دانشگاه صنعتی شاهرود
دانشکده ریاضی
گروه ریاضی کاربردی

عنوان:

بررسی مسأله مکان‌یابی تدافعی و حل آن به روش‌های ابتکاری

استاید راهنما:
دکتر جعفر فتحعلی
دکتر نادر جعفری راد

استاد مشاور:
دکتر صادق رحیمی شعرباف

نگارنده:
محمد امید کاردانی

ارائه شده جهت اخذ درجه کارشناسی ارشد در رشته ریاضی کاربردی

مهرماه ۱۳۸۸


بسم الله الرحمن الرحيم

تعهد نامه

اینجانب ..... دانشجوی دوره کارشناسی ارشد / دکتری رشته .....

دانشکده ..... دانشگاه صنعتی شاهرود نویسنده پایان نامه/ رساله .....

..... تحت راهنمائی..... متعهد می شوم .

- تحقیقات در این پایان نامه/ رساله توسط اینجانب انجام شده است و از صحت و اصالت برخوردار است .
- در استفاده از نتایج پژوهشهای محققان دیگر به مرجع مورد استفاده استناد شده است .
- مطالب مندرج در پایان نامه/ رساله تاکنون توسط خود یا فرد دیگری برای دریافت هیچ نوع مدرک یا امتیازی در هیچ جا ارائه نشده است .
- کلیه حقوق معنوی این اثر متعلق به دانشگاه صنعتی شاهرود می باشد و مقالات مستخرج با نام « دانشگاه صنعتی شاهرود » و یا « **Shahrood University of Technology** » به چاپ خواهد رسید .
- حقوق معنوی تمام افرادی که در به دست آمدن نتایج اصلی پایان نامه / رساله تأثیرگذار بوده اند در مقالات مستخرج از پایان نامه/ رساله رعایت می گردد.
- در کلیه مراحل انجام این پایان نامه/رساله ، در مواردی که از موجود زنده (یا بافتهای آنها) استفاده شده است ضوابط و اصول اخلاقی رعایت شده است .
- در کلیه مراحل انجام این پایان نامه/رساله ، در مواردی که به حوزه اطلاعات شخصی افراد دسترسی یافته یا استفاده شده است اصل رازداری ، ضوابط و اصول اخلاق انسانی رعایت شده است .

تاریخ :

امضای دانشجو

مالکیت نتایج و حق نشر

- کلیه حقوق معنوی این اثر و محصولات آن (مقالات مستخرج ، کتاب ، برنامه های رایانه ای ، نرم افزار ها و تجهیزات ساخته شده است) متعلق به دانشگاه صنعتی شاهرود می باشد . این مطلب باید به نحو مقتضی در تولیدات علمی مربوطه ذکر شود .
- استفاده از اطلاعات و نتایج موجود در پایان نامه / رساله بدون ذکر مرجع مجاز نمی باشد.

*** متن این صفحه نیز باید در ابتدای نسخه های تکثیر شده پایان نامه/ رساله وجود داشته باشد .**

مدیریت تحصیلات تکمیلی

فرم شماره (۶)

شماره :
تاریخ :
ویرایش :

بسمه تعالی

فرم صورتجلسه دفاع پایان نامه تحصیلی دوره کارشناسی ارشد

با تأییدات خداوند متعال و با استعانت از حضرت ولی عصر (عج) جلسه دفاع از پایان نامه کارشناسی ارشد **آقای محمدامید کاردانی** رشته ریاضی گرایش کاربردی تحت عنوان **"بررسی مساله مکان یابی تدافعی و حل آن به روشهای ابتکاری"** که در تاریخ ۸۸/۸/۴ با حضور هیأت محترم داوران در دانشگاه صنعتی شاهرود برگزار گردید به شرح زیر است :

قبول (با درجه : عالی امتیاز ۱۸/۷۰ ☒) دفاع مجدد ☐ مردود ☐

۱ـ عالی (۲۰ ـ ۱۸) ۲ـ بسیار خوب (۱۷/۹۹ ـ ۱۶)

۳ـ خوب (۱۵/۹۹ ـ ۱۴) ۴ـ قابل قبول (۱۳/۹۹ ـ ۱۲)

a) عضو هیات داوران	نام ونام خانوادگی	مرتبهٔ علمی	امضاء
۱ـ استادراهنمای اول	دکتر جعفر فتحعلی	استادیار	
۲- استاد راهنمای دوم	دکتر نادر جعفری راد	استادیار	
۳ـ استاد مشاور	دکتر صادق رحیمی شعرباف	استادیار	
۴ـ نماینده شورای تحصیلات تکمیلی	دکتر احمد نزاکتی	استادیار	
۵ـ استاد ممتحن	دکتر محمدرضا پیغامی	استادیار	
۶ ـ استاد ممتحن	دکتر حجت احسنی طهرانی	استادیار	

تأیید رئیس دانشکده

مدیریت تحصیلات تکمیلی
فرم شماره (۹)

بسمه تعالی

فرم اصلاحات پایان نامه

شماره :
تاریخ :
ویرایش :

مدیر محترم تحصیلات تکمیلی دانشگاه

با سلام

احتراماً ، به استحضار می رساند که جلسه دفاع دانشجوی کارشناسی ارشد **محمد امید کاردانی** در تاریخ ۸۸/۸/۴ ساعت ۱۱ در حضور هیأت داوران تشکیل و ضمن موفقیت نامبرده در جلسه دفاع ، مقرر گردید که اصلاحات مورد نظر را حداکثر یک ماه انجام و نسخه نهایی را تحویل دانشکده نماید. بدیهی است تبعات ناشی از تأخیر بعهده دانشجو می باشد .

نماینده تحصیلات تکمیلی در جلسه دفاع :
دکتر احمد نزاکتی

امضاء

رونوشت: دانشجو جهت اطلاع و اقدام لازم

پدر و مادر عزیزم:

شهباز گشودید تا پرواز را بیاموزم؛

ره پوی قله‌ها شدید تا صعود را تجربه کنم؛

شبنم وار گلبرگ وجودم را سیراب کردید تا به تکامل برسم؛

و خورشید گونه بر سبزه زار درونم تابیدید تا متعالی گردم؛

پس این پرواز، تجربه، تکامل و تعالی را در قالب صحیفه‌ای ارزشمند به نام پایان نامه کارشناسی ارشد در طبقی ذرین از گل‌های محبت و سپاس تقدیم قلوب پر مهرتان می‌نمایم.

نقش وجودتان بر حریر سبز زندگی مستدام باد.

تقدیر و تشکر

اینک که در طی طریق علم اندوزی و فرزانگی گامی دگر پیمودم و با موفقیت دوره کارشناسی ارشد خود را به اتمام رسانده و نتیجه تلاش خود را در قالب این پایان نامه ارائه کردم، صمیمانه قدردان راهنمایی‌ها و تلاش‌های کسانی هستم که مرا در این مسیر یاری رساندند؛

اساتید گرانقدر، دکتر فتحعلی و دکتر جعفری راد و دکتر رحیمی شعرباف؛

ریاست محترم دانشکده ریاضی دانشگاه صنعتی شاهرود، دکتر نزاکتی؛

و سایر اساتید محترم دانشکده ریاضی؛

دوست عزیز و یار شفیق و همکار عالی قدرم، جناب آقای سید سعید محمودی هاشمی؛

و در انتها، برادرم، عزیزم، راهنمایم و مشوقم، نوید؛

والسلام

محمد امید کاردانی

پیشگفتار

بسیاری از دانشمندان ریاضیات را مادر علوم می‌نامند. در واقع کاربرد ریاضیات در تمام علوم، حتی علوم پایه دیگر و تاثیر چشمگیر آن در پیشرفت و گسترش علم در زمینه‌های مختلف بسیاری از اندیشمندان را وادار ساخت تا این حقیقت را تایید کنند.
از ابتدای پیدایش علوم ریاضی، در کنار توسعه گام به گام این علم، کاربردهای وسیعی از آن در پیشبرد سایر علوم و همچنین کاربردهای چشمگیر آن در زندگی بشر به سرعت توسعه یافت.
در دو شاخه اصلی محض و کاربردی، ریاضیات توانسته گام‌های زیادی را در طول دوره حیاتش طی کند. گام‌هایی را که گاهاً حتی باور نکردنی بودند.
اما ریاضیات کاربردی، جایی است که علوم ریاضیات توانسته با سایر علوم ارتباط برقرار کند. در واقع، ریاضیات کاربردی وظیفه به کارگیری تمامی مفاهیم توسعه یافته در ریاضیات محض را در زندگی بشر بر عهده دارد و هدف والای آن همانند سایر علوم کاربردی، چیزی نیست جز ارتقا کیفیت زندگی بشر. مسایل زیادی در ریاضیات کاربردی تا کنون برای دانشمندان این علم مطرح شده‌اند که بسیاری از آن‌ها به روش‌های گوناگون مورد بررسی قرار گرفته و حل شده‌اند. مسایلی که اکثر آن‌ها ریشه در موضوعات مرتبط با زندگی بشر داشته‌اند.
ما نیز در این تحقیق می‌خواهیم به سراغ دسته وسیعی از این مسایل برویم. مسایلی که به خاطر هدفش به «مسایل مکان‌یابی تسهیلات[1]» شهرت دارند. در این مسایل هدف تعیین مکان تعدادی تسهیلات تحت شرایطی خاص می‌باشد. اما شرایط خاصی که

[1] Facility Location Problem

برای این مکان‌یابی در نظر گرفته می‌شوند، باعث پدید آمدن انواع خاصی از این مسأله می‌شوند. «مکان‌یابی تسهیلات روی صفحه[۲]»، «مکان‌یابی تسهیلات روی شبکه[۳]»، «مکان‌یابی تک‌وسیله‌ای یا چندوسیله‌ای[۴]»، «مکان‌یابی و تخصیص تک یا چندوسیله‌ای[۵]»، «مکان‌یابی تسهیلات رقابتی[۶]» و ...انواع خاصی از مسایل مکان‌یابی تسهیلات می‌باشند. مسایل مکان‌یابی به طور گسترده‌ای توسط محققین و ریاضی‌دانان مورد مطالعه قرار گرفته‌اند.

همان طور که اشاره شد، نوع خاصی از مسایل مکان‌یابی تسهیلات، «مسأله مکان‌یابی تسهیلات رقابتی» می‌باشد. در این مسأله، مکان‌یابی تسهیلات با توجه به مکان سایر تسهیلات رقابتی انجام می‌شود. این مسأله در فصل ۱ بیشتر معرفی می‌شود.

در سال ۲۰۰۸ نوع جدیدی از مسایل مکان‌یابی تسهیلات رقابتی، به نام «مسأله مکان‌یابی تدافعی[۷]» (DLP) توسط تاکشی اونو و هیدکی کاتاگیری [۲۸] معرفی و بسط داده شد.

در این مسأله، یک مدافع قصد دارد تسهیلات تدافعی خود را به گونه‌ای در یک شبکه مکان‌یابی کند که بتواند از یک مکان استراتژیک به نام هسته[۸] که خود یکی از راس‌های شبکه است در مقابل یک مهاجم که او نیز در یکی از راس‌های شبکه جای دارد، محافظت کند.

همان طور که از تعریف کلی مسأله و هدف آن پیداست، DLP می‌تواند در کاربردهای گوناگون در نظر گرفته شود. بدون شک اولین کاربردی که DLP به ذهن القا می‌کند در مسایل نظامی و دفاعی است. محافظت از سایت‌های استراتژیک در برابر دشمنان و کاهش بیشتر احتمال صدمه دیدن آن‌ها در برابر هجوم مهاجمین از دغدغه‌های اصلی در علوم نظامی است.

---

[۲] Facility Location on a Plane
[۳] Facility Location on a Network
[۴] Single- or Multi-Facility Location
[۵] Single- or Multi-Facility Location and Allocation
[۶] Competitive Facility Location
[۷] Defensive Location Problem
[۸] Core

توجه می‌کنیم که در بسیاری از کشورهای جهان، بودجه هنگفتی صرف مسایل نظامی و دفاعی می‌شود. با توجه به پیشرفت علوم دفاعی در سرتاسر دنیا و رقابت کشورها در زمینه‌های نظامی و دفاعی و از این رو، تحمیل شدن هزینه‌های روزافزون برای افزایش قدرت دفاعی به کشورهای گوناگون، ریاضیات و به ویژه علوم بهینه‌سازی توانسته راه خود را در علوم دفاعی نیز پیدا کند. با توجه به آن چه گفته شد، ضرورت مطالعه و مسأله مکان‌یابی تدافعی DLP که مستقیما مسأله بهینه‌سازی دفاع از مکان‌های استراتژیک با توجه به هزینه‌های موجود را مورد بررسی قرار می‌دهد، آشکار خواهد بود. علاوه بر کاربردهای نظامی و دفاعی، کاربرد DLP در بعضی زمینه‌های دیگر نیز می‌تواند ثمربخش باشد. چنین سیستم‌های دفاعی در بازی‌های گروهی، مانند فوتبال، بسکتبال و ...می‌تواند به عنوان کاربرد دیگری از DLP مد نظر قرار گیرد. با توجه به تبدیل شدن ورزش‌هایی همانند فوتبال به یک علم و صرف هزینه‌های هنگفت برای رقابت‌های ورزشی، باز هم اهمیت بهینه‌سازی ریاضی بیش از پیش واضح و آشکار خواهد شد.

سایر زمینه‌هایی که DLP می‌تواند به کار گرفته شود عبارتند از: اتخاذ استراتژی‌های امنیتی و حفاظتی در سیستم‌ها و شبکه‌های کامپیوتری، حفظ امنیت اماکن اجتماعی یا خصوصی و ...

چکیده

آن چه در تحقیق حاضر خواهد آمد، به قرار زیر است:
ابتدا در فصل اول، تعاریف، مفاهیم و توضیحاتی را که در روند تحقیق به کار گرفته خواهند شد و آشنایی با آن‌ها برای مطالعه و درک مؤثر مطالب این تحقیق ضروری خواهند بود، ارائه می‌کنیم.
سپس در فصل ۲، مسأله مکان‌یابی تدافعی DLP به طور کامل معرفی و مدل‌سازی می‌شود. در ادامه در فصول ۳ و ۴، به سراغ حل مسأله DLP خواهیم رفت. ابتدا در فصل ۳، پس از معرفی اجمالی روش جستجوی ممنوعه[۹] (TS) به بررسی روش حل DLP بوسیله TS که توسط اونو و کاتاگیری [۲۸] ارائه شد، خواهیم پرداخت و در انتها نتایج محاسباتی را ارائه می‌کنیم. روند مشابهی در فصل ۴، با به کارگیری روش جستجوی همسایگی متغیر[۱۰] (VNS) صورت خواهد گرفت. در این فصل پس از معرفی VNS، الگوریتمی ابتکاری را بر مبنای VNS برای حل DLP ارائه خواهیم کرد و در انتها به ارائه نتایج محاسباتی خواهیم پرداخت.
در ادامه مطالعه، در فصل ۵، نوع جدیدی از مسأله DLP را معرفی و مدل‌سازی خواهیم کرد. در این مسأله جدید، فرض کرده‌ایم که مدافع می‌تواند هر کسری از یک توان دفاعی کل را در راس‌های شبکه قرار دهد. این مسأله جدید که از سویی تعمیم DLP و از سوی دیگر پیوسته‌سازی متغیرهای DLP است را «مسأله مکان‌یابی تدافعی

[۹] Tabu Search Algorithm
[۱۰] Variable Neibourhood Search Method

تعمیم یافته[۱۱]» (GDLP) یا به طور معادل «مسأله مکان‌یابی تدافعی پیوسته[۱۲]» (CDLP) نامیده‌ایم.
پس از معرفی و مدل‌سازی CDLP، در فصل ٦، به ارائه روش حلی بر مبنای مکانیسم شبه الکترومغناطیس[۱۳] (EM) برای آن می‌پردازیم و نتایج محاسباتی را بیان خواهیم کرد. در انتها به مقایسه نتایج محاسباتی و نتیجه‌گیری کلی ، در فصل ۷، خواهیم پرداخت.

کلیدواژه:

مسأله مکانیابی ـ مکانیابی تدافعی ـ روش‌های بهینه سازی ابتکاری

[۱۱] Generalized Defensive Location Problem
[۱۲] Continuous Defensive Location Problem
[۱۳] Electromagnetism-like Mechanism

فهرست مندرجات

فصل ۱

مفاهیم اولیه

۱.۱ مسأله مکان‌یابی تسهیلات

مسأله مکان‌یابی تسهیلات[۱] (FLP) که به آن تحلیل مکان[۲] نیز می‌گویند، یک دسته از مسایل برنامه‌ریزی ریاضی است که به تعیین مکان تسهیلات در یک صفحه یا بر روی یک شبکه می‌پردازد. در این گونه مسایل، مکان تسهیلات باید به گونه‌ای تعیین شود که پاره‌ای از شرایط مطلوب برقرار باشد. در ساده‌ترین نوع FLP که به مسأله فرما—وبر[۳] معروف است، مکان یک وسیله باید به گونه‌ای تعیین شود که مجموع فواصل یک سری نقاط از آن وسیله حداقل شود. در مدل‌های پیچیده‌تر FLP، شرایط دیگری از قبیل مکان‌یابی چندین وسیله، محدودیت‌های بر روی مکان‌های جایگذاری و غیره می‌توانند مد نظر قرار گیرند.

