۳ مثالی از یک جواب که در آن تمام شرط ها به جز شرط ۳-۱۵ رعایت شده است.

در صورتی که شرط سوم برقرار شود، به ازای این افراز دوتایی $S = \{1,2,3\}$ و $S' = \{4,5,6\}$ قید ۳-۱۷ رعایت نمی‌شود در نتیجه این جواب، از مجموعه جواب‌ها حذف می‌شود. [3]

در این برنامه‌ریزی خطی، تمام قیود به غیر از قید ۳-۱۷، دارای تعداد چندجمله‌ای است. اما تعداد افرازهای ممکن در قید ۳-۱۷ از مرتبه‌ی نمایی است. در نتیجه برای حل این مشکل از متغیرهای جدیدی به نام f_{ij} استفاده‌کردیم. تعبیر f_{ij} جریانی است که از راس i به راس j منتقل می‌شود. ما در این جا، راس ۱ را مبدا جریان قرار می‌دهیم و سپس جریانی با اندازه $n-1$ را به رئوس دیگر منتقل می‌کنیم. در صورتی که به همه رئوس جریان برسد، می‌توان تضمین کرد که از هر راسی به راس ۱، مسیری وجود داشته است. در نتیجه جواب ما به صورت چند قسمتی مانند شکل ۳-۳ نمی‌شود.

قیدهای جدید به صورت زیر هستند.

$$f_{ij} \le \left(x_{ij} + x'_{ij}\right)(n-1) \tag{۱۹-۳}$$

$$\sum_{j=1}^{n} f_{ji} - \sum_{j=1}^{n} f_{ij} = 1 \,, \forall i: 2 \le i \le n \tag{۲۰-۳}$$

$$\sum_{j=1}^{n} f_{1j} = n - 1 \tag{۲۱-۳}$$

رابطه ۳-۱۹ بیان می‌کند در صورتی که یال ij در مسیر وجود دارد، می‌توان حداکثر مقدار $n-1$ واحد جریان از آن عبور داد. شرط ۳-۲۰ بیان می‌کند که تمام رئوس به غیر از راس اول، باید یک واحد جریان را در خود نگه‌دارند. و شرط ۳-۲۱ بیان می‌کند که جریان خروجی از راس اولیه دقیقا مقدار $n-1$ است. بدین ترتیب، با اضافه کردن قیود جدید، مشکل شکل ۳-۳ بوجود نمی‌آید. همچنین تعداد قیود ما از مرتبه‌ی $O(n^2)$ است که n تعداد رئوس گراف ما می‌باشد.

فصل چهارم
روش حل مساله توسط الگوریتم‌های تقریبی

۴- روش حل مساله توسط الگوریتم‌های تقریبی

در این فصل الگوریتم‌های استفاده شده برای حل مساله را معرفی می‌کنیم. در فصل بعد به جزییات پیاده‌سازی این الگوریتم‌ها می‌پردازیم.

۴-۱ روش‌های استفاده شده در این پروژه

در این پروژه دو الگوریتم ژنتیک و سردکردن تدریجی برای حل مساله استفاده شده‌است.

۴-۱-۱ روش سردکردن تدریجی

روش سردکردن تدریجی یک روش احتمالاتی برای تخمین جواب بهینه سراسری است. از این روش معمولا در جاهایی که جواب‌های ممکن گسسته هستند، استفاده‌می‌شود. این روش برگرفته شده از سرد کردن تدریجی در متالورژی است که با توجه به نحوه سردکردن، ماده حاصل شده تفاوت می‌کند.

در این روش، برخلاف روش تپه نوردی ساده[۲۸] این امکان وجود دارد که به جواب‌های همسایه‌ای که بدتر از جواب کنونی هستند حرکت کنیم. بدین ترتیب احتمال گیرکردن در بهینه محلی کاهش پیدا می‌کند. این احتمال با مرور زمان کاهش پیدا می‌کند. در نتیجه در ابتدای اجرای الگوریتم، الگوریتم سعی می‌کند جستجوی مناسبی برای پیدا کردن محل بهینه محلی انجام دهد.[۲۹] سپس در انتهای الگوریتم، سعی می‌شود که جواب دقیق در آن محل را پیدا کند.[۳۰]

مدل های مختلفی برای پیاده‌سازی این روش وجود دارد. در بعضی از این روش‌ها چند پارامتر آزاد داریم و در بعضی از آن‌ها فقط یک پارامتر آزاد داریم. یکی از این مدل‌ها به صورت شکل ۴-۱است. [5]

[28] Hill Climbing

[۲۹] به این کار Exploration نیز می‌گویند.

[۳۰] به این کار Exploitation نیز می‌گویند.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE – current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

شکل ۴-۱ نحوه پیاده سازی روش سردکردن تدریجی

در این روش، ابتدا یک حالت اولیه از جواب ساخته می‌شود، سپس در for ابتدا، دما توسط تابع schedule با توجه به زمان t مشخص می‌شود. سپس یکی از حالت های ثانویه، به صورت تصادفی تولید می‌شود. در صورتی که حالت ثانویه از حالت اولیه بهتر بود، حالت جدید را می‌پذیریم. در غیر این صورت، با یک احتمال که وابسته به دمای T است، ممکن است حالت جدید را بپذیریم. هرچه دما بالاتر باشد احتمال پذیرفتن حالت جدید بیشتر است.

۴-۱-۲ روش ژنتیک

این روش الهام‌گرفته از انتخاب طبیعی در طبیعت می‌باشد و جزو دسته الگوریتم‌های تکاملی است. در این روش سعی در شبیه سازی بقای نسل در حیوانات را داریم. این روش دارای چند بخش اصلی است. نحوه پیاده‌سازی این روش معمولا به صورت شکل ۴-۲ است.

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

شکل ۴-۲ پیاده سازی روش ژنتیک

در این الگوریتم، نیازمند هستیم جواب مساله خود را تبدیل به یک کروموزوم[۳۱] معادل آن جواب بکنیم. در حالت کلاسیک هر کروموزوم معمولا یک رشته از 0 و 1 است. اما در این پروژه، ما کروموزوم‌ها را رشته‌ای از اعداد صحیح می‌دانیم. همچنین برای اجرای این الگوریتم، نیازمند این هستیم که میزان مطلوب بودن[۳۲] هرکدام از کروموزوم‌ها که در واقع جواب هستند را بدست بیاوریم. این تابع به نام FITNESS_FN در الگوریتم شکل ۴-۲ دیده می‌شود. در این الگوریتم، ابتدا یک جمعیت تصادفی از جواب‌ها داریم. سپس برای بوجود آوردن نسل جدید، دو فرد x و y را با توجه به شایستگی آن‌ها انتخاب کرده و فرزندی را با توجه به کروموزوم های این دو فرد بوجود می‌آوریم. سپس به احتمال کمی، در کروموزوم‌های این فرزند جهش[۳۳] بوجود می‌آوریم. سپس با داشتن فرزندان جدید و نسل قدیمی، نسل جدید را بوجود می‌آوریم. در شکل ۴-۲ نسل جدید، صرفا از فرزندان تولید شده بدست می‌آید. اما در حالت کلی می‌توان ترکیبی از نسل جدید و نسل قدیمی را بعنوان نسل جدید انتخاب کرد. در پروژه‌ی ما از حالت ترکیبی استفاده شده‌است.

---

[31] Chromosome

[32] Fitness

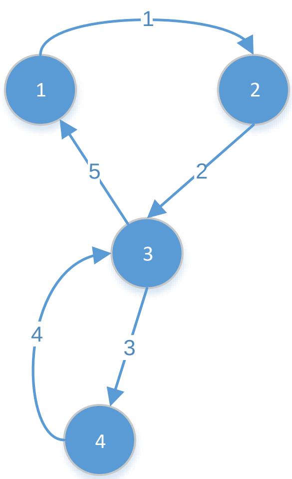
[33] Mutation

زمان خاتمه این الگوریتم زمانی خواهد بود که یا یکی از افراد جمعیت به شایستگی مطلوب رسیده باشد یا اینکه تعداد نسل‌های بوجود آمده از عددی خاص تجاوزکند.

۲-۴ نحوه نشان‌دادن جواب و ایجاد تغییر

۱-۲-۴ نحوه نشان دادن جواب

در این روش، ابتدا نیازمند این هستیم که جواب مساله را به نحوی نشان دهیم. برای نشان دادن جواب‌ها در این مساله، روش‌های مختلفی وجود دارد. یکی از روش‌ها این است برای اینکه جواب را نشان‌دهیم، به ترتیب مشاهده رئوس توجه کنیم. و ترتیب مشاهده رئوس را در یک لیست نشان دهیم. بعنوان مثال مسیر شکل ۴-۳ را می‌توان به صورت [1,2,3,4,3] نشان داد. (توجه کنید که یال شماره ۵ که به راس اولیه برمی‌گردد را در این لیست نیاورده‌ایم.)



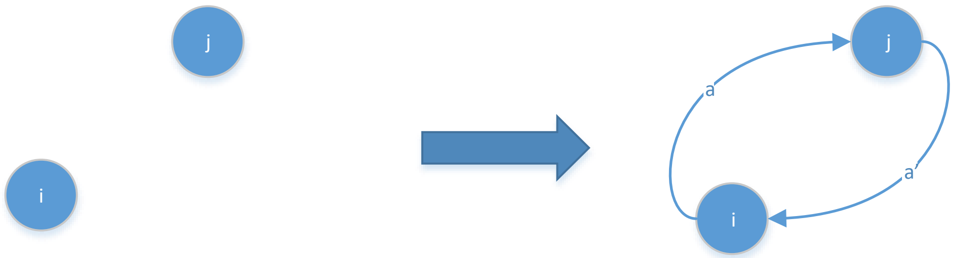
شکل ۴-۳ یک نمونه از جواب، اعداد روی یال ها ترتیب عبور را نشان می‌دهند

پس از مشخص‌کردن نحوه نشان‌دادن جواب، نیازمندآن هستیم که مجموعه کارهای ممکن برای تغییر جواب به حالت های دیگر را مشخص‌کنیم.

۴-۲-۲ تغییر حالت

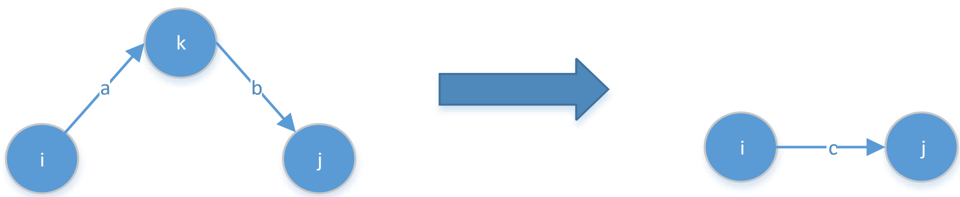
در اجرای الگوریتم سردکردن تدریجی نیاز به تغییر حالت وجود دارد. [6] برای تغییر حالت، ما چند نوع تغییر حالت معرفی می‌کنیم.

- حرکت نوع صفرم: این حرکت که معمولا در ابتدای تولید می‌شود باعث اتصال یک راس جدید به مسیر می‌شود.