مدل‌سازی پایه‌ای FLP، شامل مجموعه‌ای از مکان‌های بالقوه برای قرار دادن تسهیلات و مجموعه‌ای از نقاط تقاضا که می‌باید سرویس داده شوند، می‌باشد. هدف این مسأله انتخاب زیرمجموعه‌ای از مجموعه مکان‌ها برای قرار دادن یک وسیله در

[۱] Facility Location Problem
[۲] Location Analysis
[۳] Fermat-Weber Problem

آن‌ها و حداقل کردن مجموع فواصل نقاط تقاضا از نزدیک‌ترین سرویس دهنده خود و همچنین مجموعه هزینه‌های راه‌اندازی تسهیلات، می‌باشد.
اگر در FLP فرض شود که فاصله‌ها بین سرویس دهنده‌ها و نقاط تقاضا در نامساوی مثلث صدق نمی‌کنند، یا به عبارت دیگر تابع فاصله یک تابع متریک نیست، به آن مسأله مکان‌یابی تسهیلات غیرمتریک[۴] می‌گویند. در غیر این صورت، به آن مسأله مکان‌یابی تسهیلات متریک[۵] می‌گویند.

۲.۱ مسأله مکان‌یابی تسهیلات رقابتی (CFLP)

مسأله مکان‌یابی تسهیلات رقابتی (CFLP) یکی از مسایل مکان‌یابی برای تسهیلات تجاری، همانند فروشگاه‌ها، سوپر مارکت‌ها و ...می‌باشد. در این گونه مسایل، یک تصمیم گیرنده مکان تسهیلات خود را بر اساس مکان سایر تسهیلات رقابتی تعیین می‌کند و هدف او حدکثر کردن سود حاصل از مشتری‌هایی است که از تسهیلات استفاده می‌کنند.
مطالعه بر روی مسایل مکان‌یابی تسهیلات رقابتی اولین بار توسط هوتلینگ [۱۷] انجام شد. او مسایل CFLP را تحت شرایط زیر مورد مطالعه قرار داد:

۱) مشتری‌ها به طور یکنواخت روی یک پاره‌خط توزیع شده‌اند.

۲) هرکدام از تصمیم گیرنده‌ها می‌توانند تسهیلات خود را در هر زمانی مکان‌یابی و جابجا کنند.

۳) همه مشتری‌ها فقط از نزدیک‌ترین تسهیلات استفاده می‌کنند.

مسایل مکان‌یابی رقابتی روی صفحه توسط اوکابی و سوزوکی و همکاران [۲۲] مطالعه شد. همچنین، به عنوان توسعه‌ای از مسأله مکان‌یابی رقابتی هوتلینگ، وندل و

[۴] Non-metric Facility Location Problem
[۵] Metric Facility Location Problem

مک‌کلوی [۲۹] فرض را بر این گرفتند که مشتری‌ها بر روی تعداد متناهی نقطه، که به آن‌ها نقاط تقاضا می‌گویند، قرار دارند و CFLP را روی شبکه‌ای در نظر گرفتند که رأس‌های آن نقاط تقاضا هستند.
بر اساس مسأله مطالعه شده توسط وندل و مک‌کلوی [۲۹]، حکیمی [۱۳] مسأله CFLP را تحت این شرایط مطالعه کرد که تصمیم گیرنده تسهیلات خود را روی شبکه‌ای مکانی‌یابی می‌کند که تسهیلات دیگر قبلا روی آن قرار گرفته‌اند. درزنر [۷] مسأله حکیمی را به‌مسأله مکان‌یابی تسهیلات رقابتی بر روی صفحه‌ای شامل چندین تصمیم‌گیرنده و تسهیلات رقابتی توسعه داد.
به طور کلی، در این مسایل، مکان بهینه تمام تسهیلات به عنوان یک موقعیت تعادل شناخته می‌شود. از این لحاظ مسایل CFLP به دو دسته تقسیم می‌شوند:
دسته اول، مسایلی هستند که بر اساس اصل تعادل نش[٦] پایه‌ریزی شده و توسط هوتلینگ [۱۷]، ایتون و لیپسی [۸]، وندل و مک‌کلوی [۲۹] و ...مطالعه شده اند.
دسته دوم شامل مسایلی بر اساس تعادل استکلبرگ[۷] می‌باشند.این نوع تعادل در ادامه همین فصل معرفی خواهد شد. این مسایل توسط حکیمی [۱۳]، درزنر [۷]، کارکازیس [۱۸]، مورنو و همکاران [۲۱] و ...مطالعه شده‌اند.
مسایل دسته دوم در قالب مسایل برنامه‌ریزی دو مرحله‌ای[۸] (BPP)، که در آن دو تصمیم گیرنده با اولویت تصمیم‌گیری متفاوت وجود دارند، مدل‌سازی می‌شوند.

۳.۱ مسأله برنامه‌ریزی دو مرحله‌ای

یک مسأله برنامه‌ریزی دو مرحله‌ای، در واقع ترکیبی از دو مسأله برنامه‌ریزی ریاضی می‌باشد. فرض کنید مسأله برنامه‌ریزی ریاضی زیر موجود باشد:

[٦] Nash Equilibrium
[۷] Stackelberg Equilibrium
[۸] Bilevel Programming Problems

$$\begin{aligned} &\min_{x,y} F(x,y) \\ &\text{s.t.} \\ &\quad g(x,y) \leq ۰ \end{aligned} \qquad (۱-۱)$$

حال، فرض کنید که علاقه‌مندیم متغیر y، یک جواب بهینه برای مسأله زیر نیز باشد:

$$\begin{aligned} &\min_{y} f(x,y) \\ &\text{s.t.} \\ &\quad h(x,y) \leq ۰ \end{aligned} \qquad (۱-۲)$$

در چنین موقعیتی، می‌توانیم محدودیتی را به مسأله (۱) اضافه کنیم. این کار به‌مسأله برنامه‌ریزی دو مرحله‌ای زیر منجر می‌شود:

$$\begin{aligned} &\min_{x,y} F(x,y) \\ &\text{s.t.} \\ &\quad g(x,y) \leq ۰ \\ &\quad y \in \operatorname{argmin}\{f(x,y) : h(x,y) \leq ۰\} \end{aligned} \qquad (۱-۳)$$

مسأله اخیر نیز یک مسأله برنامه‌ریزی ریاضی همراه با محدودیت غیرخطی با یک ساختار ویژه می‌باشد. مطالعات انجام شده بر روی این مسأله نشان داده‌اند که اگر حتی تمام توابع خطی باشند این مسأله دارای پیچیدگی زمانی نمایی می‌باشد [۴].
مسایل برنامه‌ریزی دو مرحله‌ای عموما دارای سه ویژگی اصلی زیر هستند:

- ترکیبی از دو مسأله برنامه‌ریزی ریاضی می‌باشد به صورتی که یکی از آن‌ها در مجموعه محدودیت‌های دیگری قرار دارد.
- دارای رابطه سلسله مراتبی هستند، به این معنا که باید قبل از این که یک مسأله ارزیابی شود، مسأله دیگر مورد ارزیابی قرار گیرد.
- یک تصمیم گیرنده وجود دارد که بر تمام متغیرها کنترل دارد.

۴.۱ بازی استکلبرگ

یک بازی استکلبرگ[۹] شامل دو بازیکن می‌باشد به طوری که یکی از بازیکنان پیشرو[۱۰] و دیگری پیرو[۱۱] نامیده می‌شوند. پیشرو ابتدا حرکت خود را انجام می‌دهد و سپس پیرو با در نظر گرفتن حرکت پیشرو، حرکت می‌کند. اگر این بازی فقط یک مرتبه انجام شود، به آن «بازی پایا»[۱۲] می‌گویند. اگر یک بازی پایا چندین مرتبه انجام شود، به آن یک «بازی پویا»[۱۳] می‌گویند. در این نوع بازی اغلب فرض بر این است که اولا پیرو از حرکت پیشرو آگاهی کامل دارد و ثانیا هر دو بازیکن بهترین حرکت ممکن را انتخاب می‌کنند.
اگر استراتژی‌های بهینه دو بازیکن در یک بازی استکلبرگ پایا، جواب‌های مسایل برنامه‌ریزی ریاضی باشند، می‌توانیم این بازی را به صورت زیر مدل‌سازی کنیم:

$$\begin{aligned} &\min_{x,y} F(x,y) \\ &\text{s.t.} \\ &\quad g(x,y) \leq 0 \\ &\quad y \in \operatorname{argmin}\{f(x,y) : h(x,y) \leq 0\} \end{aligned} \qquad (۴-۱)$$

که در آن مسأله

$$\begin{aligned} &\min_{x,y} F(x,y) \\ &\text{s.t.} \\ &\quad g(x,y) \leq 0 \end{aligned}$$

مربوط به تصمیم‌گیری پیشرو ومسأله

$$\begin{aligned} &\min_{y} f(x,y) \\ &\text{s.t.} \\ &\quad h(x,y) \leq 0 \end{aligned}$$

---

[۹] Stackelberg Game
[۱۰] Leader
[۱۱] Follower
[۱۲] Static Game
[۱۳] Dynamic Game

مربوط به تصمیم‌گیری پیرو می‌باشد.
به این مسئله، مسأله استکلبرگ پایا[۱۴] (SSP) می‌گویند. همان طور که از مقایسه ساختارهای (۱—۳) و (۱—۴) مشاهده می‌شود SSP با BPP ارتباط دارد. اما در این مسئله، پیشرو (تصمیم گیرنده) تنها بر متغیر x کنترل دارد.
فرض کنید رابطه زیر برقرار باشد:

$$F(x,y) = -f(x,y)$$

در این صورت، بازی دارای مجموع صفر خواهد شد. با به کارگیری این فرض بر روی مدل SSP به مدل بازی استکلبرگ پایا با مجموع صفر[۱۵] (ZSSP) به صورت زیر می‌رسیم:

$$\begin{aligned} &\min_{x} f(x,y) \\ &\text{s.t.} \\ &\quad g(x,y) \leq 0 \\ &\quad y \in \operatorname{argmax}\{f(x,y) : h(x,y) \leq 0\} \end{aligned} \qquad (۱—۵)$$

توجه می‌کنیم که اگر تمام توابع در (؟؟ خطی باشند، این مسأله را مسأله ماکس—مین خطی[۱۶] (LMM) می‌نامند.

۵.۱ روش جریمه

در این بخش، به معرفی مختصر «روش جریمه[۱۷]» می‌پردازیم [۱۰].
مسأله بهینه‌سازی (P) را به صورت زیر در نظر بگیرید:

[۱۴] Static Stackelberg Problem
[۱۵] Zero-sum Static Stackelberg Problem
[۱۶] Linear Max-Min Problem
[۱۷] Penalty Method

$$\begin{aligned} &\min_x f(x) \\ &\text{s.t.} \\ &\quad g_i(x) = b_i,\ i = ۱, \ldots, m \\ &\quad x \in \mathbb{R}^n \end{aligned} \qquad (۶-۱)$$

که در آن ناحیه شدنی به صورت زیر تعریف می‌شود:

$$F = \{x \in \mathbb{R}^n | g_i(x) = b_i,\ i = ۱, \ldots, m\}.$$

روش جریمه برای حل مسأله (P)، به وسیله حل یک یا چند مسأله بدون محدودیت، طراحی شده است. دراین روش ، ناحیه شدنی مسأله (P) از مجموعه F به تمام $\mathbb{R}^n$ گسترش می‌یابد، ولی یک هزینه سنگین یا یک «جریمه» به تابع هدف برای نقاطی که خارج از F قرار دارند، اضافه می‌شود.

تعریف ۱.۵.۱ تابع $P(x) : \mathbb{R}^n \longrightarrow \mathbb{R}$ یک «تابع جریمه» برای مسئله‌بهینه‌سازی (P) نامیده می‌شود هرگاه $P(x)$ در شرایط زیر صدق کند:

- $P(x) = ۰$ اگر $g(x) = b$
- $P(x) > ۰$ اگر $g(x) \neq b$

که در آن $g(x) = \begin{pmatrix} g_۱(x) \\ \vdots \\ g_m(x) \end{pmatrix}$ و $b = \begin{pmatrix} b_۱ \\ \vdots \\ b_m \end{pmatrix}$.

مثال ۲.۵.۱ تابع

$$P(x) = \sum_{i=۱}^{m} (g_i(x) - b_i)^۲$$

یک تابع جریمه برای مسأله بهینه‌سازی (P) است که در شرایط تعریف ۱.۵.۱ صدق می‌کند.

اکنون مسئله‌ای به فرم زیر در اختیار داریم که به حل آن می‌پردازیم:

$$P_c: \quad \min_x f(x) + cP(x) \quad \text{s.t.} \quad x \in \mathbb{R}^n \qquad (۷-۱)$$

که در آن c یک دنباله صعودی از اعداد است به طوری که $c \longrightarrow \infty$. توجه کنید که در مسأله P_c یک جریمه به محدودیت‌هایی که برآورده نشده‌اند، اختصاص می‌دهیم. کمیت c یک «پارامتر جریمه[۱۸]» نامیده می‌شود.
لم و قضیه زیر، که در[۱۰] مطرح شده‌اند و به اثبات رسیده‌اند، تعدادی از خواص پایه‌ای تابع جریمه و همگرایی آن را نشان می‌دهند:

لم ۳.۵.۱ (لم جریمه)

فرض کنید $c_k \geq ۰$، $k = ۱, ۲, \ldots$، یک دنباله اکیدا صعودی از پارامترهای جریمه باشند. فرض کنید $q(c, x) = f(x) + cP(x)$ وبرای $k = ۱, ۲, \ldots$، x_k جواب‌های دقیق مسأله P_{c_k} و x^* جواب بهینه مسأله (P) باشد.

۱) $q(c_k, x_k) \leq q(c_{k+۱}, x_{k+۱})$

۲) $P(x_k) \geq P(x_{k+۱})$

۳) $f(x_k) \leq f(x_{k+۱})$

۴) $f(x^*) \geq q(c_k, x_k) \geq f(x_k)$

قضیه ۴.۵.۱ (قضیه همگرایی روش جریمه)

فرض کنید $f(x)$، $g(x)$ و $P(x)$ توابع پیوسته باشند. اگر $\{x_k\}_{k=۱}^{\infty}$ یک دنباله از جواب‌های مسأله P_{c_k} باشند، آن‌گاه هر نقطه حدی $\overline{x}$ از دنباله $\{x_k\}_{k=۱}^{\infty}$ یک جواب مسأله (P) است.

[۱۸] Penalty Parameter

۶.۱ چند تعریف اولیه

در این بخش چند تعریف مورد نیاز برای مطالعه و درک بهتر این پایان‌نامه را ارائه می‌کنیم.

تعریف ۱.۶.۱ (گراف)

گراف G، یک سه‌تایی مرتب $(V(G), E(G), \psi_G)$ است که تشکیل شده از یک مجموعه ناتهی $V(G)$ از راس‌ها، یک مجموعه $E(G)$ از یال‌ها و یک **تابع وقوع** ψ_G که به هر یال G یک زوج نامرتب از راس‌های G را نسبت می‌دهد. اگر e یک یال و u و v دو راس باشند به طوری که $\psi_G(e) = uv$، در این صورت گفته می‌شود که e، راس‌های u و v را به یکدیگر وصل کرده است و راس‌های u و v دو سر یال e نامیده می‌شوند.

تعریف ۲.۶.۱ (گراف وزن‌دار)

فرض کنید به هر یال e از G، یک عدد حقیقی $w(e)$، که **وزن** آن نامیده می‌شود، نسبت داده باشیم. در این صورت G به همراه وزن‌های روی یال‌هایش، یک **گراف وزن‌دار** نامیده می‌شود.

تعریف ۳.۶.۱ (مسیر)

یک **مسیر** از G، دنباله ناصفر متناهی $P = v_0e_1v_1e_2v_2 \ldots e_kv_k$ است به طوری که جملات آن یک درمیان از راس‌ها و یال‌های متمایز تشکیل شده و به ازای $1 \leq i \leq k$، v_{i-1} و v_i دو سر یال e_i می‌باشند.

برای سایر تعارف مربوط به نظریه گراف به[۲] مراجعه کنید.

تعریف ۴.۶.۱ (مسأله بهینه‌سازی)

یک مسأله بهینه‌سازی به طور کلی به صورت زیر نشان داده می‌شود:

$$\min\{f(x)|x \in X, X \subseteq S\}$$

که در آن S، X، x و f به ترتیب نشان دهنده فضای جواب، مجموعه شدنی، یک جواب شدنی و تابع هدف با مقادیر حقیقی می‌باشند. اگر S یک مجموعه متناهی اما با اندازه بزرگ باشد، به یک مسأله بهینه‌سازی ترکیبیاتی می‌رسیم. اگر $S = \mathbb{R}^n$، به بهینه‌سازی پیوسته خواهیم رسید.