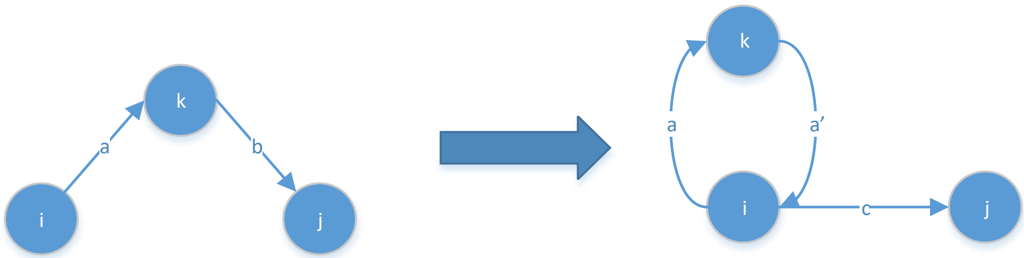
شکل ۴-۴ حالت صفرم

- حرکت نوع اول: در این حالت یکی از رئوس داخل مسیر، از مسیر حذف می‌شود. در صورتی که این راس در هیچ‌جای دیگر مسیر استفاده نشده‌باشد، این حرکت مجاز نیست.



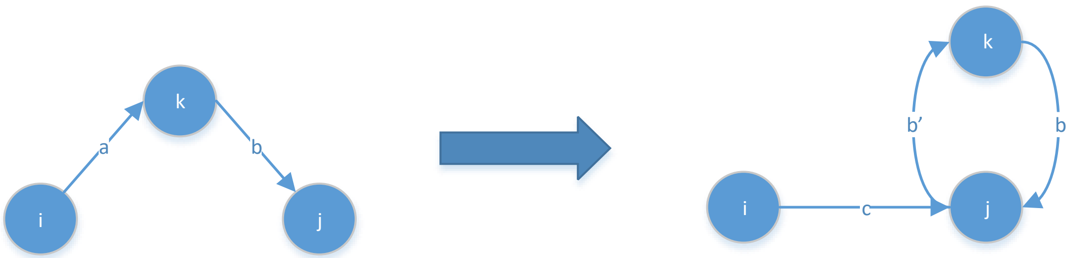
شکل ۴-۵ حالت اول تغییر

- حرکت نوع دوم: در این حالت، سعی می‌شود از یک یال، در جهت برعکس استفاده‌شود.



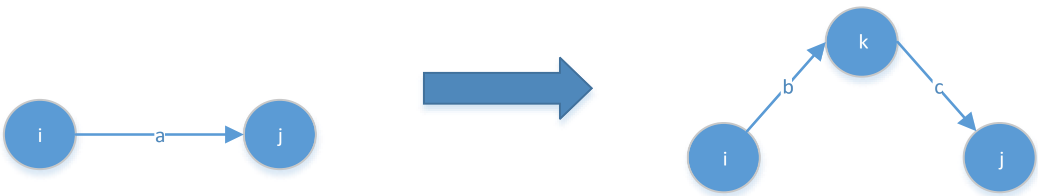
شکل ۴-۶ حالت دوم تغییر

- حرکت نوع سوم: حالت سوم مشابه حالت دوم است. تفاوت در این است که سعی می‌شود از یال بازگشتی دیگری استفاده کند.



شکل ۴-۷ حالت سوم

- حرکت نوع چهارم: در این حالت سعی می‌شود یک راس جدید به مسیر اضافه‌شود. در این حرکت معمولا حداقل یکی از یال‌های b و یا c از یال a باید کوچک‌تر باشند.



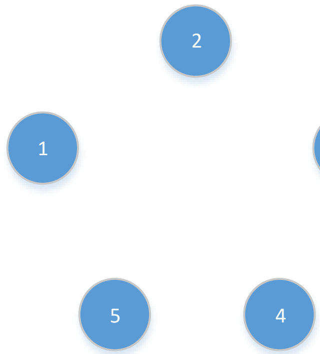
شکل ۴-۸ حالت چهارم

۴-۲-۳ امکان بوجود آمدن هر مسیر دلخواه با استفاده از حرکات موجود

در این قسمت ما توضیح می‌دهیم که با مجموع این حرکت‌ها می‌توان مسیر دلخواه خود را ساخت. با توجه به اینکه هر مسیر دلخواه را می‌توان به ترتیب رویت رئوس دید، ما در این‌جا قصد داریم توضیح دهیم که چگونه با استفاده از حرکت‌های مجاز از یک حالت اولیه به حالت نهایی می‌رسیم.

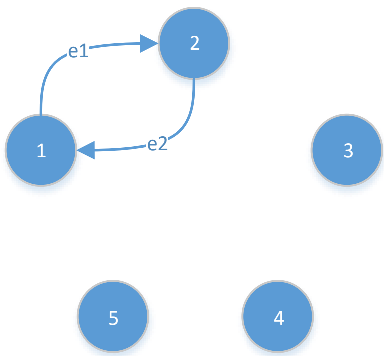
حالت اولیه ما طبق تعریف خودمان حالتی است که در آن هیچ یالی وجود ندارد و تمام رئوس به صورت منزوی هستند. حالت نهایی ما حالتی است که می‌توان دقیقا تمام مسیر توسط یال‌های جهت دار نشان داده می‌شود. برای مثال فرض کنید ۵ راس داریم، و می‌خواهیم از حالت اولیه به حالت [1,2,3,2,4,1,5] برسیم.

در این حالت ابتدا تمام رئوس منزوی هستند.



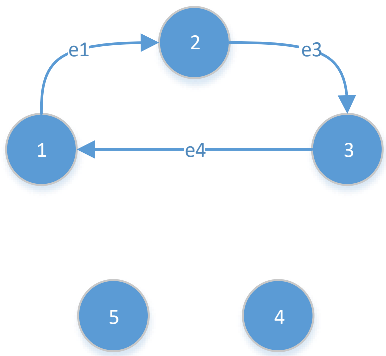
شکل ۴-۹ حالت اولیه

در این حالت از حرکت اول استفاده می‌کنیم و رئوس اول و دوم مسیر را به هم وصل می‌کنیم. مسیر کنونی [1,2] می‌باشد.



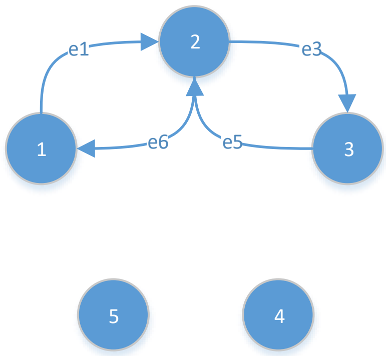
شکل ۴-۱۰ مرحله دوم

پس از آن با انجام حرکت نوع چهارم روی یال e2، راس سوم را به مسیر اضافه می‌کنیم. در نتیجه مسیر به صورت [1,2,3] می‌شود.



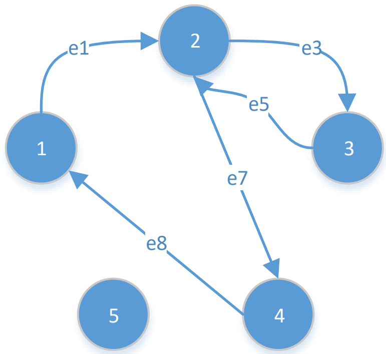
شکل ۴-۱۱مرحله سوم

سپس با انجام حرکت نوع چهارم بر روی یال e4، راس ۲ را دوباره به مجموعه رئوس اضافه می‌کنیم. مسیر کنونی [1,2,3,2] است.



شکل ۴-۱۲ مرحله چهارم

سپس با انجام حرکت نوع چهارم بر روس راس e6، می‌توان به حالت [1,2,3,2,4] رسید.



شکل ۴-۱۳ مرحله پنجم

می‌توان به همین ترتیب، بقیه مسیر را تولید کرد. توجه شود که ممکن است برای تولید یک مسیر بتوان با چند دنباله متفاوت از حرکت‌ها آن مسیر را ساخت. یکی از آن دنباله‌ها به صورت استفاده از حرکت نوع اول برای اولین یال و سپس استفاده از حرکت نوع چهارم به صورت پی‌درپی برای ساخت بقیه یال‌ها خواهدبود.

البته در پیاده‌سازی این مدل کمی تفاوت وجود دارد. بدین ترتیب که حالت اولیه متفاوت با مثال بالا است. حالت اولیه، یک دور تصادفی از رئوس گراف است. و با انجام مجموعه‌ای از حرکات سعی می‌کنیم به جواب برسیم.

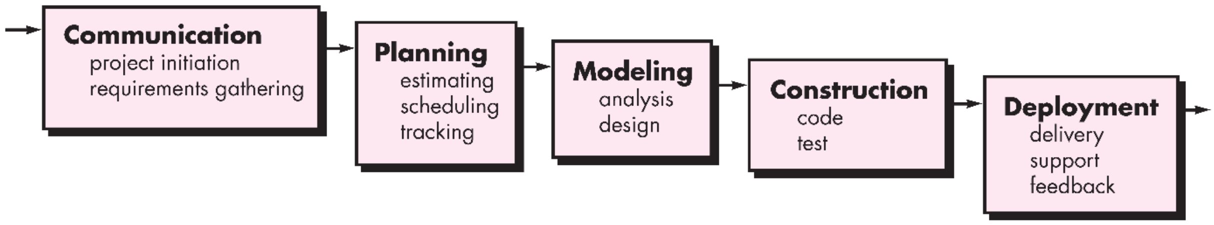
فصل پنجم
ارزیابی فرایند تولید و بررسی الگوریتم‌ها

۵- ارزیابی فرایند تولید و بررسی الگوریتم‌ها

در این بخش در ابتدا به تحلیل و طراحی مدل نرم‌افزاری می‌پردازیم و سپس مدل تولید شده را ارزیابی می‌کنیم. سپس به بررسی عملکرد الگوریتم‌های مورد استفاده در این پروژه می‌پردازیم.

۵-۱ تحلیل و طراحی نرم‌افزار

با توجه به مشخص بودن نیازهای معرفی شده و ثابت بودن نیازها در این نرم‌افزار، می‌توان از مدل آبشاری[۳۴] برای توسعه این نرم‌افزار استفاده کرد. این مدل شامل پنج مرحله ارتباط[۳۵]، برنامه‌ریزی[۳۶]، مدل‌سازی[۳۷]، ساخت[۳۸] و گسترش[۳۹] است.[7]



شکل ۵-۱ مدل فرایند آبشاری

با توجه به اینکه هدف از تولید این برنامه آشنایی با استفاده از روش‌های ژنتیک و سردکردن تدریجی در حل یک مساله مشخص بود، ارتباط ما با مشتری فقط در مرحله اولیه تعریف مساله خواهدبود. با توجه

34 Waterfall Process

35 Communication

36 Planning

37 Modeling

38 Construction

39 Deployment

به اینکه مساله به صورت دقیق در ابتدای تعریف پروژه مشخص شده‌است، نیازی به تکرار ارتباط با مشتری نخواهیم‌داشت. از مدل‌های با جریان فرایند خطی[۴۰] همچون فرایند آبشاری استفاده شده‌است.

مزیت‌های استفاده از این مدل عبارتند از:

- فهم آسان مدل
- مستندات کمتر در این مدل، که باعث می‌شود تمرکز بر روی قسمت‌های اصلی معطوف شود.
- پایان هر مرحله مشخص است و به راحتی قابل ارزیابی است.
- با توجه به مساله‌ی مورد نظر، مراحل تقریبا هم‌پوشانی ندارند.

۵-۲ مراحل فرایند

گام اول، برقراری ارتباط برای بدست آوردن نیازمندی‌های مساله است. در این مساله، این گام، در واقع پیدا کردن مساله‌ی مناسب برای رفع یک نیاز واقعی است. که پس از مطالعه و بررسی، مساله‌ی فروشنده‌ی دوره‌گرد تعمیم‌یافته به عنوان مساله انتخاب گردید.

گام بعدی، برنامه‌ریزی است. در این مساله، برنامه ریزی، شامل مشخص‌کردن زمان انجام بخش‌های مختلف و برنامه‌ریزی برای انجام آن‌ها است.