تعریف ۵.۶.۱ (مسأله برنامه‌ریزی ۱—۰)

یک مسأله بهینه‌سازی را که در آن فضای جواب متشکل از تمام بردارهایی است که مولفه‌هایشان تنها مقادیر ۰ یا ۱ را اختیار می‌کنند، **مسأله برنامه‌ریزی ۱—۰** می‌نامند.

فصل ۲

مسأله مکان‌یابی تدافعی (DLP)

۱.۲ معرفی DLP

در این بخش، به بررسی یک مسأله مکان‌یابی جدید که توسط اونو و کاتاگیری [۲۸] معرفی شده است می‌پردازیم. در این مسأله یک تصمیم گیرنده مکان تسهیلات دفاعی را برای جلوگیری از دستیابی دشمنان به یک سایت استراتژیک مهم به نام هسته، تعیین می‌کند. چنین مسائل مکان‌یابی «مسائل مکان‌یابی تدافعی» (DLP) نامیده می‌شوند، که در آن به تصمیم گیرنده مدافع[۱] و به دشمنان مهاجم[۲] گفته می‌شود [۲۸]. مثال‌هایی از مسایل DLP عبارتند از: دولت یک کشور پایگاه‌های دفاعی را به گونه‌ای مکان‌یابی می‌کند که از دستیابی مهاجمین به پایتخت آن کشور جلوگیری کند یا یک مربی فوتبال، بازیکنان تیمش را برای دفاع از دروازه در مقابل حریف در مکان‌های مناسب قرار می‌دهد و غیره. در این گونه مسایل، فرض بر این است که ناحیه‌ای که تصمیم گیرنده امکانات تدافعی خود را در آن قرار می‌دهد به صورت یک شبکه و هسته یک راس از آن شبکه می‌باشد. همچنین مدافع در اولویت تصمیم‌گیری نسبت به مهاجم قرار دارد. در ادامه نشان داده می‌شود که می‌توان مسایل DLP را به صورت

[۱] Defender
[۲] Invader

مسایل برنامه‌ریزی ۰ و ۱ مدل‌سازی کرد.
از آن جایی که مسایل DLP مدل‌سازی شده، دارای پیچیدگی زمانی نمایی هستند، باید یک الگوریتم کارآمد برای حل آن‌ها طراحی کرد. مسایل DLP دارای دو خاصیت زیر هستند:

۱) متغیرهای تصمیم‌گیری در مسایل DLP، ۰ و ۱ هستند.

۲) مدافع به اندازه‌ای امکانات تدافعی ایجاد می‌کند تا شرایط مسأله را برقرار نماید.

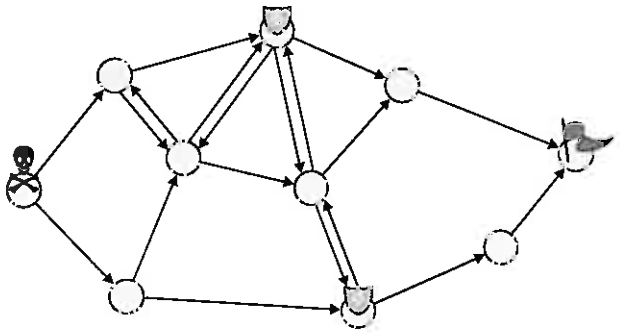
از آن جایی که مسایل کوله‌پشتی ۰ و ۱[۳] نیز دارای همین دو خاصیت هستند، می‌توان روش‌هایی ابتکاری را که برای حل مسأله کوله‌پشتی استفاده می‌شوند برای حل مسأله DLP به کار گرفت. الگوریتم‌های ابتکاری[۴] که تا کنون برای مسأله کوله‌پشتی ارائه شده‌اند عبارتند از: الگوریتم ژنتیک[۵] با ساختار رشته‌ای دوگانه توسط ساکاوا و همکاران [۲۵] و الگوریتم جستجوی ممنوعه توسط هنفی و فرویل [۱۴].

۱.۱.۲ مدل‌سازی مسأله مکان‌یابی تدافعی

در این بخش، مدل مسأله DLP روی شبکه را بیان می‌کنیم. ابتدا، این مسأله را با استفاده از یک مثال معرفی می‌کنیم. حکومت یک کشور پایگاه‌های دفاعی را برای جلوگیری از دستیابی مهاجمان به پایتخت آن کشور مکان‌یابی می‌کند. در این مثال، کشور به صورت یک شبکه، پایتخت آن، که همان هسته می‌باشد به صورت راسی از آن شبکه نمایش داده می‌شود و مهاجم نیز در یکی از راس‌های این شبکه قرار دارد. به این صورت، DLP می‌تواند به صورت یک مسأله با هدف تعیین این که کدام یک از راس‌ها برای تاسیس امکانات دفاعی مناسب می‌باشند، مدل‌سازی شود. شکل ۱.۲، مثال ارائه شده را تشریح می‌کند. برای فرموله‌سازی DLP، از نمادهایی برای این شبکه استفاده می‌شود.

---

[۳] 0-1 Knapsack Problem
[۴] Heuristic Algorithms
[۵] Genetic Algorithm



شکل ۱.۲: مساله‌ی مکان‌یابی تدافعی روی شبکه

فرض کنیم، شبکه دارای n راس، v_1، v_2، ...و v_n و r یال باشد. مجموعه‌های V و E را به ترتیب به عنوان مجموعه رئوس و یال‌های شبکه مذکور در نظر می‌گیریم. بنابراین، $G = (V, E)$ شبکه ایست که مدافع، تسهیلات دفاعی خود را روی آن مکان‌یابی می‌کند. راسی را که هسته در آن قرار دارد و مدافع می‌بایست از آن دفاع کند، با c نشان می‌دهیم. بدون از دست رفتن کلیت، فرض می‌کنیم که مهاجم در nامین راس شبکه یعنی v_n که آن را با ζ نشان می‌دهیم، قرار دارد. فرض کنیم e_{ij} یال مربوطه از راس v_i به v_j و همچنین w_{ij} معرف وزن یال e_{ij} باشد. آن‌گاه، فاصله از راس دلخواه v تا هسته، که مجموع وزن‌های یال‌های مشمول در مسیر می‌باشد، عبارت است از:

$$d^c(v) \equiv \min_{p \in P_{vc}} \sum_{e_{ij}} w_{ij}. \qquad (۱-۲)$$

که در آن P_{vc} مجموعه مسیرها از راس v به راس c است.

حال نمادهایی را برای مدافع و مهاجم معرفی می‌کنیم. فرض بر این است که مدافع تنها قادر به قرار دادن حداکثر یک وسیله دفاعی به روی هر یک از رئوس به جز ζ، که مهاجم در آن قرار دارد، می‌باشد. برای $i \in \{1, \ldots, n-1\}$، فرض کنید $q_i \in \{0, 1\}$ یک متغیر دودویی است به گونه‌ای که $q_i = 1$ اگر یک وسیله دفاعی در راس v_i قرار گیرد و در غیر این صورت $q_i = 0$. همچنین فرض کنید $q \equiv (q_1, \ldots, q_{n-1})$ بردار

تصمیم مدافع باشد. از سوی دیگر، مهاجم می‌خواهد راهی از ζ به c برای خود پیدا کند.

اکنون به مدل‌سازی تابع هدف برای مدافع و مهاجم می‌پردازیم. موارد زیر را فرض می‌کنیم:

- مهاجم دارای مقدار معینی انرژی است، و انرژی اولیه مهاجم قبل از ترک ζ با $\overline{\alpha} > 0$ نشان داده می‌شود.
- مهاجم انرژی خود را در دو صورت مصرف می‌کند:

۱) با طی کردن یال e_{ij}، انرژی مهاجم به اندازه w_{ij} کم می‌شود.

۲) در صورت برخورد با یکی از وسایل دفاعی، انرژی مهاجم به اندازه β کاهش می‌یابد که در این جا $\beta > 0$ ظرفیت وسیله دفاعی می‌باشد. در این صورت، اگر مهاجم از راس v_i به v_j برود، انرژی کاهش یافته آن به صورت زیر نمایش داده می‌شود

$$\overline{w}(e_{ij}, q) = w_{ij} + \beta q_j \qquad (۲-۲)$$

- اگر انرژی مهاجم به صفر یا کمتر از آن برسد، مهاجم خواهد مرد.

برای تصمیم مهاجم که با بردار p نمایش داده می‌شود، فرض کنیم $v^\lambda(p) \in V$ نشان دهنده λامین راس در p باشد. در این صورت، مهاجم می‌تواند در λامین راس p یعنی $v^\lambda(p)$ حضور داشته باشد اگر انرژی مهاجم در این مرحله، که با $\alpha(v^\lambda(p)|q)$ نشان داده می‌شود و از رابطه زیر حاصل می‌شود، بیشتر یا مساوی صفر باشد:

$$\alpha(v^\lambda(p)|q) \equiv \overline{\alpha} - \sum_{l=1}^{\lambda} \overline{w}(e^l(p), q) \qquad (۳-۲)$$

که در آن $e^l(p) \in E$، lامین یال در p است. هدف مهاجم رسیدن به هسته c و یا اگر این کار غیرممکن باشد، رسیدن به نزدیک‌ترین راس ممکن به c است. به این ترتیب،

تابع هدف مهاجم به صورت زیر مدل‌سازی می‌شود:

$$f^I(q,p) \equiv \min_{v\in p}\{d^c(v)|\alpha(v|q) \geq ۰\} \qquad (۴-۲)$$

در این جا DLP به عنوان یک بازی توازن دونفره استکلبرگ با مجموع صفر[۶] در نظر گرفته شده است. به این ترتیب تابع هدف مدافع که با $f(q,p)$ نمایش داده می‌شود به صورت زیر خواهد بود:

$$f(q,p) + f^I(q,p) = ۰ \qquad (۵-۲)$$

در گام بعد، مجموعه‌های شدنی برای مدافع و مهاجم نشان داده می‌شوند. در اکثر موارد در مسایل مکان‌یابی تسهیلات، محدودیت‌های گوناگونی از قبیل هزینه ساختن تسهیلات، اختصاص کارکنان و غیره مد نظر قرار می‌گیرند. در [۲۸] DLP با محدودیت‌های خطی در نظر گرفته شده است. فرض کنید m، تعداد محدودیت‌ها و A ماتریس سمت چپ و b بردار سمت راست به صورت زیر باشند:

$$A = \begin{pmatrix} a_{۱۱} & \cdots & a_{۱n-۱} \\ \vdots & \ddots & \vdots \\ a_{m۱} & \cdots & a_{mn-۱} \end{pmatrix}, \quad b = \begin{pmatrix} b_۱ \\ \vdots \\ b_m \end{pmatrix} \qquad (۶-۲)$$

که در آن برای هر $i \in \{۱,\ldots,n-۱\}$ و هر $j \in \{۱,\ldots,m\}$، $a_{ij} \geq ۰$ و $b_j \geq ۰$. بنابراین مجموعه شدنی برای مدافع به صورت زیر می‌باشد:

$$SD = \{q \in \{۰,۱\}^{n-۱} | Aq \leq b\}$$

از سوی دیگر، مجموعه شدنی برای مهاجم که با SI نمایش می‌دهیم، عبارتست از تمام مسیرها از ς به c. با توجه به تمام توضیحات ارائه شده، DLP به صورت زیر مدل‌سازی می‌شود:

[۶] Zero-sum Stackelberg Equilibrium Game

$$\begin{aligned} &\max_{q} \quad f(q,p) \\ &\qquad \text{که در آن } p \text{له زیر است: جواب مسا} \\ &\min_{p} \quad f(q,p) \\ &\text{s.t.} \\ &\qquad q \in SD,\ p \in SI \end{aligned} \tag{۲—۷}$$

برای مطالعه پیچیدگی زمانی برای پیدا کردن جواب بهینه مسأله DLP ملاحظه می‌کنیم که با توجه به معادلات (۲—۳) و (۲—۴) برای محاسبه مقدار تابع هدف برای هر بردار تصمیم q، که $q \in SD$، می‌باید دو مقدار زیر محاسبه شوند:

۱) مینیمم مجموع وزن‌ها از ζ به تمام رئوس با در نظر گرفتن بردار مکان q.

۲) فواصل تمام رئوس تا راس c، که در واقع مینیمم مجموع وزن‌ها از تمام رئوس به راس c است، وقتی هیچ تسهیلاتی روی شبکه جایگزین نشده است.

مسأله پیدا کردن این مقادیر می‌تواند در قالب مسأله کوتاه‌ترین مسیر[۷] مدل‌سازی شود. روش دایجسترا[۸] یکی از کارآمدترین الگوریتم‌ها برای حل مسایل کوتاه‌ترین مسیر می‌باشد و پیچیدگی زمانی آن از مرتبه $O(r + n\log n)$ می‌باشد[۲۸]. از آن جایی که پیچدگی محاسباتی برای پیدا کردن تعداد مکان‌های شدنی برای مدافع از مرتبه $O(۲^n)$ می‌باشد، پیچیدگی محاسباتی برای پیدا کردن جواب بهینه مسایل DLP از مرتبه $O(۲^n(r + n\log n))$ می‌باشد. این امر بدین معناست که پیدا کردن جواب بهینه مسایل DLP در مقیاس بزرگ کاری بسیار دشوار است[۲۸] .

[۷] Shortest Path Problem
[۸] Dijkstra Method

۲.۲ الگوریتم دایجسترا

همان طور که در این فصل مشاهده شد، در جریان مسأله مکان‌یابی تدافعی، برای محاسبه تابع هدف به ازای هر بردار تصمیم مدافع (q) لازم است طول کوتاه‌ترین مسیر از تمام رئوس تا رأس c و همین طور طول کوتاه‌ترین مسیر از رأس ζ به تمام رئوس محاسبه شود. از این رو، در قسمت پایانی این فصل، الگوریتم دایجسترا را برای یافتن کوتاه‌ترین مسیر بین دو رأس از یک گراف وزن‌دار معرفی می‌کنیم[۲].

طول یک مسیر p داده شده بین رأس s و رأس t در یک گراف وزن‌دار G، عبارت است از مجموع وزن‌های یال‌های مشمول در این مسیر. بدین صورت می‌خواهیم کوتاه‌ترین مسیر، یا به عبارت دیگر مسیری با کمترین طول را بین دو رأس s و t پیدا کنیم. در الگوریتم دایجسترا، وزن یال‌ها مثبت در نظر گرفته می‌شوند.

در این الگوریتم، از یک تکنیک برچسب گذاری استفاده می‌شود. در ابتدا، $\lambda(s) = 0$ در نظر می‌گیریم و به ازای $v \neq s$، $\lambda(v)$ هنوز نامعین می‌باشد. در گام بعد، تمام رئوس v مجاور با s را با $\lambda(v)$ برچسب گذاری می‌کنیم که در آن $\lambda(v)$ وزن یال بین s و v است. فرض کنیم u رأسی در بین تمام vها با کمترین مقدار λ می‌باشد. حال به سراغ رئوسی مانند w مجاور با u می‌رویم و برای آن wهایی که تا کنون برچسب گذاری نشده‌اند، برچسب $\lambda(w) = \lambda(u) + w(e)$ را در نظر می‌گیریم، که در آن $w(e)$ وزن یال متصل کننده u به v است. همچنین برای رئوسی که از قبل برچسب گذاری شده‌اند، برچسب آن‌ها را به $\lambda(u) + w(e)$ تغییر می‌دهیم تنها در صورتی که این مقدار اخیر کوچک‌تر باشد.

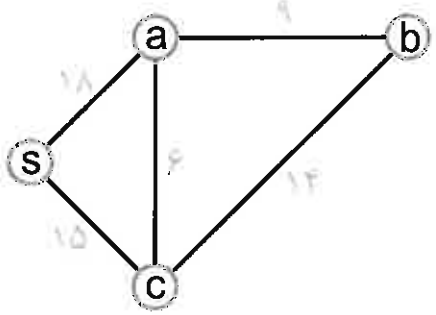
حال در بین تمام wها، آن رأسی را که دارای کمترین $\lambda(w)$ است انتخاب می‌کنیم و به همین ترتیب مراحل تکرار می‌شوند.

حال به معرفی الگوریتم در قالب گام‌های پنج‌گانه می‌پردازیم و دو قرارداد را مد نظر قرار می‌دهیم، برای هر $x \in \mathbb{R}$، $\infty + x = \infty$ و همچنین $\infty + \infty = \infty$.

الگوریتم دایجسترا

گام ۱: $\lambda(s) = 0$ قرار بده و برای تمام رئوس v که $v \neq s$، $\lambda(v) = \inf$. مجموعه T را برابر با مجموعه رئوس گراف قرار بده یعنی $T = v$ (ما مجموعه T را به عنوان مجموعه رئوس رنگ نشده در نظر می‌گیریم.).
گام ۲: u را رأسی از T انتخاب کن به طوری که $\lambda(u)$ مینیمم باشد.
گام ۳: اگر $u = t$، الگوریتم را متوقف کن.
گام ۴: برای هر رأس $e = uv$ که مجاور u می‌باشد، اگر $v \in T$ و $\lambda(v) > \lambda(u) + w(e)$ مقدار $\lambda(v)$ را به $\lambda(u) + w(e)$ تغییر بده (یعنی، با داشتن هر یال e از یک رأس رنگ نشده v به u مقدار $\lambda(v)$ را به مینمم $(\lambda(u), w(e), \lambda(v))$ تغییر بده).
گام ۵: مجموعه T را به $T - \{u\}$ تبدیل کن و به گام ۲ برو (یعنی، u را رنگ بزن و سپس به گام ۲ برو و یک رأس رنگ نشده با حداقل مقدار برچسب انتخاب کن).

مثال ۱.۲.۲ گراف شکل ۲.۲ را در نظر بگیرید. می‌خواهیم کوتاه‌ترین مسیر را از رأس s به سایر رأس‌ها بیابیم.



شکل ۲.۲: مثالی برای الگوریتم دایجسترا

در جدول زیر گام‌های طی شده الگوریتم برای مثال فوق آمده است. اطلاعات

درون جدول شامل: انتخاب رأس u (در گام دوم)، تغییرات $\lambda(v)$ (در گام چهارم) و تغییرات مجموعه T، یعنی مجموعه رئوس رنگ نشده (در گام پنجم) می‌باشد.

	u	$\lambda(v)$ s	a	b	c	T
مرحله اول	s	۰	∞	∞	∞	$\{s,a,b,c\}$
مرحله دوم	c		۱۸	∞	۱۵	$\{a,b,c\}$
مرحله سوم	a		۱۸	۲۹		$\{a,b\}$
مرحله چهارم	b			۲۷		$\{b\}$

با توجه به جدول فوق، طول کوتاه‌ترین مسیر از رأس s به رأس‌های a، b و c به ترتیب ۱۸، ۲۷ و ۱۵ می‌باشد.

فصل ۳

حل DLP به روش جستجوی ممنوعه

۱.۳ روش جستجوی ممنوعه (TS)

مشهورترین روش جستجوی همسایگی که برای پیدار کردن تقریبی از حداقل مقدار یک تابع حقیقی مقدار بر روی مجموعه‌ای مانند S به کار می‌رود [۱۶]، عبارتست از روش کاهشی[۱]، که به صورت زیر خلاصه می‌شود:

روش کاهشی:

فرض کنید $N(i)$ نشان دهنده‌ی همسایگی i باشد.

گام ۱. یک جواب اولیه i در S انتخاب کن.

گام ۲. بهترین جواب j را در $N(i)$ پیدا کن (یعنی، جواب j به طوری که: $(\forall k \in N(i) : f(j) \leq f(k)$.

گام ۳. اگر $f(j) \geq f(i)$ توقف کن. در غیر این صورت قرار بده $i = j$ و به گام ۲ برو.

[۱] Descent Method

برای افزایش کارایی فرآیند جستجو، لازم است که نه تنها اطلاعات موضعی (همانند مقدار جاری تابع هدف) بلکه اطلاعاتی در مورد فرآیند جستجو را به خاطر سپرد. این استفاده نظام‌مند از حافظه در واقع ویژگی اصلی روش TS می‌باشد. در حالی که بیشتر روش‌های جستجو تنها مقدار $f(i^*)$ را که در واقع مقدار تابع هدف برای بهترین جواب به دست آمده i^* تا کنون می‌باشد، در حافظه نگه می‌دارند، TS اطلاعاتی راجع به جواب‌های به دست آمده در طول روند جستجو را نیز به خاطر می‌سپارد. نقش حافظه، محدود کردن انتخاب بین تعدادی از زیرمجموعه‌های $N(i)$ به وسیله ممنوع کردن بعضی از جواب‌ها می‌باشد.

فرض کنید مسأله بهینه‌سازی بدین صورت داده شده است که مجموعه جواب‌های شدنی S و تابع $f : S \longrightarrow \mathbb{R}$ مفروض است. مطلوبست $i^* \in S$ به طوری که $f(i^*)$ بر طبق یک یا چند معیار قابل قبول باشد. معمولا معیار مقبولیت i^* در واقع برقراری نامساوی $f(i^*) \leq f(i)$ به ازای همه iهای متعلق به S است. در چنین موردی، TS می‌تواند یک روش دقیق باشد، به این معنی که بعد از تعدادی متناهی تکرار می‌تواند i^* را پیدا کند.

با این وجود، در بسیاری از موارد هیچ تضمینی برای دستیابی به i^* وجود ندارد، از این رو TS به عنوان یک روند ابتکاری بسیار عمومی در نظر گرفته می‌شود.

به عنوان اولین گام در توصیف TS، روش کاهشی را به گونه‌ای دیگر معرفی می‌کنیم

گام ۱. یک جواب اولیه $i \in S$ انتخاب کن.

گام ۲. زیرمجموعه V^* در مجموعه جواب‌های درون $N(i)$ را انتخاب کن.

گام ۳. بهترین j را در V^* پیدا کن (یعنی $(\forall k \in V^* : f(j) \leq f(k)$).

گام ۴. اگر $f(j) \geq f(i)$ متوقف شو. در غیر این صورت قرار بده $i = j$ و به گام ۲ برو.

در روش کاهشی مستقیم در واقع $V^* = N(i)$ در نظر گرفته می‌شود. اما در روش TS، برای انتخاب V^* در هر گام برای به کار بردن اطلاعات موجود از تابع هدف و

مقدار همسایگی $N(i)$ به طور نظام‌مند از حافظه استفاده می‌شود.
به جز موارد خاص که در آن محدب بودن فضای شدنی وجود دارد، به کار بردن روش‌های کاهشی بی‌فایده است، چون احتمال به دام افتادن در یک مینیمم موضعی، که ممکن است از نظر مقدار تابع هدف از مینیمم سراسری دور باشد، وجود دارد. بنابراین اگر بخواهیم از یک مینیمم موضعی فرار کنیم، باید به نوعی حرکات غیر بهبود بخشنده از جواب i به جواب j در V^* (یعنی $f(j) > f(i)$) را نیز قبول کنیم.
به محض این که حرکات غیر بهبود بخشنده ممکن باشند، ریسک ملاقات با جواب‌های تکراری و همین طور افتادن در دور، زیاد می‌شود. این در واقع نقش حافظه برای اجتناب از حرکت‌هایی است که به جواب‌های تکراری منجر می‌شوند. اگر چنین حافظه‌ای وجود داشته باشد، ساختار همسایگی $N(i)$ به وضعیت فرآیند جستجوی طی شده و در نتیجه به تکرار k بستگی دارد. بنابراین به جای $N(i)$ می‌توانیم از $N(i,k)$ استفاده کنیم. با این توضیحات روش کاهشی را به گونه‌ای تغییر می‌دهیم که به قالب عمومی TS نزدیک شود: (i^* بهترین جوابی است که تا کنون به دست آمده و k شمارنده تکرار می‌باشد.)

گام ۱. جواب اولیه $i \in S$ را انتخاب کن. قرار بده $i^* = i$ و $k = 0$.

گام ۲. قرار بده $k = k + 1$ و زیرمجموعه V^* از جواب‌های شدنی $N(i,k)$ را انتخاب کن.

گام ۳. بهترین j را در V^* پیدا کن و قرار بده $i = j$.

گام ۴. اگر $f(j) \geq f(i)$، آن‌گاه قرار بده $i^* = i$.

گام ۵. اگر یک شرط توقف برقرار شد، توقف کن. در غیر این صورت به گام ۲ برو.

تعریف $N(i,k)$ به گونه‌ایست که بعضی از جواب‌هایی که قبلا به دست آمده‌اند، از $N(i)$ حذف می‌شوند. این جواب‌ها به عنوان جواب‌های ممنوعه قلمداد می‌شوند که باید در تکرار بعدی از آن‌ها اجتناب شود. چنین حافظه‌ای مبنی بر تکرار قبلی

از ایجاد دور جلوگیری می‌کند. برای نمونه، در تکرار k، یک لیست T (لیست ممنوعه) شامل آخرین $|T|$ جواب به دست آمده، می‌تواند از دورهای به طول $|T|$ جلوگیری کند. در این موارد می‌توان $N(i,k) = N(i) - T$ را در نظر گرفت. با این وجود استفاده از چنین لیستی ممکن است غیر عملی باشد. بنابراین، فرآیند جستجو در S را بر پایه حرکت‌ها از یک جواب به جواب بعدی قرار می‌دهیم. برای هر جواب i در S، مجموعه $M(i)$ به عنوان تمام حرکت‌هایی که می‌توانند بر روی جواب i برای دست‌یابی به یک جواب جدید j، اعمال شوند به کار می‌بریم (نماد $j = i \oplus m$). در این صورت، $N(i) = \{j | \exists m \in M(i), j = i \oplus m\}$. به طور کلی حرکت‌هایی را در نظر می‌گیریم که معکوس‌پذیرند: برای هر $m \in M$ یک m^{-1} وجود دارد به طوری که $(i \oplus m) \oplus m^{-1} = i$. بنابراین، به جای به حافظه سپردن لیست T از آخرین $|T|$ جواب به دست آمده، می‌توانیم آخرین $|T|$ حرکت یا آخرین $|T|$ حرکت معکوس را به حافظه بسپاریم. به وضوح چنین محدودیتی در واقع باعث از دست رفتن اطلاعات می‌باشد و دیگر تضمینی برای اجتناب از دورهای به طول حداکثر $|T|$ وجود ندارد.

برای کارایی بیشتر، مناسب است که از چندین لیست T_r به طور همزمان استفاده کرد. در این صورت بعضی از مؤلفه‌های t_r (از i یا m) یک وضعیت ممنوعه را القا می‌کنند که باعث می‌شود شرکت این مؤلفه‌ها در یک حرکت مجاز نباشد. معمولا وضعیت ممنوعه یک حرکت در واقع تابعی از وضعیت ممنوعه مؤلفه‌های آن دارد، که ممکن است در هر تکرار عوض شوند. بنابراین می‌توان مجموعه‌ای از شرایط ممنوعه را به صورت زیر در نظر گرفت: $(r = 1, \ldots, t) \quad t_r(i,m) \in T_r$.

یک حرکت m (که بر روی جواب i اعمال می‌شود) یک حرکت ممنوعه است اگر همه شرایط برقرار شوند.

با این توضیحات، به معرفی قالب کلی روش جستجوی ممنوعه می‌پردازیم:

گام ۱. یک جواب اولیه i در S انتخاب کن. قرار بده $i^* = i$ و $k = 0$.

گام ۲. قرار بده $k = k + 1$ و زیرمجموعه V^* از $N(i,k)$ را به گونه‌ای انتخاب کن که حداقل یکی از شرایط ممنوعه $t_r(i,m) \in T_r$ برقرار نشود.

گام ۳. بهترین جواب $j = i \oplus m$ را در V^* انتخاب کن و قرار بده $i = j$.

گام ۴. اگر $f(i) \leq f(i^*)$، آن‌گاه $i^* = i$.

گام ۵. شرایط ممنوعه را به روز کن.

گام ۶. اگر یک شرط توقف برقرار شد، توقف کن. در غیر این صورت به گام ۲ برو.

۲.۳ به‌کارگیری روش TS برای حل DLP

اونو و کاتاگیری [۲۸] در مقاله خود در سال ۲۰۰۸، بعد از معرفی و مدل‌سازی مسأله DLP، یک روش حل بر مبنای روش جستجوی ممنوعه برای مسأله DLP ارائه کردند. در این بخش، به توضیح این روش می‌پردازیم.
فرض کنید $t^1 \equiv (1, 0, \ldots, 0)$، ... و $t^{n-1} \equiv (0, \ldots, 0, 1)$، $n-1$ بردار یکه باشند. همچنین، $q^1, q^2 \in SD$ (SD مجموعه شدنی مدافع) مکان‌هایی باشند که $q^1 + t^i = q^2$ که در آن v_i رأسی است که $v_i \in V \setminus \{\zeta\}$. بدین ترتیب، رابطه زیر برقرار می‌باشد:

$$f(q^1, p) \leq f(q^2, p) \qquad (۱-۳)$$

که در آن p بردار تصمیم مهاجم است. زیرا در q^2 یک وسیله دفاعی به شبکه اضافه شده است، بنابراین مهاجم درهمان رأس و یا در رأس دورتری نسبت به هسته متوقف می‌شود.
از معادله (۳-۱)، معلوم می‌شود که مدافع باید مادامی که شرایط DLP برقرار است هر چه بیشتر تسهیلات دفاعی را در شبکه قرار دهد. این امر بدین معناست که یک جواب بهینه برای مسایل DLP در نزدیکی یکی از مرزهای مجموعه جواب‌های شدنی SD وجود دارد. توجه کنید که مسایل کوله‌پشتی نیز دارای ویژگی معادله (۳-۱)

می‌باشند. هنفی و فرویل [۱۴]، یک روش جستجوی ممنوعه را برای مسایل چندبعدی کوله‌پشتی ارائه دادند، که می‌تواند یک جواب بهینه تقریبی مناسب را به وسیله جستجوی کارآمد در نواحی مرزی مجموعه شدنی، پیدا کند. در این جا، یک الگوریتم کارآمد را برای مسایل DLP بر مبنای الگوریتم هنفی، که توسط اونو و کاتاگیری [۲۸] مورد مطالعه قرار گرفت، ارائه می‌کنیم. جستجوی ممنوعه یک روش جستجوی موضعی است. فرض کنید، $S_M = \{\pm t^i | i = ۱, \ldots, n-۱\}$، که به هر کدام از اعضای مجموعه S_M یک حرکت گفته می‌شود و ناحیه همسایگی بردار q را با $N(q) \equiv q + S_M$ تعریف می‌شود. در روش جستجوی موضعی، یک نقطه جستجوگر از یک جواب جاری به یک جواب جدید که از لحاظ یک تابع تخمینی دارای بهترین مقدار است، حرکت می‌کند. در روش‌های جستجوی ممنوعه، حرکت‌هایی که در گام‌های قبلی جستجو انتخاب شده‌اند، در یک دوره معین انتخاب نمی‌شوند، دوره‌ای که به آن دوره ممنوعه می‌گویند و با $T \geq ۰$ نشان می‌دهند، حتی اگر چنین حرکتی بهترین حرکت از لحاظ تابع تخمینی باشد. چنین حرکت‌هایی به عنوان حرکت‌های ممنوعه، و حافظه‌ای که دوره ممنوعه را برای هر حرکت نگه می‌دارد، لیست ممنوعه نامیده می‌شود.

الگوریتم جستجوی ممنوعه که به وسیله هنفی و فرویل [۱۴] ارائه شد، از دو نوع گام که در زیر مطرح شده‌اند، ساخته شده است. یکی گام‌هایی است که در آن‌ها تعداد تسهیلات افزایش می‌یابد و دیگری گام‌هایی است که در آن‌ها تعداد تسهیلات کاهش می‌یابد. دو گام مذکور به شرح زیرند:

۱) **در مواردی که تعداد تسهیلات کاهش می‌یابد:** الگوریتم جستجوی ممنوعه که توسط هنفی و فرویل [۱۴] ارائه شده از یک تابع ارزش‌یابی استفاده می‌کرد که عبارتست از درجه بهبود مقادیر تابع هدف بخش بر درجه بهبود شدنی بودن در مورد شرایط. با این وجود، برای محاسبه درجه بهبود مقادیر تابع هدف، لازم است که یک مسأله کوتاه‌ترین مسیر با پیچیدگی $O(r + n \log n)$، حل شود. چون لازم است الگوریتم جستجوی ممنوعه، تابع ارزش‌یابی را بارها محاسبه کند، مطلوب است که تابع ارزش‌یابی ساده باشد. در الگوریتم پیشنهادی توسط

اونو و کاتاگیری [۲۸] از تابع ارزش‌یابی حرکت $-t^i$، که در واقع درجه بهبود شدنی بودن شرایط می‌باشد و با $\delta(-t^i)$ نشان داده می‌شود، استفاده می‌شود:

$$\delta(-t^i) \equiv t^i \cdot a_i. \qquad (۲-۳)$$

که در آن a_i ستون iام ماتریس A می‌باشد.

چون تابع ارزش‌یابی احتیاجی به محاسبه مقدار تابع هدف ندارد، مقدار تابع هدف $f(q,p)$ تنها وقتی q از مقدار جاری به مقدار بعدی انتقال می‌یابد محاسبه می‌شود.

توجه کنید که اگر از تابع ارزش‌یابی δ استفاده کنیم و دوره ممنوعه T کوتاه باشد، رئوس کاندید برای قرار دادن تسهیلات دفاعی به شدت محدود می‌شوند. بنابراین از یک دوره ممنوعه بلند در الگوریتم جستجوی ممنوعه برای حل DLP استفاده می‌شود.

۲) **در مواردی که تعداد تسهیلات افزایش می‌یابد:** الگوریتم جستجوی ممنوعه پیشنهادی توسط هنفی و فرویل [۱۴] از تابع ارزش‌یابی که در واقع درجه بهبود مقادیر تابع هدف است استفاده می‌کند. برای محاسبه درجه بهبود مقادیر تابع هدف، لازم است که مسأله کوتاه‌ترین مسیر را با پیچیدگی $O(r + n\log n)$ حل کنیم. از این رو تعداد دفعات محاسبه مقادیر تابع ارزش‌یابی باید کاهش یابد. قضیه زیر که در [۲۸] آمده است، برای کاهش تعداد دفعات محاسبه، مفید خواهد بود.