در قسمت بعدی، ما به مدل‌سازی نرم‌افزار می‌پردازیم. در این قسمت با توجه به نیازمندی‌های تعریف شده در مساله، تحلیل صورت می‌گیرد و نمودارهای مربوط به تحلیل و طراحی نرم‌افزار تولید می‌شود.

سپس با توجه به مدل، برنامه مربوطه را تولید می‌کنیم. و در مرحله آخر محصول نهایی را ارائه می‌کنیم.

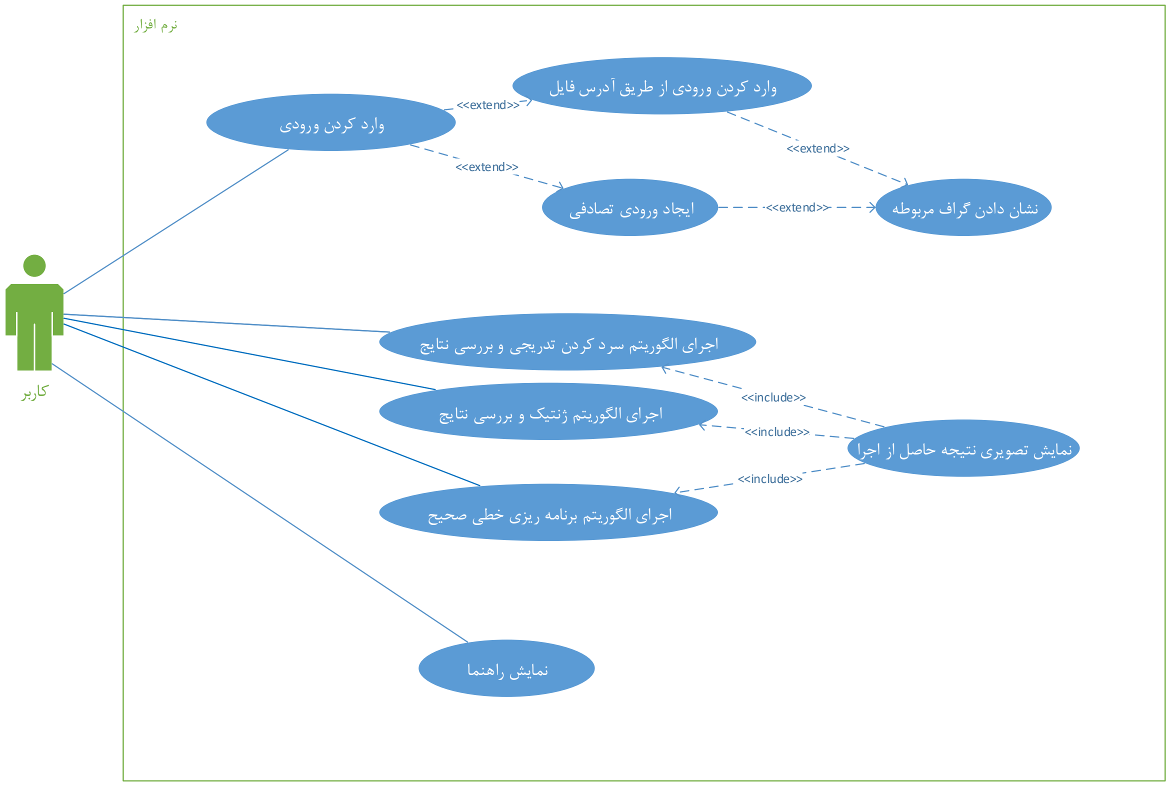
۵-۳ مدل‌سازی نرم‌افزار

در این مرحله، ما به مدل‌سازی مساله می‌پردازیم. با توجه به تعریف پروژه، مدل درخواست سیستم[۴۱] به صورت شکل ۵-۲ است.

---

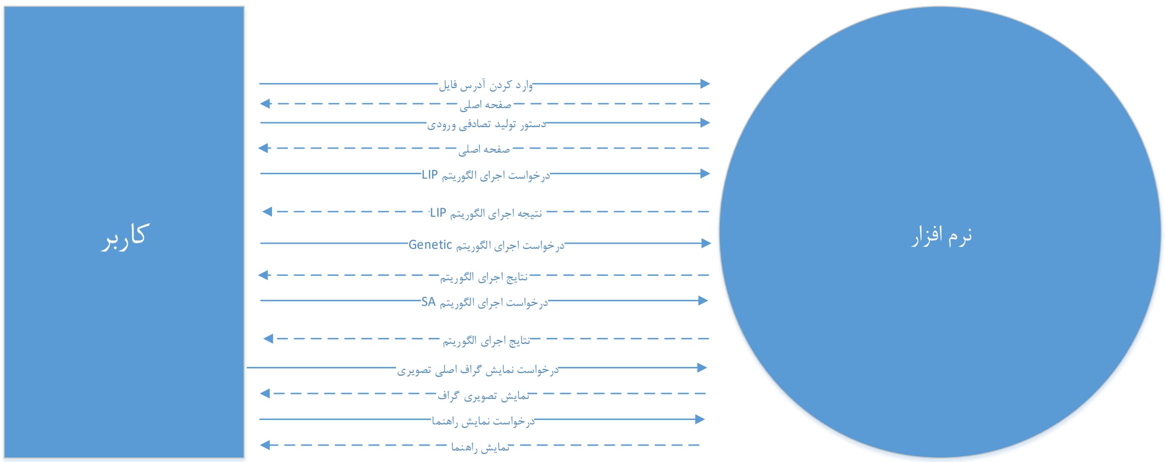
[40] Linear Process Flow

[41] Use Case Diagram



شکل ۵-۲ نمودار درخواست سیستم

نمودار جریان داده سیستم[42] به صورت شکل ۵-۳ است:

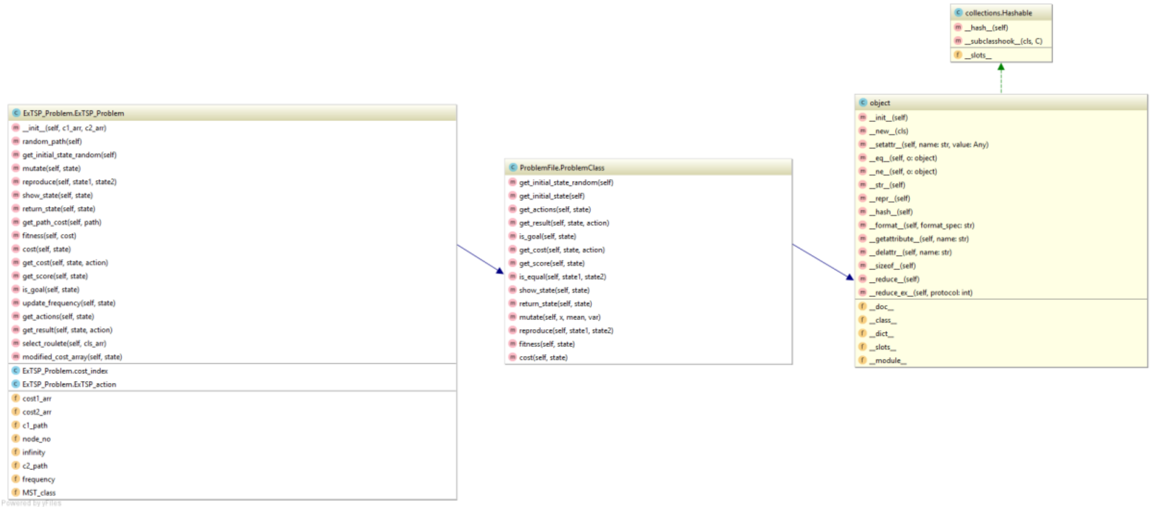


شکل ۵-۳ نمودار جریان داده سیستم

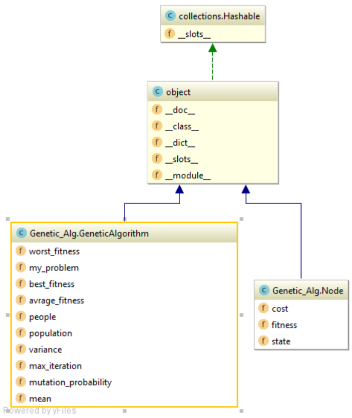
[42] Context Diagram

۵-۳-۱ نمودار کلاس‌ها[43]

در این قسمت نمودار مربوط به کلاس‌ها را نشان می‌دهیم.

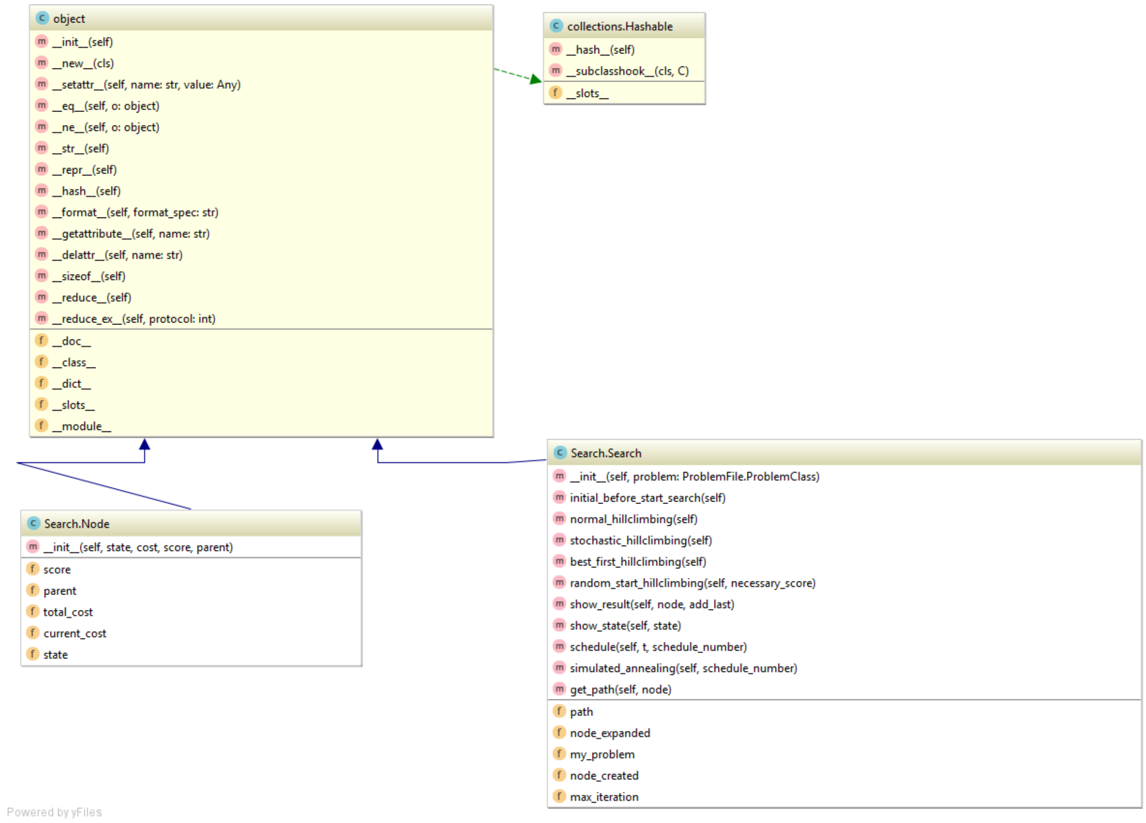


شکل ۵-۴ نمودار کلاس ExTsp_problem



شکل ۵-۵ نمودار کلاس Genetic_algorithm

[43] Class Diagram



شکل ۵-۶ نمودار کلاس **Simulated Annealing**

۵-۳-۲ رابط کاربری گرافیکی

برای ایجاد رابط گرافیکی از کتابخانه PyQt5 در python استفاده کردیم. در این پروژه از QMainWindow برای ایجاد صفحه اصلی استفاده کردیم. سپس برای اضافه کردن عناصر مختلف به کلاس، کلاس جدیدی به نام MyTableWidget ساختیم که از QWidget ارثبری[۴۴] می‌کند. صفحه اصلی شامل ۳ نوار است.

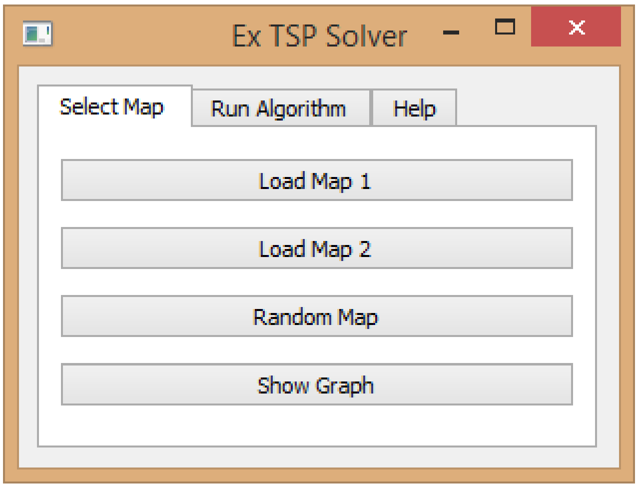
۵-۳-۲-۱ نوار اول

مربوط به انتخاب گراف مورد نظر است. در این‌جا شما می‌توانید فایل مورد نظر را برای گراف انتخاب کرده و یا اینکه به صورت تصادفی نقشه را بکشید.

44 Inherit

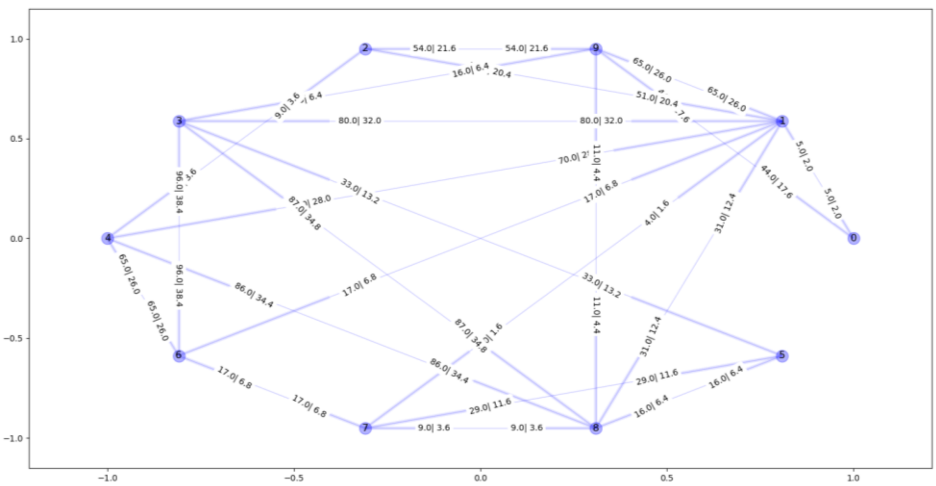
در صورتی که نقشه را انتخاب کردید، می‌توانید با فشردن دکمه 'Show Graph'، گراف حاصل را ببینید.

برای پیاده‌سازی دکمه‌ها کلاس‌های مختلفی را تعریف کردیم. که این کلاس‌ها از QPushButton ارثبری می‌کنند.



شکل ۵-۷ تصویر نوار Select Map

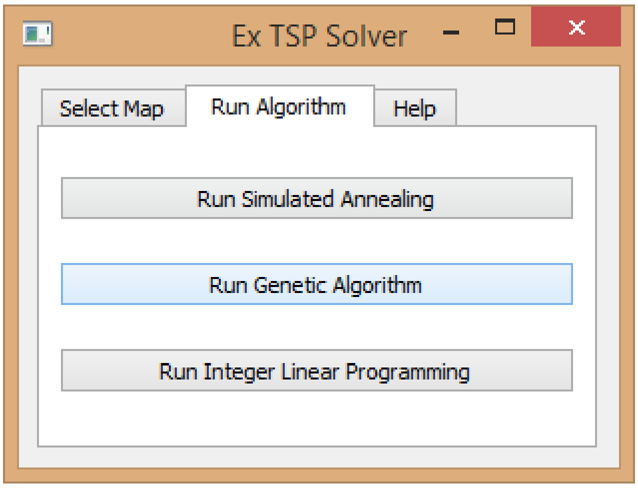
در صورتی که دکمه 'Show Graph' را بزنید، گرافی شبیه شکل ۵-۸ نمایش داده می‌شود.



شکل ۵-۸ تصویر گراف

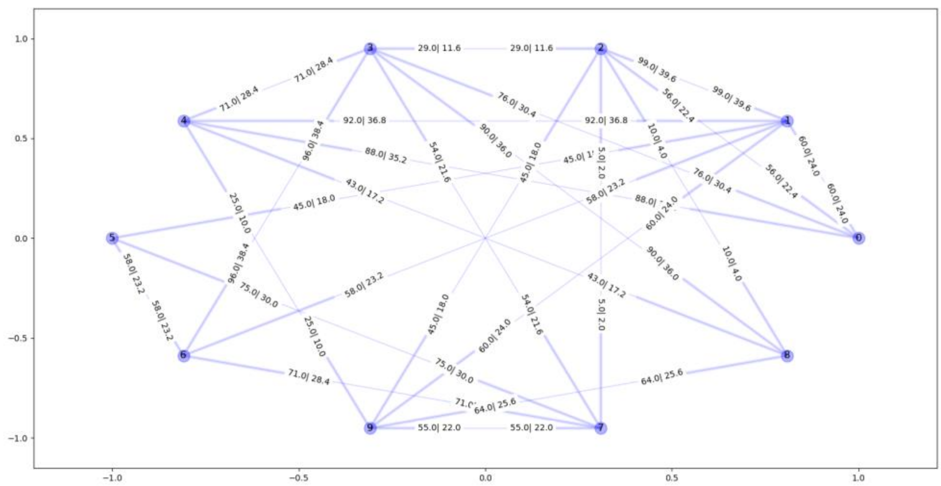
۵-۳-۲-۲ نوار دوم

نوار دوم برای اجرای الگوریتم‌های پیاده‌سازی شده در این پروژه است. این نوار دارای سه دکمه است. در صورت فشار دادن هرکدام از این دکمه‌ها، الگوریتم مربوطه اجرا می‌شود و گراف حاصل نشان داده می‌شود.

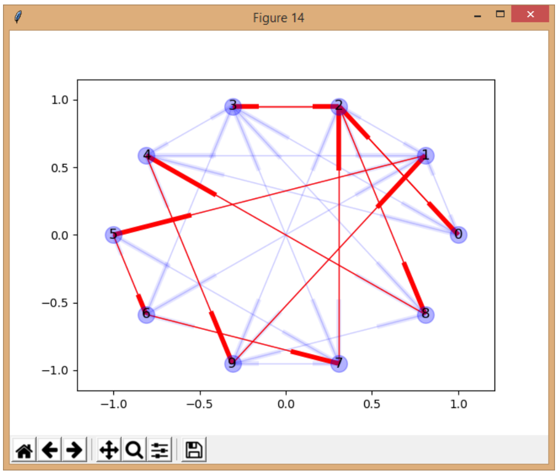


شکل ۵-۹ نوار دوم

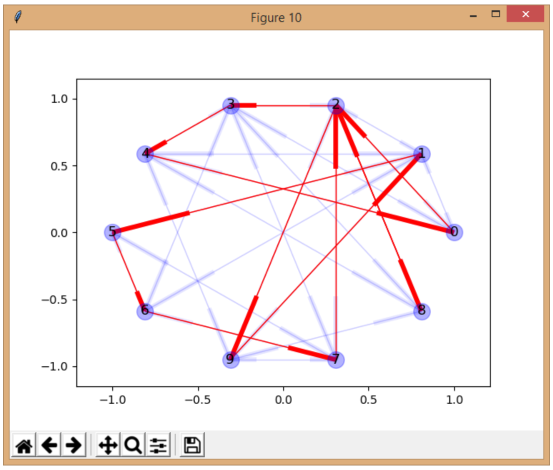
برای مثال برای گراف شکل ۵-۱۰ خروجی‌ها به صورت زیر است.



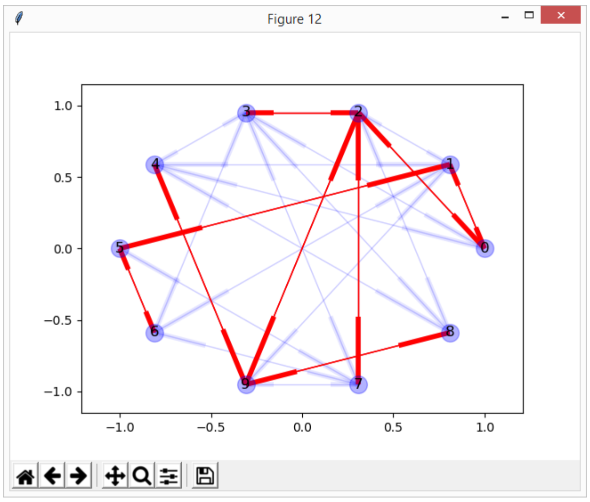
شکل ۵-۱۰ گراف اولیه



شکل ۵-۱۱ خروجی مربوط به آی‌ال‌پی (جواب بهینه)



شکل ۵-۱۲ خروجی مربوط به سردکردن تدریجی



شکل ۵-۱۳ خروجی مربوط به الگوریتم ژنتیک

۵-۴ نحوه پیاده سازی الگوریتم‌ها

در این پروژه، ما قصد داشتیم، سه الگوریتم متفاوت را پیاده‌سازی کنیم. برای اینکه بتوانیم این سه الگوریتم را به صورت مستقل از هم پیاده سازی کنیم، همچنین برای اینکه بتوانیم الگوریتم‌ها را تا جای ممکن از مساله مستقل بکنیم، از کلاس‌های زیر استفاده کردیم.

۵-۴-۱ کلاس‌ها

در این قسمت به تعریف کلاس‌ها می‌پردازیم. ترتیب معرفی توابع، به ترتیب بیان آن‌ها وابسته است.

۵-۴-۱-۱ کلاس ProblemClass

این کلاس، کلاس اصلی برای تعریف یک مساله است. ما در این کلاس، صرفا تعریف نام توابع و ورودی توابع مشخص شده‌است. با توجه به الگوریتم‌های مورد استفاده، داشتن این توابع برای اجرای الگوریتم‌ها لازم است.

```
class ProblemClass:
    def get_initial_state_random(self):
        pass
    def get_initial_state(self):
        pass
    def get_actions(self, state):
        pass
    def get_result(self, state, action):
        pass
    def is_goal(self, state):
        pass
    def get_cost(self, state, action):
        pass
    def get_score(self, state):
        pass
    def is_equal(self, state1, state2):
        pass
    def show_state(self, state):
        pass
    def return_state(self, state):
        pass
    def mutate(self, x, mean, var):
        pass
    def reproduce(self, state1, state2):
        pass
    def fitness (self, state):
        pass
    def cost (self, state):
        pass
```

شکل ۵-۱۴ تعریف کلاس ProblemClass

۵-۴-۱-۲ کلاس ExTSP_Problem

در این کلاس، به پیاده‌سازی توابع تعریف شده در کلاس والد یعنی ProblemClass می‌پردازیم.

```
class ExTSP_Problem(ProblemClass):
    global MST_class
    MST_class = MinSpanTreeClass()

    def __init__(self, c1_arr, c2_arr):
        some_code=1

    def get_initial_state(self):
        some_code = 1

    def get_initial_state_random(self):
        some_code = 1

    def mutate (self, state):
        some_code = 1
    ...
```

شکل ۵-۱۵ پیاده سازی توابع در کلاس ExTSP_Problem که کلاس ProblemClass والد آن است به این طریق صورت می‌گیرد.