قضیه ۱.۲.۳ به ازای یک جواب جاری q، فرض کنید $v_i \in V \setminus \{\zeta\}$، رأسی باشد که هیچ وسیله دفاعی در آن قرار نگرفته، و p^* بهترین پاسخ مهاجم در مقابل q باشد و همچنین شامل رأس v_i نباشد. در این صورت، p^* یکی از

مسیرهای بهینه برای مهاجم در مقابل بردار تصمیم مدافع $q + t^i$، نیز می‌باشد، و معادله زیر برقرار است

$$f(q + t^i, p^*) = f(q, p^*) \qquad (۳-۳)$$

اثبات. با توجه به معادله (۳-۱)، رابطه زیر برقرار است

$$f(q + t^i, p^*) \geq f(q, p^*) \qquad (۳-۴)$$

چون p^* مسیری است که از v_i نمی‌گذرد، رابطه زیر برقرار است

$$f(q + t^i, p^*) = f(q, p^*) \qquad (۳-۵)$$

از معادله (۳-۴)، p^* یکی از مسیرهای بهینه در مقابل مکان‌یابی $q + t^i$ می‌باشد. بنابراین حکم برقرار است. □

روش جستجوی ممنوعه ارائه شده، نه تنها مکان‌های تسهیلات دفاعی، بلکه مقادیر تابع هدف و مسیرهای بهینه در مقابل آن‌ها را به حافظه می‌سپارد. در این صورت، با توجه به قضیه ۱.۲.۳، فقط لازم است که مقدار تابع هدف برای مکان‌یابی تمام رأس‌هایی محاسبه شود که مشمول در یک مسیر بهینه برای مهاجم باشند. در الگوریتم ارائه شده، پارامترهای $N_z, N_{TD} > 0$ به عنوان معیار توقف در نظر گرفته شده‌اند.

الگوریتم ۱.۳ (الگوریتم جستجوی ممنوعه برای حل کردن مسایل مکان‌یابی تدافعی)

گام ۰: لیست ممنوعه را مقداردهی اولیه کنید. قرار دهید، $i = ۱$، $s = False$ و $z = ۰$. یک جواب اولیه q در $\{۰,۱\}^{n-۱}$ به طور تصادفی انتخاب کنید. اگر $q \in SD$، آن‌گاه قرار دهید $s = True$ و به گام ۵ بروید.

گام ۱: مطابق با بهترین حرکت با معیار تابع δ در بین تمامی حرکت‌های غیرممنوع در $N(q)$، یک وسیله دفاعی از q حذف کنید. این گام را تا وقتی که q در SD قرار گیرد، ادامه دهید.

گام ۲: قرار دهید $i \leftarrow i + ۱$. اگر $i < n$، آن‌گاه به گام ۳ بروید. در غیر این صورت، قرار دهید $i = ۱$ و $z = z + ۱$. اگر $z > N_z$، آن‌گاه الگوریتم را متوقف کنید. جواب تقریبی به دست آمده در واقع بهترین جواب برای تابع هدف مسأله DLP (۲—۷) در بین تمامی جواب‌های جستجو شده می‌باشد. در غیر این صورت، اگر $s = False$، آن‌گاه قرار دهید $s = True$ و به گام ۴ بروید، در غیر این صورت، قرار دهید $s = False$ و به گام ٦ بروید.

گام ۳: برای q، اگر وسیله‌ای بر روی رأس $v_i \in V \setminus \{\zeta\}$ قرار نگرفته، یک وسیله دفاعی روی آن رأس در نظر گرفته و به گام ۱ بروید. در غیر این صورت، وسیله موجود را حذف کرده و به گام ۵ بروید.

گام ۴: مطابق با بهترین حرکت با معیار تابع δ در بین تمامی حرکت‌های غیرممنوع در $N(q)$، یک وسیله دفاعی را از بردار q حذف کنید. این گام N_{TD} بار تکرار می‌شود.

گام ۵: مطابق با بهترین حرکت با معیار تابع هدف در بین تمامی حرکت‌های غیرممنوع در $N(q)$، یک وسیله دفاعی به q اضافه کنید. این گام مادامی که q در SD قرار داشته باشد و $f(q,p)$ بهبود یابد، انجام می‌شود. به گام ۲ برگردید.

گام ٦: مطابق با بهترین حرکت با معیار تابع هدف در بین تمامی حرکت‌های غیرممنوع در $N(q)$، یک وسیله دفاعی به q اضافه کنید. این گام مادامی که q در شرط $A_q < b$ صدق کند، ادامه می‌یابد. به گام ۱ بروید.

۳.۳ نتایج محاسباتی

در این جا نتایج حاصل از پیاده‌سازی روش جستجوی ممنوعه بر روی چند نمونه عددی از مسأله DLP نشان داده می‌شود. در این نمونه‌ها، که در [۲۸] آمده است، یک شبکه شامل $n = ۲۰۰$ رأس و $r = ۱۹۷۰$ یال در نظر گرفته شده است. وزن‌های هر یال به طور تصادفی از مجموعه $\{۱, \ldots, ۱۰۰۰\}$ انتخاب شده‌اند. انرژی اولیه مهاجم را $\bar{\alpha} = ۲۰۰$ و ظرفیت دفاعی هر یال $\beta = ۵۰$ در نظر گرفته شده است. همچنین $m = ۵۰$ محدودیت در نظر گرفته شده که در آن درایه‌های ماتریس A، یعنی a_{ij} $(i = ۱, \ldots, m$ و $j = ۱, \ldots, n)$ به طور تصادفی از $\{۱, \ldots, ۱۰۰۰\}$ انتخاب شدند. برای ماتریس b نیز، هر b_i $(i = ۱, \ldots, n)$ به صورت زیر تعریف می‌شود:

$$b_i = \theta_i \cdot \sum_{j=۱}^{n} a_{ij}$$

که در آن θ_i $(i = ۱, \ldots, m)$ یک مقدار ثابت تصادفی بین $[۰/۱, ۰/۲]$ است.
همچنین پارامترهای مربوط به روش جستجوی ممنوعه به قرار زیر تنظیم شده است.
$T = ۴۰$، $N_{TD} = ۷$ و $N_z = ۱۰۰$.
در این محاسبات روش TS، ۲۰ مرتبه بر روی هر یک از ۴ نمونه عددی تصادفا تولید شده انجام شده است. نتایج حاصل در جدول ۱ آمده است.

نمونه	۱	۲	۳	۴
بهترین مقدار تابع هدف	۱۰۴	۸۸	۱۲۲	۸۴
میانگین مقادیر تابع هدف	۹۹/۲	۸۵/۸	۱۱۳/۲	۸۳/۵
بدترین مقدار تابع هدف	۹۵	۷۳	۱۰۰	۸۳
میانگین زمان محاسبات (ثانیه)	۲۳۲/۹	۲۰۵/۵	۲۴۱/۲	۲۵۲/۲

جدول ۱: نتایج محاسباتی روش TS بر روی چهار نمونه عددی از DLP

فصل ۴

حل DLP به روش جستجوی همسایگی متغیر

۱.۴ روش جستجوی همسایگی متغیر (VNS)

جستجوی همسایگی متغیر (VNS) یک روش فوق ابتکاری[۱]، یا یک چارچوب برای ساختن روش‌های ابتکاری است، که هدف آن حل مسایل بهینه‌سازی سرتاسری[۲] یا ترکیبیاتی[۳] می‌باشد [۲۰]. ایده اصلی این روش در واقع تغییر سیستماتیک همسایگی، در ترکیب با یک روند جستجوی موضعی می‌باشد. از همان ابتدای پیدایش، VNS به شیوه‌های مختلف توسعه یافته و در زمینه‌های متعدد به کار رفته است، به گونه‌ای که می‌توان ادعا کرد تقریبا در تمام علوم هرجا مسأله بهینه‌سازی ریاضی وجود داشته است، برای حل آن با استفاده از روش VNS اقدام شده است. زمینه‌هایی از قبیل صنعت، ارتباطات، بهداشت و غیره از اصلی‌ترین بسترهای به کارگیری روش VNS می‌باشد.

---

[۱] Meta-heurestic Method
[۲] Global Optimization
[۳] Combinatorial Optimization

در این فصل ما قواعد پایه‌ای VNS و اصلی‌ترین توسعه‌های آن را مطالعه خواهیم کرد.
یک مسأله بهینه‌سازی به طور کلی به صورت زیر فرموله می‌شود:

$$\min\{f(x)|x \in X, X \subseteq S\} \qquad (۱-۴)$$

که در آن S، X، x و f به ترتیب نشانده فضای جواب، مجموعه شدنی، یک جواب شدنی و تابع هدف با مقادیر حقیقی می‌باشند. اگر S یک مجموعه متناهی اما با اندازه بزرگ باشد، یک مسأله بهینه‌سازی ترکیبیاتی تعریف می‌شوند. اگر $S = \mathbb{P}^n$، به بهینه‌سازی پیوسته خواهیم رسید.
جواب $x^* \in X$ یک جواب بهینه است اگر:

$$f(x^*) \leq f(x), \forall x \in X \qquad (۲-۴)$$

یک الگوریتم دقیق برای مسأله (۴-۱) در صورت وجود، یک جواب بهینه x^* برای مسأله می‌یابد و یا ثابت می‌کند که مسأله دارای جواب شدنی نیست. یعنی $X = \varnothing$.
به علاوه، در عمل، زمان لازم برای انجام این کار باید متناهی باشد (و بسیار طولانی نیز نباشد). در صورتی که با یک تابع پیوسته سروکار داشته باشیم، منطقی به نظر می‌رسد که یک محک توقف در نظر گرفته شود، بدین معنی که الگوریتم با یافتن یک جواب شدنی x^* که دارای شرایط زیر است، متوقف می‌شود، یعنی عدد مثبت وبه قدر کافی کوچک ε به گونه‌ای موجود باشد که:

$$f(x^*) < f(x) + \varepsilon, \forall x \in X \qquad (۳-۴)$$

یا

$$\frac{f(x^*) - f(x)}{f(x^*)} < \varepsilon, \forall x \in X \qquad (۴-۴)$$

به ازای یک .
بسیاری از نمونه‌های علمی به فرم مسأله (۴-۱)، که در تحقیق در عملیات یا سایر

زمینه‌ها موجودند، نمی‌توانند توسط الگوریتم‌های دقیق در زمان معقول حل شوند. از تئوری پیچیدگی زمانی [۲۳] می‌دانیم که بسیاری از مسایل دارای پیچیدگی زمانی نمایی هستند، به گونه‌ای که هیچ الگوریتمی با تعداد گام‌های چندجمله‌ای برای حل کردن آن‌ها وجود ندارد.
دلایلی برای روی‌آوردن به روش‌های ابتکاری هستند که به سرعت یک جواب بهینه تقریبی را به دست می‌دهند، یا گاهی جواب بهینه واقعی را به دست می‌دهند اما اثباتی برای بهینه بودن آن ارائه نمی‌دهند.
علاوه بر جلوگیری از صرف زمان طولانی برای محاسبات، روش‌های ابتکاری برای حل یک مشکل دیگر نیز تلاش می‌کنند و آن بهینه موضعی است. یک نقطه بهینه موضعی x_L برای (۴—۱) به صورت زیر می‌باشد:

$$f(x_L) \leq f(x), \forall x \in N(x_L) \cap X \qquad (۴—۵)$$

که در آن $N(x_L)$ نشان دهنده یک همسایگی از نقطه x_L می‌باشد.
اگر نقاط بهینه موضعی زیادی موجود باشند، گستردگی آن‌ها در فضا ممکن است بسیار زیاد باشد.
به علاوه، مقدار بهینه سرتاسری $f(x^*)$ ممکن است تا حد زیادی به مقدار متوسط بهینه‌های موضعی یا حتی با بهترین مقدار در بین تمام بهینه‌های موضعی، تفاوت داشته باشد. با این وجود، روش‌های زیادی برای فرار از بهینه‌های موضعی، یا به طور دقیق‌تر ناحیه‌ای که شامل آن‌ها است، وجود دارد.
روش‌های فوق ابتکاری در واقع چارچوب‌هایی برای ساختن روش‌های ابتکاری برای حل مسایل بهینه‌سازی می‌باشند. برای اطلاع از مشهورترین آن‌ها و بسیاری از کاربردهای موفق آن‌ها می‌توانید به کتاب‌های جامع [۳]، [۱۱] و [۲۴] مراجعه کنید.
جستجوی همسایگی متغیر (VNS) یک روش فوق ابتکاری است که به طور سیستماتیک از ایده تغییر همسایگی، هم در جهت کاهش به سمت مینیمم‌های موضعی و هم برای فرار از دره‌های شامل آن‌ها، استفاده می‌کند. VNS تا حد زیادی به سه اصل زیر متکی است:

اصل ۱) یک مینیمم موضعی نسبت به یک همسایگی، لزوما برای سایر همسایگی‌ها مینیمم موضعی نمی‌باشد.

اصل ۲) یک مینیمم سرتاسری در واقع یک مینیمم موضعی نسبت به تمام ساختارهای همسایگی می‌باشد.

اصل ۳) برای بسیاری از مسایل، مینیمم‌های موضعی نسبت به یک یا چند همسایگی، نسبتا به یکدیگر نزدیک می‌باشند.

این مشاهده آخر به طور ضمنی بیان می‌کند که یک نقطه بهینه موضعی اغلب اطلاعاتی در مورد نقطه بهینه سرتاسری فراهم می‌آورد. برای نمونه، در مواردی ممکن است چندین متغیر در هردوی آن‌ها دارای مقدار یکسان باشند.
برخلاف بسیاری از روش‌های فوق ابتکاری، قالب پایه‌ای VNS و توسعه‌های آن ساده هستند و به تعداد کمی پارامتر نیاز دارند. بنابراین، علاوه بر فراهم آوردن جواب‌های بسیار خوب، VNS اغلب به طرق ساده‌تری نسبت به سایر روش‌ها، نگرش راجع به دلایل چنین عملکردی را ارائه می‌دهد که در جای خود می‌تواند به روش‌های اجرایی کارآمدتر و پیچیده‌تر منتهی می‌شود.

۱.۱.۴ جستجوی موضعی

یک روش ابتکاری جستجوی موضعی[۴] شامل انتخاب یک جواب اولیه x، یافتن یک جهت کاهش از x، درون یک همسایگی مانند $N(x)$، و حرکت به سمت نقطه مینیمم $f(x)$ در همسایگی $N(x)$ در همان جهت می‌باشد. اگر هیچ جهت کاهشی موجود نباشد، این روش ابتکاری متوقف می‌شود؛ در غیر این صورت، تکرار می‌شود. معمولا جهتی که باعث بیشترین کاهش می‌شود و به عنوان بهترین بهبود شناخته می‌شود، به کار می‌رود. این مجموعه قواعد در الگوریتم ۲.۴، جایی که ما فرض کرده‌ایم یک جواب اولیه x داده شده است، ارائه شده‌اند. خروجی الگوریتم شامل یک مینیمم

[۴] Local Search

موضعی و مقدار تابع هدف آن است. توجه کنید که برای هر $x \in X$ یک ساختار همسایگی مانند $N(x)$ تعریف شده است. در بهینه‌سازی گسسته، این همسایگی معمولا شامل همه بردارهایی است که از x با یک تغییر ساده به دست آمده‌اند، برای مثال در مورد بهینه‌سازی ۱ — ۰، تغییر یک یا دو مولفه از بردار می‌تواند یک تغییر ساده به شمار آید. سپس، در هر گام همسایگی $N(x)$ از x به طور کامل جستجو می‌شود. از آن جایی که این فرآیند ممکن است زمان‌بر باشد، به جای این کار می‌توان از یک روش ابتکاری اولین کاهش[۵] استفاده کرد. در این روش، بردارهای $x_i \in N(x)$ به طور سیستماتیک شمرده می‌شوند و به محض این که یک جهت کاهش یافت شد، حرکت در جهت آن آغاز می‌شود. این فرآیند در الگورتیم ۱.۴، نشان داده شده است.