۵-۴-۱-۳ کلاس GeneticAlgorithm

در این کلاس الگوریتم ژنتیک پیاده‌سازی می‌شود. در این کلاس تابع الگوریتم ژنتیک صدازده می‌شود که به صورت زیر تعریف شده است:

```
def genetic_alg(self):
    self.initialization()
    for i in range(0, self.max_iteration):
        print("iteration ", i)
        self.iterate()
    print("result: ")
    best_node = max(self.people, key=attrgetter("fitness"))
    print(self.my_problem.return_state(best_node.state), " cost:",
str(best_node.cost))
    return best_node
```

شکل ۵-۱۶ تابع الگوریتم ژنتیک

در این الگوریتم، ابتدا تنظیمات اولیه انجام می‌شود. سپس به تعداد max_iteration تعریف شده در کلاس، نسل‌های جدید تولید می‌شود. در انتها، اطلاعات مربوط به نسل‌های مختلف نشان‌داده می‌شود.

تابع iterate

```
def iterate (self):
    new_generation = []
    for i in range (0, self.population):
        X=self.find_parent()
        Y=self.find_parent()
        tmp_state=self.my_problem.reproduce(X.state, Y.state)
        if (random.uniform(0,1) < self.mutation_probability ):
            tmp_state=self.my_problem.mutate(tmp_state)
        child= Node (tmp_state, self.my_problem.cost(tmp_state))
        new_generation.append(child)
    best_node=new_generation[0]
    best_fitness=new_generation[0].fitness
    worst_fitness=new_generation[0].fitness
    sum_fitness=0.0
    for i in range (0, self.population):
        tmp_val=new_generation[i].fitness
        if (tmp_val > best_fitness):
            best_node=new_generation[i]
            best_fitness=tmp_val
        if (tmp_val < worst_fitness):
            worst_fitness=tmp_val
        sum_fitness += tmp_val
    average_fitness = sum_fitness / self.population
    self.best_fitness.append(best_fitness)
    self.worst_fitness.append(worst_fitness)
    self.avrage_fitness.append(average_fitness)
```

شکل ۵-۱۷ تابع iterate

در این تابع، نسل جدید با توجه به نسل قبلی تولید می‌شود. تابع find_parent در خط ۴ و ۵ برنامه، باعث ایجاد والد می‌شود و سپس توسط تابع reproduce که در کلاس ExTSP_Problem تعریف شده‌است، فرزند تولید می‌شود. سپس فرزند با احتمال کمی، توسط تابع mutate، جهش پیدا می‌کند. پس از تولید نسل جدید، از خط ۱۱ به بعد، اطلاعات آماری مربوط به فرزندان به مجموعه داده‌ها اضافه می‌شود.

تابع reproduce

یکی از روش‌های تولید نسل جدید، استفاده از درخت پوشای کمینه است. به این ترتیب که ابتدا با توجه به یال‌های استفاده شده در دو مسیر والد، یک مجموعه از یال‌ها بوجود می‌آید. سپس تنها با استفاده از آن یال‌ها، درخت پوشای کمینه‌ای می‌سازیم. پس از تولید درخت، با استفاده از آن، سعی می‌کنیم مسیر فرزند را تولید کنیم. بدین ترتیب، میزان تاثیر والدین در تولید فرزند خود به میزان استفاده از یال‌های کوچکتر خواهد داشت.

```
1.  def reproduce(self, state1, state2):
2.      inf = self.infinity
3.      arr_size = len(cost1_arr)
4.      arr = np.full ((arr_size, arr_size), self.infinity, dtype=float)
5.      n = len(state1)
6.      for i in range (0, n):
7.          j = i + 1
8.          if j == n:
9.              j=0
10.         a = state1[i]
11.         b = state1[j]
12.         arr[(a, b)] = cost1_arr[(a,b)]
13.         arr [(b,a)] = cost1_arr[(a,b)]
14.     n = len(state2)
15.     for i in range(0, n):
16.         j = i + 1
17.         if j == n:
18.             j = 0
19.         a = state2[i]
20.         b = state2[j]
21.         arr[(a, b)] = cost1_arr[(a, b)]
22.         arr[(b, a)] = cost1_arr[(a, b)]
23.     global MST_class
24.     path = MST_class.do_all(arr)
25.     return path
```

شکل ۵-۱۸ تابع reproduce

در خطوط ۲ تا ۲۲، یال‌های مجاز با توجه به والدین استخراج می‌شود و سپس توسط تابع do_all در کلاس MST_class، مسیر جدید تولید می‌شود.

تابع muatation

این تابع به صورت تصادفی، جای دو عنصر در مسیر را با هم عوض می‌کند.

```
def mutate(self, state):
    new_state = state[:]
    n = len(new_state)
    i = random.randint(0, n - 1)
    j = random.randint(0, n - 1)
    new_state[i] = state[j]
    new_state[j] = state[i]
    return new_state
```

شکل ۵-۱۹ تابع mutation

۵-۴-۱-۴ کلاس Simulated_annealing

برای ایجاد الگوریتم سردکردن تدریجی، از کلاس Search استفاده کرده‌ایم. این کلاس دارای توابع مختلف برای جستجو است. یکی از این الگوریتم‌ها، الگوریتم Simulated_annealing است. برای استفاده از این تابع، نیازمند تعریف کلاسی به نام exTSP_action هستیم.

کلاس exTSP_action

در این کلاس، حرکت‌های مورد نظر انجام می‌شود. همانطور که در قبلا اشاره شد، ۴ نوع حرکت گفته شد. که ما در این قسمت به پیاده‌سازی آن حرکت‌ها می‌پردازیم.

```
class ExTSP_action:
    def __init__(self, index, action_type ,new_node):
        self.index = index
        self.new_node = new_node
        self.action_type = action_type
```

شکل ۵-۲۰ کلاس ExTSP_action

تابع Simulated_annealing

در این کلاس، در خط ۱۸، مجموعه‌ای از حرکات را گرفته و در خط ۲۰، یکی از این حرکات را به تصادف انتخاب می‌کنیم. در خط ۲۴ تا ۳۶، با توجه به دما و نتیجه گرفته شده، حالت کنونی را به‌روزرسانی می‌کنیم.

```
1. def simulated_annealing(self, schedule_number):
```

```
    self.initial_before_start_search()
    tmp_state = self.my_problem.get_initial_state_random()
    current_node = Node(tmp_state, 0,
self.my_problem.get_score(tmp_state), None)
    self.path.append(current_node)
    t = 0
    best_node = None
    while (t < self.max_iteration):
        temperature = self.schedule(t, schedule_number)
        if (temperature == 0):
            print("temp is 0 and result is ")
            self.show_result(current_node, True)
            return current_node
        if (self.my_problem.is_goal(current_node.state) == True):
            print("we found the goal")
            self.show_result(current_node, False)
            return current_node
        current_actions =
self.my_problem.get_actions(current_node.state)
        rand = randint(0, len(current_actions) - 1)
        result_state = self.my_problem.get_result(current_node.state,
current_actions[rand])
        tmp_node = Node(result_state,
self.my_problem.get_cost(current_node.state, current_actions[rand]),
                        self.my_problem.get_score(result_state),
current_node)
        self.node_created += 1
        self.my_problem.show_state(tmp_node.state)
        if tmp_node.score >= current_node.score:
            current_node = tmp_node
            self.path.append(current_node)
            self.node_expanded += 1
        else:
            r = random.uniform(0, 1)
            delta_e = tmp_node.score - current_node.score
            probability = math.exp(delta_e / temperature)
            print("probability: ", probability)
            if r < probability:
                current_node = tmp_node
                self.path.append(current_node)
                self.node_expanded += 1
        t += 1
    print("max iteration exceeded and result is:")
    self.show_result(current_node, True)
    return current_node
```

شکل ۵-۲۱ تابع سردکردن تدریجی (Simulated Annealing)

۵-۱-۴-۵ کلاس ILP_Solver

این کلاس برای حل مساله با استفاده از روش برنامه ریزی خطی صحیح است. در این کلاس تابع solve را تعریف کرده و از آن برای حل استفاده می‌کنیم.

کلاس solve

با توجه به روابط ۳-۹ و ۳-۱۷ در خط ۶ مقادیر x_{ij} تعریف می‌شوند. در خط ۷ مقادیر x'_{ij} و در خط ۸، متغیر f_{ij} تعریف می‌شود. در خط ۱۰ تابع هدف مشخص می‌شود. سپس قیود در خطوط بعدی مشخص می‌شود. در خط ۳۱، با استفاده از کتابخانه pulp، مساله حل می‌شود. در صورتی که مساله به جواب بهینه برسد، در خط ۳۳، کلمه Optimal چاپ می‌شود.

```
1.  def solve(self):
2.      node_index = self.node_index
3.      node_no = self.node_no
4.      cost1_arr = self.cost1_arr
5.      cost2_arr = self.cost2_arr
6.      x1 = pulp.LpVariable.dicts("Xij", [(i, j) for i in node_index for j
    in node_index], 0, 1, pulp.LpBinary)
7.      x2 = pulp.LpVariable.dicts("X'ij", [(i, j) for i in node_index for j
    in node_index], 0, 1, pulp.LpBinary)
8.      f = pulp.LpVariable.dicts("Flow(i,j)", [(i, j) for i in node_index
    for j in node_index], 0, node_no - 1)
9.      prob = pulp.LpProblem("Extended TSP", pulp.LpMinimize)
10.     prob += pulp.lpSum(
            cost1_arr[(i, j)] * x1[(i, j)] + cost2_arr[(i, j)] * x2[(i, j)]
11. for i in node_index for j in node_index)
12.     for i in node_index:
13.         prob += pulp.lpSum(x1[(i, j)] + x2[(i, j)] - x1[(j, i)] - x2[(j,
    i)]
14.  for j in node_index) == 0
15.     for j in node_index:
16.         prob += pulp.lpSum(x1[(i, j)] + x2[(i, j)] for i in node_index)
    >= 1
17.     for i in node_index:
18.         prob += x1[(i, i)] == 0
19.         prob += x2[(i, i)] == 0
20.         prob += f[(i, i)] == 0
21.
22.     for i in range(0, node_no):
23.         for j in range(0, node_no):          if not i == j:
24.
25.                 prob += x2[(i, j)] <= x1[(j, i)]
26.                 prob += x1[(i, j)] + x2[(i, j)] <= 1
27.                 prob += f[(i, j)] <= (x1[(i, j)] + x2[(i, j)]) *
    (node_no - 1)
28.                 prob += x1[(i, j)] + x1[(j, i)] <= 1
29.     for i in range(1, node_no):
30.         prob += pulp.lpSum(f[(j, i)] - f[(i, j)] for j in node_index) ==
    1
31.     prob.solve()
32.     edge_list = []
33.     print(pulp.LpStatus[prob.status])
34.     for i in node_index:
35.         for j in node_index:
36.             print(str(x1[(i, j)].varValue) + ",", end='')
37.             if x1[(i, j)].varValue == 1.0:
38.                 edge_list.append((i, j))
39.         print()
40.     print()
41.     for i in node_index:
```

```
        for j in node_index:
            print(str(x2[(i, j)].varValue) + ",", end='')
            if x2[(i, j)].varValue == 1.0:
                edge_list.append((i, j))
        print()
    print("The cost is", pulp.value(prob.objective))
    return edge_list
```

شکل ۵-۲۲ تابع solve

۵-۵ بررسی عملکرد الگوریتم‌ها

در این قسمت به تحلیل عملکرد الگوریتم‌ها می‌پردازیم.