الگوریتم ۱.۴ (تابع اولین بهبود (ورودی: بردار x))

۱. مادامی که شرط $(f(x) < f(x'))$ برقرار است، گام‌های ۲ تا ۵ را تکرار کن؛
۲. مقدار صفر را به i اختصاص بده: بردار x را به x' اختصاص بده؛
۳. مادامی که شرط $(i < |N(x)|, f(x) \geq f(x_i))$ برقرارند، گام‌های ۴ و ۵ را تکرار کن؛
۴. مقدار $i+1$ را به i اختصاص بده:
۵. بین دو بردار x و x_i (که $x_i \in N(x)$) آن را که دارای مقدار تابع هدف کمتر است، به x اختصاص بده؛

[۵] First Descent Method

الگوریتم ۲.۴ (تابع بهترین بهبود (ورودی: بردار x))

۱. مادامی که شرط $(f(x) < f(x'))$ برقرار است، گام‌های ۲ و ۳ را تکرار کن؛
۲. بردار x را به بردار x' اختصاص بده؛
۳. برداری در $N(x)$ را که دارای کمترین مقدار تابع هدف است، به x اختصاص بده:

۲.۱.۴ چارچوب اصلی

فرض کنید $N_k, (k = ۱, \ldots, k_{\max})$ نشان دهنده یک مجموعه متناهی از ساختارهای همسایگی باشد که از پیش انتخاب شده‌اند و همچنین $N_k(x)$ نشان دهنده مجموعه جواب‌های درون kامین همسایگی از x باشد. به علاوه، نماد $N'_k, (k = ۱, \ldots, k'_{\max})$ را برای توصیف کاهش موضعی استفاده می‌کنیم. همسایگی‌های N_k و N'_k ممکن است از یک یا چند تابع متریک القا شده باشد که بر روی ناحیه جواب یعنی S تعریف شده‌اند. یک جواب بهینه x_{opt} (یا بهینه سرتاسری) یک جواب شدنی است که به ازای آن مینیمم مسأله (۴—۱) حاصل شود. نقطه $x' \in X$ را یک مینیمم موضعی مسأله (۴—۱) نسبت به N_k می‌نامیم اگر هیچ جواب xای با شرایط زیر وجود نداشته باشد،

$$x \in N_k(x') \subseteq X \text{ s.t.} \quad f(x) < f(x') \qquad (۶—۴)$$

برای حل مسأله (۴—۱) با استفاده از چندین همسایگی، اصول ۱ تا ۳ به سه روش متفاوت می‌توانند به کار روند. (۱) قطعی (۲) تصادفی (۳) قطعی و تصادفی. ابتدا در الگوریتم ۳.۴ گام‌های تابع تغییر همسایگی که بعدا به کار خواهد رفت، معرفی می‌شوند.
تابع تغییر همسایگی مقدار جدید $f(x')$ را با مقدار قبلی $f(x)$ که در همسایگی kام به دست آمده مقایسه می‌کند (خط ۱). اگر بهبودی حاصل شد، k به مقدار اولیه خود

باز می‌گردد و مقدار جدید x بروز می‌شود (خط ۲). در غیر این صورت، همسایگی بعدی مورد مطالعه قرار می‌گیرد (خط ۳).

الگوریتم ۳.۴ (تابع تغییر همسایگی (ورودی: بردار x، بردار x' و k))

۱. اگر $f(x') < f(x)$ می‌باشد، آن‌گاه:
۲. x' را به x اختصاص بده؛ ۱ را به k اختصاص بده؛
درغیر این صورت
۳. مقدار $k+1$ را به k اختصاص بده؛

۳.۱.۴ روش کاهش همسایگی متغیر (VND)

روش کاهش همسایگی متغیر[۶] (VND) وقتی به کار می‌رود که برای تغییر همسایگی از یک شیوه قطعی استفاده می‌شود. گام‌های این روش در الگوریتم ۴.۴ نشان داده شده‌اند. در توصیف تمام الگوریتم‌های بعدی، فرض شده است که یک جواب اولیه x داده شده است. بیشتر روش‌های جستجوی موضعی در فاز کاهش خود از تعداد بسیار کمی از همسایگی‌ها استفاده می‌کنند (معمولا یک یا دو همسایگی یعنی $k'_{\max} \leq ۲$). توجه کنید که جواب نهایی باید یک مینیمم موضعی نسبت به تمام همسایگی‌های با اندیس $k'_{\max}$ باشد؛ بنابراین وقتی از VND استفاده می‌شود تغییرات برای تغییر دستیابی به یک بهینه سرتاسری نسبت به یک ساختار همسایگی ساده، بزرگترند.

[۶] Variable Neighbourhood Descent

الگوریتم ۴.۴ (تابع VND (ورودی: بردار x و $k'_{\max}$))

۱. مادامی که بهبودی در جواب حاصل می‌شود، گام‌های ۲ تا ۵ را تکرار کن؛
۲. مقدار ۱ را به k اختصاص بده؛
۳. مادامی که $k \neq k'_{\max}$، گام‌های ۴ و ۵ را تکرار کن؛
۴. برداری در $N'_k(x)$ که دارای کمترین مقدار تابع هدف است، به x' اختصاص بده؛
۵. تابع تغییر همسایگی را با ورودی‌های x، x' و k اجرا کن؛

۴.۱.۴ جستجوی همسایگی متغیر کاهش‌یافته (RVNS)

روش جستجوی همسایگی کاهش‌یافته[۷] (RVNS) به این صورت می‌باشد که نقاط نمونه به طور تصادفی از همسایگی $N_k(x)$ انتخاب می‌شوند و هیچ کاهشی اعمال نمی‌شود. سپس، مقادیر این نقاط جدید با مقدار بهینه جاری مقایسه می‌شود و در صورت لزوم به روزرسانی انجام می‌شود. فرض می‌کنیم که یک شرط پایانی انتخاب شده است. به طور مثال، حداکثر زمان محاسبات برای CPU یعنی $t_{\max}$ و یا حداکثر تعداد تکرارهایی که در آن بهبودی حاصل نشود. برای ساده‌سازی توصیف الگوریتم‌ها ما از $t_{\max}$ استفاده می‌کنیم. بنابراین، RVNS از دو پارامتر استفاده می‌کند: $t_{\max}$ و $k_{\max}$. گام‌های این روش در الگوریتم ۵.۴ نشان داده شده‌اند. در خط ۴، ما یک نقطه را به طور تصادفی از kامین همسایگی x انتخاب می‌کنیم.

[۷] Reduced Variable Neighbourhood Search

الگوریتم ۵.۴ (تابع RVNS (ورودی: بردار x، $k_{\max}$ و $t_{\max}$))

۱. مادامی که شرط ($t \leq t_{\max}$) برقرار است، گام‌های ۲ تا ٦ را تکرار کن؛
۲. مقدار ۱ را به k اختصاص بده؛
۳. مادامی که $k \neq k_{\max}$، گام‌های ۴ و ۵ را تکرار کن؛
۴. یک نقطه تصادفی از $N_k(x)$ انتخاب کن و آن را به x' اختصاص بده؛
۵. تابع تغییر همسایگی را با ورودی‌های x، x' و k اجرا کن؛
٦. زمان صرف شده برای محاسبات CPU را به t اختصاص بده؛

RVNS در نمونه‌های خیلی بزرگ که در آن فرآیند جستجوی موضعی هزینه‌بر است، مفید می‌باشد. مشاهده شده است که بهترین مقدار برای پارامتر $k_{\max}$، اغلب ۲ می‌باشد. به علاوه، معمولا حداکثر تکرارهایی که در آن بهبودی حاصل نشود به عنوان شرط توقف به کار می‌رود.

۵.۱.۴ روش جستجوی همسایگی متغیر پایه‌ای (BVNS)

روش VNS پایه‌ای[۸] (BVNS) تغییرات قطعی و تصادفی همسایگی‌ها را با هم ترکیب می‌کند. گام‌های این روش در الگوریتم ٦.۴ ارئه شده‌اند.

الگوریتم ٦.۴ (تابع BVNS (ورودی: بردار x، $k_{\max}$ و $t_{\max}$))

۱. مادامی که شرط ($t \leq t_{\max}$) برقرار است، گام‌های ۲ تا ۷ را تکرار کن؛

[۸] Basic Variable Neighbourhood Search

۲. مقدار ۱ را به k اختصاص بده؛
۳. مادامی که $k \neq k_{\max}$، گام‌های ۴، ۵ و ٦ را تکرار کن؛
۴. یک نقطه تصادفی از $N_k(x)$ انتخاب کن و آن را به x' اختصاص بده؛
۵. تابع اولین بهبود را با ورودی x' اجرا کن و خروجی آن را به x'' اختصاص بده:
٦. تابع تغییر همسایگی را با ورودی‌های x، x'' و k اجرا کن؛
۷. زمان صرف شده برای محاسبات CPU را به t اختصاص بده؛

معمولا همسایگی‌های متوالی N_k به طور شبکه‌ای در نظر گرفته می‌شوند. مشاهده کنید که نقطه x' در گام ۴ به طور تصادفی تولید شده است تا از ایجاد دور جلوگیری شود.

در گام ۵، معمولا جستجوی موضعی اولین بهبود (الگوریتم ۱.۴) به کار می‌رود. با این وجود، می‌توان آن را با روش بهترین بهبود جایگزین کرد (الگوریتم ۲.۴).

٦.۱.۴ جستجوی همسایگی متغیر کلی (GVNS)

توجه کنید که می‌توان جستجوی موضعی در گام پنجم الگوریتم ٦.۴ را با VND (الگوریتم ۴.۴) جایگزین کرد. با استفاده از این رویکرد (NNS/NND) می‌توان به موفق‌ترین نتایج حاصله دست یافت. گام‌های جستجوی همسایگی متغیر کلی[۹] (GVNS) در الگورتیم ۷.۴ ارائه گشته‌اند.

الگوریتم ۷.۴ (تابع GVNS (ورودی: بردار x، $k'_{\max}$، $k_{\max}$ و $t_{\max}$))

۱. مادامی که شرط ($t \leq t_{\max}$) برقرار است، گام‌های ۲ تا ۷ را تکرار کن؛
۲. مقدار ۱ را به k اختصاص بده؛
۳. مادامی که $k \neq k_{\max}$، گام‌های ۴، ۵ و ٦ را تکرار کن؛

[۹] General Variable Neighbourhood Search

۴. یک نقطه تصادفی از $N_k(x)$ انتخاب کن و آن را به x' اختصاص بده؛
۵. تابع VND را با ورودی‌های x' و $k'_{\max}$ اجرا کن و خروجی آن را به x'' اختصاص بده:
۶. تابع تغییر همسایگی را با ورودی‌های x، x'' و k اجرا کن؛
۷. زمان صرف شده برای محاسبات CPU را به t اختصاص بده؛

۷.۱.۴ روش‌هایی برای بهبود کارایی VNS پایه‌ای

گاهی اوقات روش VNS پایه‌ای نتایج خیلی خوبی را فراهم نمی‌کند و می‌بایست به یکی از روش‌های زیر اصلاح شود:

۱) **بهترین بهبود در مقابل اولین بهبود.** استراتژی‌های اولین یا بهترین بهبود را در یک جستجوی محلی به طور تجربی مورد آزمایش قرار دهید. نتایج زیر توسط تجارب اخیر پیشنهاد شده‌اند:
اگر جواب اولیه به طور تصادفی انتخاب شود، از قاعده اولین بهبود استفاده کنید، اما اگر یک روش ابتکاری ساختاری به کار رود، قاعده بهترین بهبود استفاده می‌شود.

۲) **همسایگی را محدود کنید.** دلیل بدرفتاری هر جستجوی موضعی ممکن است جستجوهای غیر ضروری در میان جواب‌های یک همسایگی باشد. سعی کنید زیرمجموعه‌های معینی از همسایگی را تشخیص دهید و فقط آن‌ها را جستجو کنید. در بهترین حالت، قاعده‌ای را بیابید تا به طور خودکار جواب‌هایی از همسایگی را که دارای مقادیر بدتری از نقاط جاری هستند، از جریان جستجو خارج کند.

۳) **یک روش VND را برای جایگزینی جستجوی موضعی برگزینند و از GVNS استفاده کنید.** VND، چندین ساختار همسایگی ممکن را تجزیه و تحلیل کنید،

اندازه‌های آن‌ها را برآورد کنید، آن‌ها را مورد آزمایش قرار دهید و بهترین‌ها را نگه دارید.

۴) **پارامترهای گوناگون را آزمایش کنید.** تنها پارامتر مرکزی برای VNS عبارتست از $k_{\max}$ که می‌بایست به صورت تجربی انتخاب شود. با این وجود، روند الگوریتم معمولا نسبت به $k_{\max}$ حساس نیست، و برای دست‌یابی به یک VNS بدون پارامتر، می‌توان مقدار آن را مطابق با یک پارامتر ورودی تنظیم کرد: برای مثال، برای مسأله p—میانه[۱۰] $k_{\max} = p$، برای مینیمم مربعات[۱۱] $k_{\max} = m$ و ....

۲.۴ به کارگیری روش VNS برای حل DLP

در این بخش، یک روش ابتکاری را بر مبنای روش VNS ارائه می‌کنیم. در این الگوریتم، قبل از اعمال بدنه اصلی روش VNS، طی سه گام ابتدایی جواب تصادفی اولیه بهبود و سپس به الگوریتم VNS وارد می‌شود. در این مراحل پس از چک کردن شدنی بودن جواب تصادفی، طی یک الگوریتم ساده، جواب تصادفی اصلاح می‌شود. در این اصلاح جواب اولیه، از خاصیت (۳—۱) استفاده شده است، بدین معنی که تلاش شده تا جایی که شرط شدنی بودن برقرار است، هر چه می‌توان تعداد وسایل دفاعی را افزایش داد.

برای ورود به الگوریتم VNS، لازم است پارامتر $k_{\max}$ و همچنین ساختار همسایگی $N_k, (k = ۱, \ldots, k_{\max})$ مشخص شود. در الگوریتم طرح شده، ما ساختار همسایگی kام را به صورت زیر تعریف می‌کنیم: $N_k(q)$ مجموعه تمام جواب‌هایی است (بردارهای صفر و یک) که دقیقا در k مؤلفه با x تفاوت دارند. همچنین انتخاب ما برای $k_{\max}$ عبارت است از تعداد وسایل دفاعی در جواب ورودی به الگوریتم VNS، به عبارت دیگر، تعداد ۱ها در بردار q به دست آمده پس از طی سه گام ابتدایی الگوریتم.

---

[۱۰] p-center Problem

[۱۱] Least Square

سایر گام‌ها مطابق بدنه اصلی VNS طراحی شده و تنها چک کردن شدنی بودن در آن‌ها گنجانده شده است. همچنین حداکثر زمان $t_{\max}$ برای اجرای الگوریتم به عنوان یک شرط توقف الگوریتم در نظر گرفته شده است.
قبل از بیان صورت گام به گام الگوریتم طرح شده، ابتدا الگوریتم ساده‌ای که برای اصلاح جواب اولیه تصادفی q اعمال شده است را نشان می‌دهیم. همان طور که اشاره شد، در این الگوریتم سعی شده با توجه به خاصیت (۳—۱)، تعداد وسایل دفاعی تا حد امکان افزایش یابد.

الگوریتم ۸.۴ (الگوریتم اصلاح جواب اولیه q برای ورود به VNS)

گام ۰: قرار دهید $t = ۱$ و $s =$ تعداد صفرهای بردار q.
گام ۱: tامین صفر در بردار q را به یک تبدیل کنید و بردار جدید را q' بنامید (ترتیب شمارش همان ترتیب معمولی صعودی است.).
گام ۲: اگر q' شدنی باشد به گام ۳ برو و اگر شدنی نباشد به گام ۵ برو.
گام ۳: قرار بده $q = q'$ و $t = t + ۱$.
گام ۴: اگر $t \leq s$ بود به گام ۱ برگرد و در غیر این صورت الگوریتم را متوقف کن.
گام ۵: قرار بده $t = t + ۱$ و به گام ۴ برو.

حال به سراغ معرفی گام به گام الگوریتم اصلی می‌پردازیم:

الگوریتم ۹.۴ (به کارگیری VNS برای حل مسأله DLP)

گام ۰: بردار $q \in \{۰, ۱\}^{n-۱}$ را به طور تصادفی انتخاب کنید. اگر q شدنی بود به گام ۲ برو، در غیر این صورت به گام ۱ برو.
گام ۱: مطابق با بهترین حرکت با معیار تابع هدف در $N_۱(q)$، یک وسیله دفاعی از q حذف کنید. این گام را تا وقتی که q در SD قرار گیرد، ادامه دهید.

گام ۲: مطابق الگوریتم ۸.۴، q را اصلاح کن.
گام ۳: قرار بده $k = ۱$.
گام ۴: در همسایگی kام q، یک مقدار تصادفی انتخاب کن و به q' اختصاص بده.
اگر q' شدنی نباشد، همانند روند گام ۱، q را شدنی کن.
گام ۵: در همسایگی مرتبه اول q'، اولین بهبود شدنی را به q'' اختصاص بده.
گام ٦: الگوریتم تغییر همسایگی را با ورودی (q, q'', k) انجام بده.
گام ۷: اگر $k < k_{\max}$ به گام ۴ برو، در غیر این صورت به گام ۸ برو.
گام ۸: قرار بده $t = CPUtime$.
گام ۹: اگر $t \leq t_{\max}$، به گام ۳ برو، در غیر این صورت الگوریتم را متوقف کن.

۳.۴ نتایج محاسباتی

در این قسمت نتایج محاسباتی به کارگیری الگوریتم ۹.۴، یعنی روش VNS، برای حل چند نمونه عددی از DLP ارائه گشته‌اند. ۴ نمونه عددی مورد بررسی، نمونه‌های ارائه شده در فصل ۳ هستند که برای پیاده‌سازی روش TS تولید شدند. تنها پارامتری که در VNS باید تعریف شود $t_{\max}$ است.
ابتدا ما حداکثر زمان اجرا را برابر میانگین زمان اجرا در ۴ نمونه حل شده با استفاده از TS در نظر گرفتیم. این کار به منظور مقایسه کارآیی دو روش در زمان‌های مساوی است. نتایج به دست آمده در جدول ۲ ارائه شده‌اند.
اما روش VNS دارای توانایی بالایی در تقریب جواب بهینه در زمان بالا می‌باشد. برای نشان دادن کارآیی این روش، در محاسبه بعدی، ما $t_{\max} = ۱۰۰۰s$ در نظر گرفتیم. با این انتخاب، جواب‌های به دست آمده از VNS، بهبود قابل توجهی یافتند. نتایج این محاسبه در جدول ۳ ارائه شده‌اند.
در این محاسبات نیز، همانند روش TS، در هر دو حالت، روش VNS را ۲۰ مرتبه بر روی هر کدام از نمونه‌ها به کار برده‌ایم.

نمونه	۱	۲	۳	۴
بهترین مقدار تابع هدف	۱۰۴	۸۵	۱۲۰	۸۴
میانگین مقادیر تابع هدف	۹۸/۲	۸۰/۸	۱۱۱/۵	۸۳/۲
بدترین مقدار تابع هدف	۹۳	۷۰	۹۸	۸۲
حداکثر زمان محاسبات (ثانیه)	۲۳۳	۲۰۶	۲۴۱	۲۵۲

جدول ۲: نتایج محاسباتی روش VNS بر روی چهار نمونه عددی از DLP با حداکثر زمان معادل با میانگین زمان اجرای روش TS

نمونه	۱	۲	۳	۴
بهترین مقدار تابع هدف	۱۰۴	۹۰	۱۲۲	۸۵
میانگین مقادیر تابع هدف	۱۰۲/۳	۸۵/۸	۱۱۸/۲	۸۴/۵
بدترین مقدار تابع هدف	۹۸	۷۸	۱۱۱	۸۴
حداکثر زمان محاسبات (ثانیه)	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰

جدول ۳: نتایج محاسباتی روش VNS بر روی چهار نمونه عددی از DLP با حداکثر زمان محاسبات ۱۰۰۰ ثانیه

فصل ۵

مسأله مکان‌یابی تدافعی پیوسته

همان طور که مشاهده شد، مسأله DLP که اولین بار در سال ۲۰۰۸ توسط اونو و کاتاگیری [۲۸] معرفی شد، یک مسأله مکان‌یابی مدرن می‌باشد. این مسأله در واقع ترکیبی از یک برنامه ریزی دو مرحله‌ای و یک مسأله بهینه‌سازی ۰—۱ می‌باشد. یعنی اولا دارای دو تصمیم‌گیرنده می‌باشد که دارای اولویت تصمیم‌گیری متفاوت هستند. تصمیم گیر مقدم، یعنی مدافع می‌خواهد وسایل دفاعی خود را در یک شبکه مکان‌یابی کند. در این صورت بردار تصمیم مدافع، در واقع یک بردار ۰ و ۱ و معرف مکان وسایل دفاعی در شبکه می‌باشد. در سطح بعدی تصمیم‌گیری، مهاجم قرار دارد که با توجه به فرض‌های پایه‌ای مسئله، سعی می‌کند در رأسی از شبکه قرار گیرد که کوتاه‌ترین فاصله را تا هسته دارد.

در این جا علاقه‌مندیم که توسعه‌ای از مسأله مکان‌یابی تدافعی DLP را معرفی و بسط دهیم. همان طور که در مدل اولیه مسأله DLP مشاهده شد، مدافع تنها تصمیم می‌گیرد که آیا در یک رأس شبکه وسیله دفاعی قرار دهد یا خیر. از این رو بردار تصمیم مدافع در واقع یک بردار ۰ و ۱ است. یعنی $q \in \{0,1\}^{n-1}$. اما در کاربردهای حقیقی، ارزش دفاعی وسایل دفاعی یکسان نیستند. در واقع هر وسیله دفاعی دارای توان دفاعی مشخص است. در کاربرهای استراتژیک نظامی، چنین توان

دفاعی را می‌توان به راندمان دفاعی تسهیلات، تعداد نیروهای دفاعی، منابع مالی تجهیز تسهیلات دفاعی برای رسیدن به توان دفاعی خاص و یا سایر پارامترهای دیگر تعبیر کرد. در کاربردهای دیگری نظیر چینش سیستم‌های دفاعی در بازی‌هایی از قبیل فوتبال، و در نظر گرفتن نظام‌های دفاعی در شبکه‌های کامپیوتری و ...نیز تجهیزات دفاعی دارای توان دفاعی متفاوت می‌باشند. این واقعیت‌ها ما را بر آن داشت تا مدلی جدید از مسأله DLP را معرفی کنیم.

در نوع جدید مسأله DLP، که آن را مسأله مکان‌یابی تدافعی تعمیم‌یافته[۱] (GDLP) یا به طور معادل مسأله مکان‌یابی تدافعی پیوسته[۲] (CDLP) می‌نامیم، فرض بر این است که مدافع دارای یک توان کل دفاعی است که می‌تواند هر کسری از آن را در رأس‌های شبکه قرار دهد. در کاربردهای حقیقی، این توان دفاعی می‌تواند در وسایل دفاعی متفاوت به منظور ارتقای راندمان دفاعی آن‌ها اعمال شود. مثلا، مقدار منبع مال مشخصی را که مدافع می‌تواند آن را برای تجهیز وسایل دفاعی‌اش و رساندن راندمان دفاعی آن به هر میزان، هزینه کند، می‌توان به عنوان توان دفاعی مدافع در نظر گرفت. توجه می‌کنیم که در واقع این مسأله تعمیم مسأله DLP است. اگر بعد از حل مسأله GDLP، بردار تصمیم q به گونه‌ای باشد که مؤلفه‌های غیر صفر آن برابر مقدار ثابت β باشند، در واقع این بردار می‌تواند جواب مسأله DLP باشد، به گونه‌ای که مقدار ثابت β همان ظرفیت دفاعی ثابت در DLP است. دقیقا به همین دلیل، مسأله جدید را مسأله مکان‌یابی تدافعی تعمیم‌یافته نامیدیم.

از سوی دیگر، این تعمیم باعث شد تا متغیرهای تصمیم، مقادیر حقیقی در بازه $[۰, \gamma]$ را اختیار کند، که در آن γ توان کل دفاعی مدافع می‌باشد. از این رو، در واقع این تعمیم به نوعی به پیوسته‌سازی مسأله بهینه‌سازی DLP منجر می‌شود. دقیقا به همین دلیل، می‌توان عنوان مسأله مکان‌یابی تدافعی پیوسته را نیز به آن اختصاص داد.

با این توضیحات به مدل‌سازی مسأله CDLP می‌پردازیم.

همانند مسأله DLP، فرض کنیم n رأس $v_۱$، ...و v_n و r یال در شبکه موجود می‌باشد و

---

[۱] Generalized Defensive Location Problem

[۲] Continuous Defensive Location Problem

مجموعه‌های $V = \{v_1, \ldots, v_n\}$ و E به ترتیب معرف مجموعه رئوس و یال‌های شبکه باشند. فرض می‌کنیم که موقعیت استراتژیک دفاعی (هسته) یکی از رأس‌های شبکه، c، می‌باشد یعنی $c \in V$. بدون از دست رفتن کلیت فرض کنیم مهاجم بر روی رأس nام، یعنی v_n قرار دارد. این رأس را به ζ نشان می‌دهیم. همچنین $w_{ij} > 0$ را وزن یال متصل کننده v_i به v_j در نظر می‌گیریم. سپس، وضعیت تصمیم‌گیری مدافع و مهاجم را در نظر می‌گیریم. فرض می‌کنیم کل توان دفاعی مدافع $\gamma \in \mathbb{R}$ است. همچنین فرض می‌کنیم مدافع می‌تواند هر کسری از توان دفاعی کل خود، γ، را در همه رأس‌های شبکه به جز ζ قرار دهد. به ازای هر i، $i \in \{1, \ldots, n-1\}$، $q_i \in \mathbb{R}$ را مقدار توان دفاعی اختصاص یافته به رأس iام در نظر می‌گیریم. با توجه به محدود بودن کل توان دفاعی داریم:

$$\sum_{i=1}^{n-1} q_i = \gamma. \qquad (۱-۵)$$

با این فرض‌ها، $q \equiv (q_1, \ldots, q_{n-1})$ بردار تصمیم مدافع خواهد بود. از سوی دیگر، مهاجم می‌خواهد یک مسیر از ζ به c بیابد.

فرموله‌سازی تابع هدف مهاجم مشابه با مسأله DLP است. یعنی فرض می‌کنیم که مهاجم دارای انرژی اولیه $\bar{\alpha} > 0$ است و انرژی مهاجم به ازای حرکت از رأس i به رأس j به اندازه w_{ij} و به ازای برخورد با هرکدام از وسایل دفاعی به اندازه توان دفاعی آن وسیله کاسته می‌شود. پس اگر مهاجم از رأس v_i به v_j برود، مقدار انرژی کاسته شده آن از رابطه زیر حاصل می‌شود

$$\bar{w}(e_{ij}, q) = w_{ij} + q_j \qquad (۲-۵)$$

و نهایتا مهاجم وقتی انرژی آن به کمتر از صفر برسد، از بین می‌رود.

سایر موارد مشابه DLP است، که در معادلات (۲-۳)، (۲-۴) و (۲-۵) در فصل ۲ بیان شد.

برای تعریف مجموعه‌های شدنی مهاجم و مدافع نیز مانند قبل عمل می‌کنیم. فرض کنیم m تعداد شرایط و ماتریس سمت چپ و سمت راست را به ترتیب A، b می‌نامیم

$$A \equiv \begin{pmatrix} a_{11} & \cdots & a_{1n-1} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn-1} \end{pmatrix}, \quad b \equiv \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \qquad (۳-۵)$$

با این تعریف، SD، مجموعه شدنی مدافع به صورت زیر تعریف می‌شود.

$$SD = \{q \in \mathbb{R}^{n-1} | Aq = b\}. \qquad (۴-۵)$$

مجموعه شدنی مهاجم نیز، همانند قبل تمام مسیرهای شدنی از ζ به c می‌باشد. با این فرض‌ها، فرمول‌بندی مسأله GDLP به صورت زیر خواهد بود:

$$\begin{aligned} \max_{q} \quad & f(q,p) \\ & \text{که در آن } p \text{له زیر است: جواب مسا} \\ \min_{p} \quad & f(q,p) \\ \text{s.t.} \quad & \\ & q \in SD, \\ & p \in SI, \\ & \textstyle\sum_{i=1}^{n-1} q_i = \gamma, \\ & 0 \le q_i. \end{aligned} \qquad (۵-۵)$$

همان طور که مشاهده شد، تعمیم مسأله DLP به CDLP باعث شد تا متغیرهای تصمیم مقادیر حقیقی در بازه $[0, \gamma]$ را اختیار کنند. در واقع این تعمیم به نوعی به پیوسته‌سازی مسأله DLP شد. به این ترتیب، این مدل مسأله DLP را می‌توان با روش‌های ابتکاری بهینه‌سازی پیوسته حل کرد. روش‌هایی که عموما دارای توانایی بسیار زیادی در پیدا کردن نقطه بهینه سرتاسری می‌باشند در فصل بعد، برای حل مسأله CDLP یک روش بهینه‌سازی پیوسته به نام روش شبه الکترومغناطیس را به کار گرفته‌ایم. این روش که به تفصیل در ادامه معرفی خواهد شد، توانایی خوبی در حل مسایل بهینه‌سازی پیوسته نشان داده است.

شایان توجه است که فرض جدیدی که به مسأله DLP اضافه کردیم و به مسأله CDLP رسیدیم، باعث پیدا شدن خاصیت مفیدی در مجموعه جواب‌های مسأله CDLP گشته است. قضیه زیر نشان می‌دهد که فضای جواب مسأله CDLP، یک فضای محدب است.

قضیه ۱.۵ فضای جواب‌های شدنی مسأله CDLP محدب است.

اثبات. فرض کنیم $q^1 = (q_1^1, q_2^1, \ldots, q_{n-1}^1)$ و $q^2 = (q_1^2, q_2^2, \ldots, q_{n-1}^2,)$ دو بردار دلخواه در فضای شدنی CDLP باشند. با توجه به این فرض داریم:

$$q^1, q^2 \in SD \quad \Rightarrow \quad Aq^1 = b,\ Aq^2 = b$$

فرض کنیم $\lambda \in (0, 1)$ دلخواه باشد.

$$A(\lambda q^1 + (1-\lambda)q^2) = \lambda(Aq^1) + (1-\lambda)(Aq^2) = \lambda b + (1-\lambda)b = b$$
$$\Rightarrow \quad \lambda q^1 + (1-\lambda)q^2 \in SD \qquad (1)$$

همچنین داریم:

$$\sum_{i=1}^{n-1} q_i^1 = \gamma, \quad \sum_{i=1}^{n-1} q_i^2 = \gamma$$

در نتیجه:

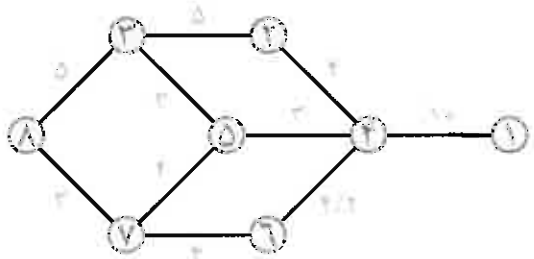
$$\sum_{i=1}^{n-1}(\lambda q_i^1 + (1-\lambda)q_i^2) = \lambda \sum_{i=1}^{n-1} q_i^1 + (1-\lambda)\sum_{i=1}^{n-1} q_i^2 = \lambda\gamma + (1-\lambda)\gamma = \gamma \quad (2)$$

با توجه به نتایج (۱) و (۲)، $\lambda q^1 + (1-\lambda)q^2$ نیز در فضای شدنی CDLP قرار دارد. بنابراین این فضا محدب است. □

با توجه به محدب بودن فضای جواب‌های شدنی CDLP روش‌های بهینه‌سازی پیوسته ابتکاری می‌تواند اغلب به طور موثری برای حل این مسأله به کار گرفته شود. در فصل آتی یکی از روش‌های ابتکاری بهینه‌سازی سرتاسری را که دارای کارایی

مناسب است، برای حل مسأله CDLP به کار خواهیم گرفت.
در این قسمت، علاقه‌مندیم مثالی ارائه کنیم که مشخص می‌کند مدل‌سازی مسأله مکان‌یابی تدافعی در قالب CDLP کاراتر از DLP است. به عبارت دیگر، اگر بتوان هر کسری از توان دفاعی را در رأس‌های شبکه مستقر کرد، می‌توان مهاجم را در فواصل دورتری از هسته متوقف ساخت.

مثال ۱.۵ شبکه ۸ رأسی شکل ۱.۵ را در نظر می‌گیریم. فرض کنیم هسته در رأس شماره ۱ و مهاجم در رأس شماره ۸ قرار دارند. همچنین فرض کنیم که حداکثر توان دفاعی که مهاجم می‌تواند در شبکه مستقر کند، ٦ واحد و نیز انرژی اولیه مهاجم ۱۵ واحد باشد. شکل ۱.۵ شبکه مذکور و وزن یال‌های آن را نشان می‌دهد:



شکل ۱.۵: شبکه ۸ راسی مثال ۱.۵

اگر بخواهیم مسأله مکان‌یابی تدافعی را بر روی شبکه فوق، در قالب یک مسأله DLP (برنامه‌ریزی ۰ و ۱) حل کنیم، توان دفاعی تسهیلات باید باهم برابر در نظر گرفته شوند، یعنی کل توان دفاعی مدافع که برابر ٦ واحد است به چند وسیله تدافعی با توان دفاعی مساوی اختصاص یابد. به سادگی دیده می‌شود با در نظر گرفتن چنین توزیعی در شبکه فوق، بهترین مقدار تابع هدف ۱۰ خواهد بود، یعنی در بهترین حالت مهاجم در رأس شماره ۴ متوقف می‌شود. در حالی که اگر این مسأله در قالب CDLP حل شود، با در نظر گرفتن بردار تصمیم مدافع به صورت

$q \equiv (۰, ۰, ۰, ۲, ۲/۱, ۰/۹, ۱)$، می‌توان مهاجم را در رأس شماره ۵ متوقف کرد که در این حالت مقدار تابع هدف ۱۳ خواهد بود.

این مثال نشان دهنده برتری مدل‌سازی مسأله مکان‌یابی تدافعی در قالب CDLP نسبت به DLP است.

فصل ٦

حل مسأله مکان‌یابی تدافعی پیوسته به روش شبه الکترومغناطیس

١.٦ روش شبه الکترومغناطیس

در بهینه‌سازی سرتاسری تصادفی، الگوریتم‌های جمعیت‌محور[١] با انتخاب تصادفی نقاط نمونه از فاصله شدنی آغاز به کار می‌کنند و با توجه به مقادیر تابع هدف برای نقاط نمونه، نواحی جذب مشخص می‌شوند. سپس یک مکانیسم خاص برای بهره‌برداری از این نواحی منتخب، به کار می‌رود. در الگوریتم‌های ژنتیک، این مکانیسم به عملگرهای زاد و مرگ[٢] و جهش[٣] مربوط می‌شود [١٩]، در حالی که در روش‌های دوفازی[٤] جستجوی در ناحیه شدنی به وسیله نمونه‌برداری تصادفی و در پی آن فرایند صعود از تپه[٥] [٢٧] یا روش‌های مبتنی بر گرادیان[٦] [٩] هدایت می‌شود.