۵-۵-۱ الگوریتم ژنتیک

ابتدا عملکرد الگوریتم ژنتیک را به صورت جداگانه بررسی می‌کنیم. با توجه به اینکه ممکن است خروجی الگوریتم ژنتیک به ازای اجرا های مختلف متفاوت باشد، ما به ازای هر حالت، الگوریتم را چند بار اجرا کرده و نتایج تمام این اجراها را آورده ایم.

۵-۵-۱-۱ تاثیر تکرار های متوالی

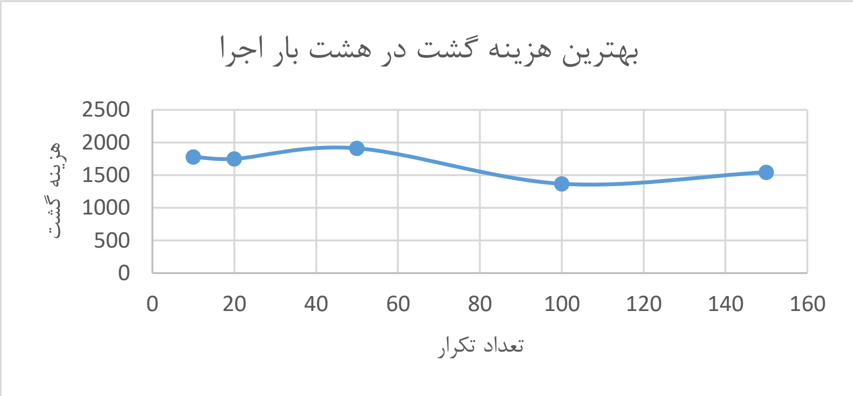
اولین بررسی، بررسی تاثیر تعداد تکرار[۴۵] در عملکرد الگوریتم است. بدین منظور ما به ازای هر حالت تکرار، الگوریتم را هشت بار اجرا کرده و نتایج این هشت اجرا، میانگین اجراها، بهترین و بدترین نتیجه در هشت اجرا را در جدول ۵-۱ نشان داده‌ایم.

تکرار	1	2	3	4	5	6	7	8	بهترین نتیجه	میانگین نتایج	بدترین نتیجه
10	3096	4264	2818	2208	2E+08	3576	2216	1780	1780	25002798	2E+08
20	1930	1750	2770	1848	2E+08	2064	4194	2380	1750	25002395	2E+08
50	2920	1912	1964	3097	1952	2551	2000	2250	1912	2330.75	3097
100	2226	2832	2488	1690	2784	1368	2178	1806	1368	2171.5	2832
150	2540	2104	1916	2354	1546	2071	1927	1920	1546	2047.25	2540

جدول ۵-۲ نتیجه به ازای تکرارهای مختلف. در این جدول، اعداد نشان دهنده خروجی الگوریتم ژنتیک (هزینه گشت) هستند. سطرها مشخص‌کننده تعداد تکرارهای انجام شده در اجرای الگوریتم‌ها است. ستون ۱ تا ۸، نشان‌دهنده ۸ اجرای مختلف هستند. (به ازای هر حالت تکرار، هشت بار متفاوت الگوریتم ژنتیک اجرا شده و نتایج در جدول آمده است.) در ستون بهترین نتیجه، میانگین نتایج و بدترین نتیجه به ترتیب بهترین نتیجه از بین این ۸ نتیجه، میانگین این ۸ نتیجه و بدترین نتیجه از بین این ۸ اجرا آمده‌است.

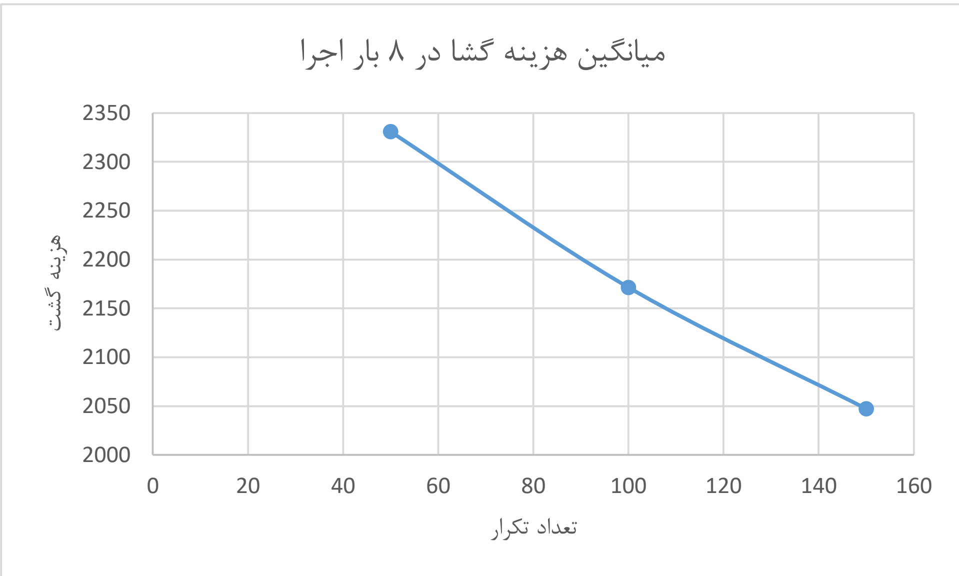
[45] iteration

در شکل ۵-۲۳ نمودار تغییرات بهترین نتیجه بدست آمده (در بین هشت اجرا) به ازای تکرارهای مختلف آمده‌است.



شکل ۵-۲۴ نمودار بهترین نتیجه به ازای تکرارهای متفاوت. در این نمودار محور افقی تعداد تکرار است و محور عمودی هزینه گشت است. نقاط داخل نمودار مشخص کننده بهترین خروجی الگوریتم در ۸ اجرا به ازای آن مقدار از تکرار است. برای مثال نقطه (20, 1750) یعنی بهترین هزینه گشت در ۸ اجرا زمانی که تعداد تکرار ۲۰ باشد، برابر با ۱۷۵۰ است.

در شکل ۵-۲۵ نمودار میانگین هشت اجرا به ازای تکرارهای مختلف آمده است.



شکل ۵-۲۶ نمودار میانگین نتایج-تکرار. در این نمودار محور افقی تعداد تکرار است و محور عمودی هزینه گشت است. نقاط داخل نمودار مشخص کننده میانگین خروجی الگوریتم در ۸ اجرا به ازای آن مقدار از تکرار است.

با توجه به شکل ۵-۲۶ و جدول ۵-۲ افزایش تکرار باعث بهبود عملکرد الگوریتم می‌شود. اما افزایش تکرار باید تا جایی ادامه پیدا کند که از لحاظ زمانی به صرفه باشد. مقدار ۱۰۰ را برای تکرار انتخاب می‌کنیم.

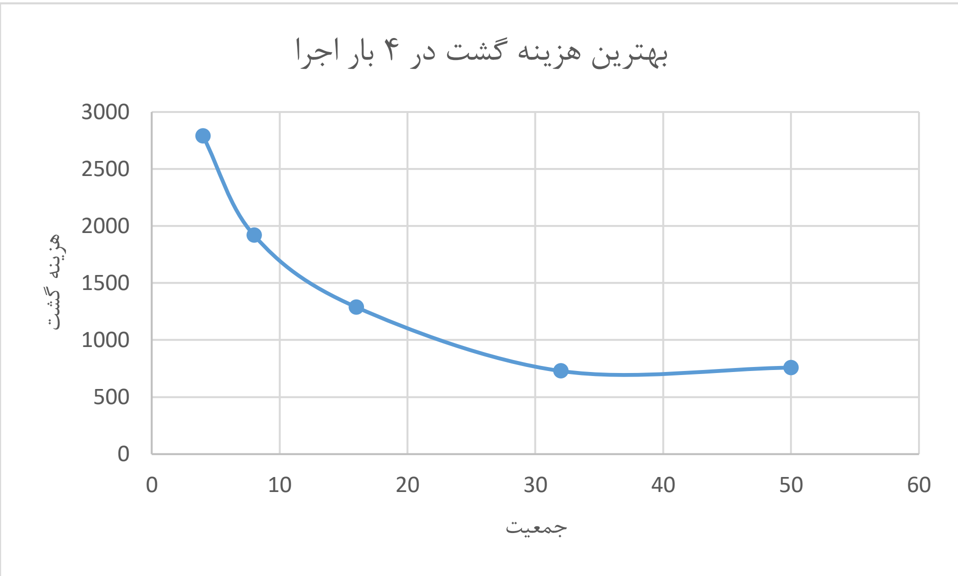
۵-۵-۱-۲ تاثیر جمعیت بر خروجی

در این قسمت به تاثیر جمعیت بر خروجی می‌پردازیم. بدین منظور به ازای جمعیت‌های مختلف، الگوریتم ژنتیک را چهار مرتبه اجرا کردیم. و نتایج بدست آمده را در جدول ۵-۳ آورده ایم.

جمعیت	1	2	3	4	بهترین نتیجه	میانگین نتایج	بدترین نتیجه
4	2790	3022	3588	3372	2790	3193	3588
8	2504	1918	2335	2706	1918	2365.75	2706
16	1544	1720	1288	1446	1288	1499.5	1720
32	826	1010	855	728	728	854.75	1010
50	840	779	886	758	758	815.75	886

جدول ۵-۴ نتیجه به ازای جمعیت‌های مختلف. در این جدول، مقادیر نشان دهنده خروجی الگوریتم (هزینه گشت) هستند. و ردیف‌ها نشان‌دهنده‌ی جمعیت الگوریتم در آن اجرا است. ستون ۱ تا ۴، نشان‌دهنده ۴ اجرای مختلف هستند. (به ازای هر حالت تکرار، ۴ بار متفاوت الگوریتم ژنتیک اجرا شده و نتایج در جدول آمده است.) در ستون بهترین نتیجه، میانگین نتایج و بدترین نتیجه به ترتیب بهترین نتیجه از بین این ۴ نتیجه، میانگین این ۴ نتیجه و بدترین نتیجه از بین این ۴ اجرا آمده‌است.

در شکل ۵-۲۷ به ازای جمعیت‌های متفاوت، بهترین نتیجه چهار اجرا را در نمودار آورده‌ایم. این نمودار تاثیر افزایش جمعیت در بهبود عملکرد الگوریتم را نشان می‌دهد.



شکل ۵-۲۸ نمودار تاثیر جمعیت بر روی بهترین نتیجه. در این نمودار محور افقی جمعیت است و محور عمودی هزینه گشت است. نقاط داخل نمودار مشخص کننده بهترین خروجی الگوریتم در ۴ اجرا به ازای آن مقدار از جمعیت است.

۵-۵-۱-۳ بررسی احتمال رخدادن جهش

در این قسمت به بررسی میزان تاثیر متغیر "احتمال رخدادن جهش" می‌پردازیم. ما به ازای هر مقدار متفاوت از "احتمال رخدادن جهش"، الگوریتم را چهار مرتبه اجرا کرده و نتایج را در جدول ۵-۵ نشان داده‌ایم.

احتمال جهش	1	2	3	4
0.1	712	772	3490	952
0.4	840	779	886	758
0.5	812	884	1338	817
0.8	864	761	712	744
1	915	918	780	716

جدول ۵-۶ تاثیر متغیر احتمال جهش بر روی خروجی. در این جدول، مقادیر نشان دهنده خروجی الگوریتم (هزینه گشت) هستند. و ردیف‌ها نشان‌دهنده‌ی مقدار متغیر احتمال جهش در آن اجرا است. ستون ۱ تا ۴، نشان‌دهنده ۴ اجرای مختلف هستند. (به ازای هر حالت تکرار، ۴ بار متفاوت الگوریتم ژنتیک اجرا شده و نتایج در جدول آمده است.)

با توجه به جدول ۵-۶ تاثیر احتمال جهش[۴۶] به خروجی، تاثیر مشخصی نیست. به نظر می‌رسد واریانس جواب در زمانی که احتمال جهش برابر با ۰,۵ باشد، کمترین است.

۵-۵-۲ الگوریتم سردکردن تدریجی

در این قسمت به بررسی عملکرد الگوریتم سردکردن تدریجی می‌پردازیم.

[46] Mutation Probability

۵-۵-۲-۱ بررسی تعداد تکرار در نتیجه الگوریتم

در این قسمت، به ازای تعداد تکرار مختلف، الگوریتم را چهارمرتبه اجرا کرده و نتایج را در جدول ۵-۷ آورده‌ایم.

تعداد تکرار	1	2	3	4	میانگین
500	3073	3250	3128	2957	3102
2000	1946	2298	2638	2306	2297
3000	2030	1829	1587	1777	1805.75

جدول ۵-۸ تاثیر تکرار بر عملکرد سردکردن تدریجی. در این جدول، مقادیر نشان دهنده خروجی الگوریتم (هزینه گشت) هستند. و ردیف‌ها نشان‌دهنده‌ی تعداد تکرار در آن اجرا است. ستون ۱ تا ۴، نشان‌دهنده ۴ اجرای مختلف هستند. (به ازای هر حالت تکرار، ۴ بار متفاوت الگوریتم سردکردن تدریجی اجرا شده و نتایج در جدول آمده است.) در ستون میانگین میانگین خروجی این ۴ اجرا آمده‌است.

به ازای تکرارهای متفاوت، میانگین نتیجه چهار اجرا را در نمودار شکل ۵-۲۹ آورده‌ایم. همانطور که می‌بینیم افزایش تکرار تاثیر زیادی در بهبود عملکرد الگوریتم داشته‌است.

شکل ۵-۳۰ نمودار تاثیر تکرار بر خروجی سردکردن تدریجی. در این نمودار محور افقی تعداد تکرار است و محور عمودی هزینه گشت است. نقاط داخل نمودار مشخص کننده بهترین خروجی الگوریتم در ۴ اجرا به ازای آن مقدار از تکرار است.

۵-۵-۳ مقایسه این دو الگوریتم

حال به مقایسه عملکرد این دو الگوریتم با هم می‌پردازیم. برای این کار، جواب این دو الگوریتم را با جواب حاصل از مساله‌ی آی‌ال‌پی مقایسه می‌کنیم. زیرا آی‌ال‌پی جواب بهینه را به ما می‌دهد. در این روش، هرکدام از الگوریتم‌ها را ۴ بار اجرا کرده و میانگین و بهترین نتیجه را در جدول زیر آورده ایم.

در ابتدا، مقایسه عملکرد الگوریتم‌ها در گرافی با ۵ راس صورت گرفته است. نتایج در این گراف بسیار به هم نزدیک بوده است و به جواب بهینه نیز نزدیک بوده است. در این حالت، سرد کردن تدریجی به جواب بهینه رسیده است.

	1	2	3	4	میانگین نتایج	بهترین نتیجه
Genetic Algorithm	107	107	107	107	107	107
Simulated Annealing	106	108	215	143	143	106
Integer Linear Programming	106	106	106	106	106	106

جدول ۵-۹ خروجی گراف ۵ راسه. در این جدول، مقادیر نشان دهنده خروجی الگوریتم بر روی گراف ۵ راسه هستند. و ردیف‌ها نشان‌دهنده‌ی الگوریتم‌های استفاده شده هستند. ستون ۱ تا ۴، نشان‌دهنده ۴ اجرای مختلف هستند. (هر الگوریتم را ۴ بار متفاوت اجرا کردیم و خروجی را در ۴ ستون متفاوت گزارش کرده ایم.) در ستون بهترین نتیجه و میانگین نتایج به ترتیب بهترین نتیجه از بین این ۴ نتیجه و میانگین این ۴ نتیجه آمده‌است.

در این قسمت، مقایسه عملکرد الگوریتم‌ها در گرافی با ۱۰ راس صورت گرفته است. در این حالت، الگوریتم سردکردن تدریجی نسبت به الگوریتم ژنتیک عملکرد بهتری داشته است. اما هیچ کدام از این دو الگوریتم به جواب بهینه نرسیده اند.

	1	2	3	4	میانگین نتایج	بهترین نتیجه
Genetic Algorithm	453	554	464	490.3333	490.3333	453
Simulated Annealing	551	404	507	487.3333	487.3333	404
Integer Linear Programming	385	385	385	385	385	385

جدول ۵-۱۰ خروجی بر روی گراف با ۱۰ راس. در این جدول، مقادیر نشان دهنده خروجی الگوریتم بر روی گراف ۱۰ راسه هستند. و ردیف‌ها نشان‌دهنده‌ی الگوریتم‌های استفاده شده هستند. ستون ۱ تا ۴، نشان‌دهنده ۴ اجرای مختلف هستند. (هر الگوریتم را ۴ بار متفاوت اجرا کردیم و خروجی را در ۴ ستون متفاوت گزارش کرده ایم.) در ستون بهترین نتیجه و میانگین نتایج به ترتیب بهترین نتیجه از بین این ۴ نتیجه و میانگین این ۴ نتیجه آمده‌است.

در این قسمت، مقایسه عملکرد الگوریتم‌ها در گرافی با ۱۵ راس که تعداد یال‌ها در آن گراف کم است، صورت گرفته است.

	1	2	3	4	5	6	7	میانگین نتایج	بهترین نتیجه
Genetic Algorithm	1077	937	1095	1526	1180	927	1108	1121.429	927
Simulated Annealing	100001005	1451	1202	884	1083	1E+08	1E+08	42858244	884
Integer Linear Programming	740	740	740	740	740	740	740	740	740

جدول ۵-۱۱ خروجی بر روی گراف با ۱۵ راس و با یال‌های کم. در این جدول، مقادیر نشان دهنده خروجی الگوریتم بر روی گراف ۱۵ راسه با یال‌های کم هستند. و ردیف‌ها نشان‌دهنده‌ی الگوریتم‌های استفاده شده هستند. ستون ۱ تا ۴، نشان‌دهنده ۴ اجرای مختلف هستند. (هر الگوریتم را ۴ بار متفاوت اجرا کردیم و خروجی را در ۴ ستون متفاوت گزارش کرده ایم.) در ستون بهترین نتیجه و میانگین نتایج به ترتیب بهترین نتیجه از بین این ۴ نتیجه و میانگین این ۴ نتیجه آمده‌است.

همانطور که در جدول ۵-۱۱ مشاهده می‌کنید، بهترین نتیجه الگوریتم سردکردن تدریجی، بهتر از بهترین نتیجه الگوریتم ژنتیک بوده است. اما تفاوت نتایج اجراهای مختلف در الگوریتم سردکردن بسیار زیاد است. میانگین نتایج در الگوریتم سردکردن تدریجی بسیار بالا است.

در این قسمت، مقایسه عملکرد الگوریتم‌ها در گرافی با ۱۵ راس که تعداد یال‌ها در آن گراف، زیاد است و گراف به گراف کامل شبیه است، انجام شده است.

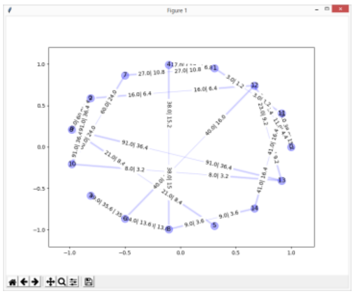
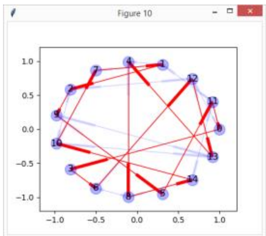
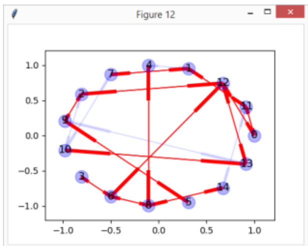
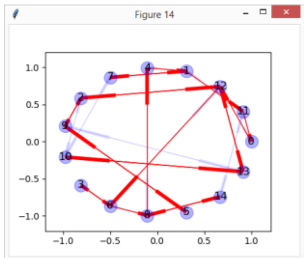
	1	2	3	4	5	6	میانگین نتایج	بهترین نتیجه
Genetic Algorithm	288	272	266	294	252	252	270.6667	252
Simulated Annealing	425	288	359	379	351	280	347	280
ILP	208	208	208	208	208	208	208	208

جدول ۵-۱۲ خروجی بر روی گراف با ۱۵ راس و با یال‌های زیاد. در این جدول، مقادیر نشان دهنده خروجی الگوریتم بر روی گراف ۱۵ راسه با یال‌های زیاد هستند. و ردیف‌ها نشان‌دهنده‌ی الگوریتم‌های استفاده شده هستند. ستون ۱ تا ۴، نشان‌دهنده ۴ اجرای مختلف هستند. (هر الگوریتم را ۴ بار متفاوت اجرا کردیم و خروجی را در ۴ ستون متفاوت گزارش کرده ایم.) در ستون بهترین نتیجه و میانگین نتایج به ترتیب بهترین نتیجه از بین این ۴ نتیجه و میانگین این ۴ نتیجه آمده‌است.

در این حالت، بر خلاف حالت گراف با یال کم، نتیجه الگوریتم ژنیتک بهتر از سرد کردن تدریجی بوده است. همچنین در الگوریتم سردکردن تدریجی تفاوت نتایج کم است.

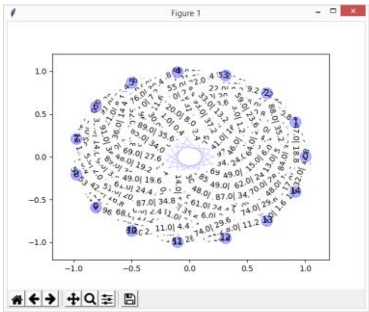
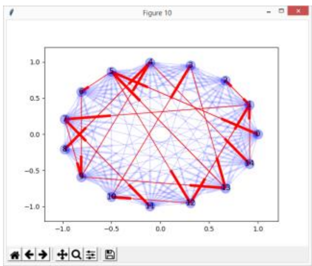
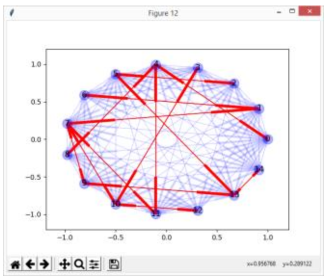
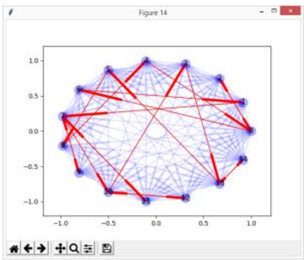
۵-۵-۴ مقایسه خروجی به صورت گراف

در این قسمت سعی شده‌است خروجی تصویری الگوریتم‌ها نشان داده‌شود. شکل ۵-۳۱ خروجی برای گراف با یال‌های کم را نشان می‌دهد.

		
		
Simulated Annealing	Genetic Algorithm	Integer Linear Programming

شکل ۵-۳۱ خروجی برای گراف با یال‌های کم. در ردیف اول، گراف و هزینه اولیه و ثانویه هر یال مشخص شده‌است و در ردیف دوم نتایج اجرای الگوریتم های متفاوت مشخص شده‌است. خطوط قرمز نشان‌دهنده مسیری هستند که الگوریتم مربوطه پیدا کرده است. در صورتی که در مسیر پیدا شده توسط الگوریتم، از هر دو جهت در یک یال استفاده شود، هر دو انتهای آن یال ضخیم است. در صورتی که فقط از یک یال در یک جهت استفاده شده باشد، ابتدای آن یال جهت دار ضخیم نیست و انتهای آن یال ضخیم است.

شکل ۵-۳۲ خروجی برای گراف با یال‌های زیاد را نشان می‌دهد.

		
		
Simulated Annealing	Genetic Algorithm	Integer Linear Programming

شکل ۵-۳۲ خروجی برای گراف با یال‌های زیاد. در ردیف اول، گراف و هزینه اولیه و ثانویه هر یال مشخص شده‌است و در ردیف دوم نتایج اجرای الگوریتم های متفاوت مشخص شده‌است. خطوط قرمز نشان‌دهنده مسیری هستند که الگوریتم مربوطه پیدا کرده است. همچنین اگر هر ضخامت انتهای هر یال قرمز مشخص کننده جهت آن یال است. در صورتی که در مسیر پیدا شده توسط الگوریتم، از هر دو جهت در یک یال استفاده شود، هر دو انتهای آن یال ضخیم است. در صورتی که فقط از یک یال در یک جهت استفاده شده باشد، ابتدای آن یال جهت دار ضخیم نیست و انتهای آن یال ضخیم است.

فصل ششم

جمع‌بندی و نتیجه‌گیری و پیشنهادات

۶- جمع‌بندی و نتیجه‌گیری و پیشنهادات

در این بخش به جمع بندی و نتیجه‌گیری پروژه و پیشنهادات مربوط به تحقیقات آینده می‌پردازیم.

۱-۶ جمع‌بندی و نتیجه‌گیری

هدف از این پروژه، آشنایی با نحوه پیاده‌سازی الگوریتم ژنتیک و سردکردن تدریجی برای حل مسائل واقعی بوده است. به همین منظور مساله‌ی تعمیم‌یافته فروشنده‌ی دوره‌گرد انتخاب شد. پس از بررسی مسائل مرتبط با مساله‌ی پروژه و بررسی راه‌حل های ارائه شده برای آن‌ها، سعی به حل مساله‌ی پروژه با الهام‌گرفتن از راه‌حل های قبلی شد.

پس از بررسی خروجی‌های حاصل از اجرای الگوریتم ژنتیک و سردکردن تدریجی و الگوریتم برنامه‌ریزی خطی صحیح، به نتایج زیر می‌رسیم.

- به طور میانگین جواب‌های حاصل از اجرای این دو الگوریتم به جواب بهینه نزدیک است.
- جواب‌های حاصل از اجرای الگوریتم سردکردن تدریجی بسیار متنوع‌تر است. اما امکان رسیدن به جواب بهینه با توجه به جدول ۵-۱۱ و جدول ۵-۱۲، بیشتر از الگوریتم ژنتیک بوده‌است. مخصوصا هنگامی که یال‌های گراف کم باشد.
- دلیل این تفاوت، این است که الگوریتم سردکردن تدریجی فرض خاصی در مورد ویژگی‌های گراف نکرده است. اما در الگوریتم ژنتیک جواب به صورت ضمنی به یک درخت نزدیک می‌شویم. در صورتی که هزینه بازگشت از هریال، بسیار ناچیز باشد، الگوریتم ژنتیک عملکرد بهتری دارد. در واقع حذف یال‌ها در مرحله reproduce باعث حذف یال‌های مربوط به دور در فرایند تکامل می‌شود.

در الگوریتم سردکردن تدریجی، اجرای زیاد باعث ایجاد بوجود آمدن یال‌های تکراری می‌شوند. می‌شود با استفاده از الگوریتم‌های خاص در هر مرحله، یال‌های اضافی را حذف کرد.

۶-۲ پیشنهادات

با توجه به اینکه الگوریتم ژنتیک به سرعت به جواب‌های نزدیک به بهینه نزدیک می‌شود و همچنین احتمال رسیدن به جواب بهینه، در الگوریتم سردکردن تدریجی بیشتر است، ممکن است بتوان این دو روش را با هم ترکیب کرد و به صورت هم‌روند این دو الگوریتم را اجرا کرد.

جواب برنامه ریزی خطی صحیح برای این الگوریتم ما را به جواب بهینه می‌رساند اما اجرای آن زمانبر است. می‌توان از برنامه‌ریزی خطی عادی که زمان اجرای چند جمله‌ای دارد، برای بهبود الگوریتم استفاده کرد.

با توجه به اینکه مساله‌ی مورد نظر برگرفته از مسائل طبیعی است، و اندازه هزینه در مسائل واقعی معمولا به فاصله اقلیدسی رئوس ربط دارد، استفاده از روش‌های هندسی برای بهبود این دو الگوریتم می‌تواند برای مسائل با فاصله اقلیدسی جواب بسیار بهتری به ما بدهد.

۷- منابع و مراجع

[1] G. Reinelt, The Traveling Salesman Computational Solutions for TSP Applications, Heidelberg: Springer-Verlag, 1994.

[2] D. Naddef and R. Giovanni , "The graphical relaxation: A new framework for the symmetric traveling salesman polytope," *Mathematical Programming,* vol. 58, no. 1-3, pp. 53-88, 1993.

[3] S. M. R. Modarresi, "Disaster vehicle routing problem," M.S. Thesis, Dep. of Computer Engineering and Information technology, AmirKabir Univ. of Tech., 2014.

[4] J. F. D. N. Gérard Cornuéjols, "The traveling salesman problem on a graph and some related integer polyhedra," *Mathematical programming,* vol. 33, no. 1, pp. 1-27, 1985.

[5] S. J. Russell, and P. Norvig, Artificial Intelligence A Modern Approach, Prentice hall, 2010.

[6] K. Bryant, "Genetic Algorithms and the Traveling Salesman Problem," *Department of Mathematics, Harvey Mudd College,* pp. 10-12, 2001.

[7] R. S. Pressman, Software Engineering a practitioner's approach, 7 ed., McGraw-Hill, 2010.

۸- پیوست‌ها

۸-۱ پیوست ۱: واژه‌نامه کاری انگلیسی – فارسی

Benchmark	محک
Boolean Satisfiability Problem	مساله‌ی ارضای گزاره‌های بولی
Chromosome	کروموزوم
Class Diagram	نمودار کلاس
Closed Walk	گردش بسته
Communication	ارتباط
Computational Complexity Theory	نظریه پیچیدگی محاسباتی
Construction	ساخت
Context Diagram	جریان داده سیستم
Decision Problem	مساله تصمیم‌گیری
Deployment	گسترش
Disaster Vehicle Routing Problem	مساله‌ی مسیریابی خودرو در محیط آسیب‌دیده
DNA Sequencing	ترتیب‌دهی دی‌ان‌ای
DVRP (Disaster Vehicle Routing Problem)	دی‌وی‌آرپی
Dynamic Programming	برنامه‌نویسی پویا
Fitness	مطلوب بودن
Genetic Algorithms	الگوریتم ژنتیک
Hamiltonian Cycle	دور همیلتنی
Hill Climbing	روش تپه نوردی ساده

ILP (Integer Linear Programming)	آی‌ال‌پی
Integer Linear Programming	مساله‌ی برنامه‌ریزی خطی صحیح
Iteration	تکرار
Knapsack Problem	مساله کوله پشتی
Linear	خطی
Linear Process Flow	جریان فرایند خطی
Logistics	علم منطق
Microchips	چیپ‌های کوچک
Minimum Spanning Tree (MST)	مساله‌ی درخت پوشای کمینه
Modeling	مدل‌سازی
Mutation	جهش
Mutation Probability	احتمال جهش
Nondeterministic Polynomial Time Complete	ان‌پی تمام
NP (Nondeterministic Polynomial)	ان‌پی
NP-Hard	ان‌پی‌سخت
Optimization Problems	مسائل بهینه‌سازی
Planning	برنامه‌ریزی
Planning	برنامه‌ریزی
Simulated Annealing	الگوریتم سردکردن تدریجی
Travelling Salesman Problem	مساله فروشنده‌ی دوره‌گرد
TSP (Travelling Salesman Problem)	تی‌اس‌پی
Use Case Diagram	مدل درخواست سیستم

Walk	گشت
Waterfall Process	مدل آبشاری

۸-۲ پیوست ۲: واژه‌نامه کاری فارسی – انگلیسی

احتمال جهش	Mutation Probability
ارتباط	Communication
الگوریتم ژنتیک	Genetic Algorithms
الگوریتم سردکردن تدریجی	Simulated Annealing
ان‌پی	NP (Nondeterministic Polynomial)
ان‌پی تمام	Nondeterministic Polynomial Time Complete
ان‌پی‌سخت	NP-Hard
آی‌ال‌پی	ILP (Integer Linear Programming)
برنامه‌ریزی	Planning
برنامه‌ریزی	Planning
برنامه‌نویسی پویا	Dynamic Programming
ترتیب‌دهی دی‌ان‌ای	DNA Sequencing
تکرار	Iteration
تی‌اس‌پی	TSP (Travelling Salesman Problem)
جریان داده سیستم	Context Diagram
جریان فرایند خطی	Linear Process Flow
جهش	Mutation
چیپ‌های کوچک	Microchips
خطی	Linear
دور همیلتنی	Hamiltonian Cycle
دی‌وی‌آرپی	DVRP (Disaster Vehicle Routing Problem)
روش تپه نوردی ساده	Hill Climbing
ساخت	Construction
علم منطق	Logistics
کروموزوم	Chromosome
گردش بسته	Closed Walk
گسترش	Deployment
گشت	Walk

محک	Benchmark
مدل آبشاری	Waterfall Process
مدل درخواست سیستم	Use Case Diagram
مدل‌سازی	Modeling
مساله تصمیم‌گیری	Decision Problem
مساله فروشنده‌ی دوره‌گرد	Travelling Salesman Problem
مساله کوله پشتی	Knapsack Problem
مساله‌ی ارضای گزاره‌های بولی	Boolean Satisfiability Problem
مساله‌ی برنامه‌ریزی خطی صحیح	Integer Linear Programming
مساله‌ی درخت پوشای کمینه	Minimum Spanning Tree (MST)
مساله‌ی مسیریابی خودرو در محیط آسیب‌دیده	Disaster Vehicle Routing Problem
مسائل بهینه‌سازی	Optimization Problems
مطلوب بودن	Fitness
نظریه پیچیدگی محاسباتی	Computational Complexity Theory
نمودار کلاس	Class Diagram

Amirkabir University of Technology
(Tehran Polytechnic)

Computer Engineering and Information Technology Department

BSc Thesis

Title of Thesis
Implementation and evaluation of Genetic and Simulated Annealing algorithms for extended travelling salesman problem

By
Ali Mortazavi

Supervisor
Dr. MohammadReza Razazi

October 2017