[١] Population-based Algorithms
[٢] Reproduction and Crossover Operators
[٣] Mutation Operators
[٤] Two-phase Methods
[٥] Hill Climbing Process
[٦] Gradient-based Method

یک روش بهینه‌سازی سرتاسری که اولین بار توسط بیربیل و فنگ [۱] برای حل مسایل بهینه‌سازی مورد استفاده قرار گرفت، روش شبه‌الکترومغناطیس[۷] (EM) می‌باشد. در این بخش به توضیح مکانیسم شبه‌الکترومغناطیسم می‌پردازیم.

در روش EM، مکانیسمی معرفی می‌شود که نقاط را وادار می‌سازد به سمت نواحی دارای جاذبه بیشتر همگرا شوند و از سوی دیگر از نواحی دارای دافعه بیشتر دور شوند [۱]. این ایده در واقع اساس شباهت این مکانیسم و مکانیسم جاذبه—دافعه[۸] در تئوری الکترومغناطیس می‌باشد.

مشابه با آن چه در تئوری الکترومغناطیس مقدماتی وجود دارد، می‌توان هرکدام از نقاط نمونه را به عنوان یک ذره باردار که در فضا رها شده است، در نظر گرفت. دراین رویکرد، مقدار بار هر ذره به مقدار تابع هدف برای آن نقطه مربوط می‌شود. این مقدار بار همچنین تعیین کننده میزان جاذبه یا دافعه هر نقطه نسبت به سایر جمعیت نمونه می‌باشد. هر چه مقدار تابع هدف بهتر باشد، میزان جاذبه بیشتر است.

بعد از محاسبه این بارها، آن‌ها برای یافتن مسیری استفاده می‌شود که هرکدام از نقاط می‌بایست در تکرارهای بعدی در راستای آن حرکت کنند. این مسیر را با محاسبه نیروی برآیند که از سوی دیگر نقاط بر هر نقطه وارد می‌شود، انتخاب می‌شود. همانند نیروهای الکترومغناطیس، این نیرو با محاسبه مجموع برداری تک‌تک نیروهای اعمال شده از سوی هرکدام از نقاط که به صورت جداگانه محاسبه می‌شوند، به دست می‌آید.

در این جا لازم است به این نکته اشاره شود که اگرچه شباهت با تئوری الکترومغناطیس باعث بروز این ایده شده است، تفاوت‌های قابل توجهی نیز وجود دارند که در ادامه به آن اشاره خواهد شد.

در نهایت، همانند الگوریتم‌های جمعیت‌محور ترکیبی [۱۲]، از یک روش جستجوی محلی برای بهبود مقادیر توابع هدف در میان جمعیت نمونه استفاده می‌شود.

روش EM به طور کلی بر روی مسائل بهینه‌سازی سرتاسری با متغیرهای کراندار

[۷] Electromagnetism-like Mechanism
[۸] Attraction-Repulsion Mechanism

به صورت کلی زیر اعمال می‌شود:

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in [l,u] \end{array} \qquad (۱-٦)$$

که در آن $[l,u] = \{x \in \mathbb{R}^n | l_k \leq x_k \leq u_k, k = ۱,\ldots,n\}$.
به خاطر سادگی در حفظ خاصیت شدنی بودن، این مسأله به طور گسترده‌ای به وسیله روش‌های جستجوی تصادفی[۹] [٦] و همچنین روش‌های مستقیم[۱۰] [۱۵] مورد مطالعه قرار گرفته است.
پارامترهای زیر برای نشان دادن خصوصیات مسأله به کار گرفته می‌شوند:

n: بعد مسأله (تعداد مؤلفه‌های نقاط)
u_k: کران بالای برای مؤلفه kام
l_k: کران پایین برای مؤلفه kام
$f(x)$: تابعی که می‌خواهیم آن را کمینه سازیم

حال به معرفی چارچوب کلی روش EM و زیربرنامه‌های آن می‌پردازیم.

۱.۱.٦ چارچوب کلی EM

روش EM شامل چهار فاز می‌شود. این فازها عبارتند از: مرحله آغازین الگوریتم، محاسبه نیروی برآیند اعمال شده بر روی هر ذره، حرکت در جهت این نیرو و به کارگیری جستجوی محلی برای استخراج مینیمم موضعی.

۲.۱.٦ مرحله آغازین

مرحله آغازین جهت انتخاب تصادفی m نقطه نمونه از ناحیه شدنی، که یک فضای nبعدی است، به کار می‌رود. فرض بر این است که هر مؤلفه از یک نقطه به طور

[۹] Random Search Methods
[۱۰] Direct Methods

یکنواخت بین کران بالا و کران پایین توزیع شده است. بعد از این که یک نقطه از فضا انتخاب شد، مقدار تابع هدف با استفاده از اشاره‌گر $f(x)$ محاسبه می‌شود (شبه‌کد ۱.٦، خط ٦). در انتهای این فاز m نقطه نمونه تعیین گشته‌اند و نقطه‌ای که دارای بهترین (کمترین) مقدار تابع هدف است به عنوان x^{best} مشخص می‌شود.

شبه‌کد ۱.٦ (فاز اول: مرحله آغازین)

```
1: for i = 1 to m do
2:    for k = 1 to n do
3:       λ ← U(0,1)
4:       x_k^i ← l_k + λ(u_k − l_k)
5:    end for
6:    Calculate f(x^i)
7: end for
8: x^best ← argmin {f(x^i), ∀i}
```

۳.۱.٦ جستجوی محلی

این فاز الگوریتم، جهت جمع‌آوری اطلاعات موضعی برای نقطه x^i (یکی از نقاط نمونه) به کار می‌رود. پارامترهای LSITER و δ که به این فاز انتقال می‌یابند، به ترتیب نشان دهنده تعداد تکرارها و ضریب جستجوی موضعی می‌باشند.

این روند به صورت زیر تکرار می‌شود:

ابتدا، بیشترین طول گام شدنی مطابق با پارامتر δ محاسبه می‌شود. (شبه‌کد ۲.٦، خط ۲). سپس، برای هر نقطه نمونه، با تغییر هر مؤلفه به دنبال بهبود آن نقطه هستیم (خطوط ۵ تا ۱۳). به ازای هر مؤلفه، نقطه x^i به یک نقطه موقتی به نام y اختصاص می‌یابد، تا اطلاعات اولیه ذخیره شوند. پس از آن، یک عدد تصادفی به عنوان طول گام انتخاب می‌شود و نقطه y در آن جهت حرکت داده می‌شود. اگر نقطه y در جریان LSITER تکرار به یک نقطه بهتر رسید، نقطه x^i با نقطه y جایگزین

می‌شود و جستجوی موضعی برای x^i پایان می‌یابد (خطوط ۱۴ تا ۱۷). نهایتا، نقطه بهینه جاری دوباره تعیین می‌شود (خط ۲۲).

شبه‌کد ۲.٦ (فاز دوم: جستجوی محلی)

```
1: counter ← 1
2: Lenght ← δ (max_k {u_k − l_k})
3: for i = 1 to m do
4:    for k = 1 to n do
5:       λ1 ← U(0,1)
6:       while counter < LSITER do
7:          y ← x^i
8:          λ2 ← U(0,1)
9:          if λ1 > 0.5 then
10:            y_k ← y_k + λ2(Lenght)
11:         else
12:            y_k ← y_k − λ2(Lenght)
13:         end if
14:         if f(y) < f(x^i) then
15:            x^i ← y
16:            counter ← LSITER − 1
17:         end if
18:         counter ← counter + 1
19:      end while
20:   end for
21: end for
22: x^best ← argmin {f(x^i), ∀i}
```

این رویه، یک الگوریتم جستجوی تصادفی خطی ساده است که به صورت مؤلفه به مؤلفه عمل می‌کند. این روند به هیچ گونه اطلاعات راجع به مشتق تابع برای اجرای جستجوی محلی نیاز ندارد. به جای استفاده از دیگر روش‌های جستجوی محلی قدرتمند [٢٦]، این رویه به کار برده شده است تا نشان داده شود که حتی با این

روش ساده نیز، الگوریتم EM خواص همگرایی قطعی را از خود نشان می‌دهد.

۴.۱.٦ محاسبه بردار نیروی برآیند

اصل برهم‌نهی[11] از تئوری الکترومغناطیس بیان می‌کند که نیروی وارد بر یک نقطه از جانب دیگر نقاط متناسب است با حاصلضرب بارهای نقاط و معکوس فاصله بین آن‌ها [۵].

در هر تکرار، بار هر نقطه بر طبق مقدار تابع هدف آن محاسبه می‌شود. با این وجود، در این روش ابتکاری، بار هر نقطه ثابت نیست و در هر تکرار تغییر می‌کند.

بار هر نقطه x^i که با نماد q^i مشخص می‌شود، تعیین کننده قدرت جاذبه یا دافعه نقطه x^i است. این بار به صورت زیر به دست می‌آید:

$$\forall i, q^i = \exp\left(-n\frac{f(x^i) - f(x^{\text{best}})}{\sum_{k=1}^{m}(f(x^k) - f(x^{\text{best}}))}\right) \qquad (٦-٢)$$

به این ترتیب، نقاطی که دارای مقدار تابع هدف بیشتری هستند، مقدار بار بیشتری خواهند داشت. مقدار کسر بالا در بعد فضا یعنی n ضرب شده است، زیرا در ابعاد بالاتر تعداد نقاط در جمعیت نمونه بیشتر خواهد شد. در نتیجه، ممکن است کسر بالا خیلی کوچک شود، و به مشکلات محاسباتی در محاسبه تابع نمایی برخورد کنیم.

مقدار بار هر نقطه در جمعیت، متناسب با اثربخشی نسبی مقدار تابع هدف آن نقطه تعریف می‌شود. به وضوح، این انتخاب لزوما نه تنها انتخاب و نه بهترین انتخاب آن‌هاست. می‌توان از روش محاسباتی دیگر، که نقاط را بر اساس مقدار تابع هدفشان رتبه‌بندی می‌کند، استفاده کرد.

به این نکته توجه کنید که بر خلاف بارهای الکتریکی، هیچ علامتی به بار یک نقطه در معادله (٦—٢) اختصاص نیافته، در عوض، در مورد جهت یک نیرو بین دو نقطه بعد از مقایسه مقادیر تابع هدف آن‌ها، تصمیم گیری می‌شود. بنابراین، نیروی برآیند

[11] Superposition Principle

F^i که بر نقطه i وارد می‌شود توسط معادله زیر محاسبه می‌شود:

$$F^i = \sum_{j \neq i}^{m} \left\{ \begin{array}{lll} (x^j - x^i)\frac{q^i q^j}{\|x^j - x^i\|^2} & \text{if} & f(x^j) < f(x^i) \\ (x^i - x^j)\frac{q^i q^j}{\|x^j - x^i\|^2} & \text{if} & f(x^j) \geq f(x^i) \end{array} \right\} \qquad (٣-٦)$$

همان طور که در شبه‌کد ٣.٦، خطوط ٨-٧، دیده می‌شود، بین دو نقطه، آن نقطه‌ای که دارای مقدار تابع هدف بهتری می‌باشد، دیگری را جذب می‌کند. بالعکس، نقطه‌ای که دارای مقدار تابع هدف بدتری است، دیگری را دفع می‌کند (خطوط ١٠-٩). از آن جایی که x^{best} دارای کمترین مقدار تابع هدف است، تمام نقاط دیگر جمعیت را جذب می‌کند.

شبه‌کد ٣.٦ (فاز سوم: محاسبه بردار نیروی برآیند)

```
1: for i = 1 to m do
2:    q^i ← exp( -n (f(x^i) - f(x^best)) / (Σ_{k=1}^{m} (f(x^k) - f(x^best))) )
3:    F^i ← 0
4: end for
5: for i = 1 to m do
6:    for i = 1 to m do
7:       if f(x^j) < f(x^i) then
8:          F^i ← F^i + (x^j - x^i) q^i q^j / ||x^j - x^i||^2 {Attraction}
9:       else
10:         F^i ← F^i - (x^j - x^i) q^i q^j / ||x^j - x^i||^2 {Repulsion}
11:      end if
12:   end for
13: end for
```

وقتی الگوریتم به دقت بررسی می‌شود، می‌توان دید که تعیین یک جهت از طریق محاسبه نیروی برآیند، شبیه تخمین آماری گرادیان f می‌باشد. با این وجود، تخمینی

که به وسیله این روش ابتکاری محاسبه می‌شود، متفاوت است، چرا که این جهت به فاصله اقلیدسی بین دو نقطه بستگی دارد. به عبارت دیگر، نقاطی که به قدر کافی به هم نزدیک می‌شوند، ممکن است یک دیگر را به جهتی هدایت کنند که با جهت محاسبه شده توسط تخمین آماری گرادیان متفاوت است.

٥.١.٦ حرکت در جهت نیروی برآیند

بعد از محاسبه بردار نیروی برآیند F^i، نقطه iام در جهت این نیرو با طول گام تصادفی، مطابق با معادله (٦—۴)، حرکت داده می‌شود. در این جا فرض شده که طول گام تصادفی λ به طور یکنواخت بین ۰ و ۱ توزیع شده است. به وضوح، روش‌های توزیع دیگر زیادی می‌توانند در محاسبه این طول گام به کار روند. اما برای تسهیل محاسبات، توزیع یکنواخت به کار برده شده است.
طول گام λ به طور تصادفی انتخاب می‌شود تا احتمال این که نقاط در امتداد این جهت به سمت مناطق بکر ناحیه شدنی حرکت کنند، صفر نباشد.
در معادله (٦—۴)، RNG برداری است که مؤلفه‌های آن نشان دهنده حرکت شدنی مجاز به سمت کران بالا، u^k، یا کران پایین، l^k، می‌باشد (شبه‌کد ۴.٦، خطوط ۱۰—٦). علاوه بر این، نیروی وارده بر هر نقطه نرمال شده تا بتوان شدنی بودن را حفظ کرد.

$$x^i = x^i + \lambda \frac{F^i}{\|F^i\|}(RNG), \quad i = ۱, ۲, \ldots, m \qquad (۴—٦)$$

شبه‌کد ۴.٦، شبه‌کدهای مربوط به زیربرنامه MOVE را نشان می‌دهد. توجه کنید که نقطه بهینه x^{best} حرکت داده نمی‌شود و به تکرارهای بعدی منتقل می‌شود (خط ۲). این امر بدین معناست که می‌توان از محاسبه نیروی برآیند در مورد نقطه بهینه جاری در مرحله محاسبه بردار نیروی برآیند صرفه نظر کرد.

شبه‌کد ۴.٦ (فاز چهارم: حرکت در جهت نیروی برآیند)

```
1: for i = 1 to m do
2:   if i ≠ best then
3:     λ ← U(0,1)
4:     F_i ← F_i / ||F_i||
5:     for k = 1 to n do
6:       if F_k^i > 0 then
7:         x_k^i ← x_k^i + λ F_k^i (u_k − x_k^i)
8:       else
9:         x_k^i ← x_k^i + λ F_k^i (x_k^i − l_k)
10:      end if
11:    end for
10:  end if
12: end for
```

٦.١.٦ معیار توقف

در [۱]، بیربیل و فنگ از حداکثر دفعات تکرار به عنوان شرط توقف روش EM استفاده کردند. بر اساس نتایج محاسباتی آن‌ها، عموما تعداد ۲۵ تکرار به ازای هر بعد (یعنی MAXITER=25n) برای همگرایی به نقطه بهینه، در مورد توابع با درجه سختی متوسط، رضایت‌بخش بود.

یک معیار توقف دیگر که می‌تواند مورد استفاده قرار گیرد، عبارت است از تعداد تکرارهای متوالی که در آن‌ها نقطه بهینه تغییر نکند. به عبارت دیگر، اگر نقطه بهینه در تعداد معینی از تکرارها تغییر نکرد، الگوریتم می‌تواند متوقف شود. با این وجود تصمیم‌گیری در این مورد می‌بایست به طور دقیق مورد مطالعه قرار گیرد، چرا که ممکن است الگوریتم قبل از همگرایی به نقطه بهینه سراسری، متوقف شود. از طرف

دیگر، با توقف سریعتر، می‌توان از محاسبات غیر ضروری اجتناب کرد.

٧.١.٦ مدل‌های سه‌گانه EM

در [١]، مکانیسم EM با استفاده از سه روش متفاوت جستجوی محلی مورد آزمون قرار گرفته است. ابتدا، به طور کلی روند جستجوی محلی از برنامه اصلی حذف شد. در دومین روش، روند جستجوی محلی برای تمام نقاط جمعیت نمونه به کار برده شده و در سومین روش، به کارگیری روند جستجوی محلی تنها برای نقاط بهینه جاری مورد آزمایش قرار گرفته است.

در روش اول یعنی حذف جستجوی محلی از الگوریتم، اطلاعات موضعی نقاط از دست می‌رود. این امر باعث می‌شود که به دلیل عدم جمع آوری اطلاعات اضافی، تعداد متوسط محاسبات تابعی و در نتیجه زمان اجرای الگوریتم کاهش یابد. در [١] نشان داده شده است که در این حالت گرچه EM برای بسیاری از مسائل جواب تقریبی خوبی ارائه می‌دهد در بعضی مسائل دقت مقادیر تابع هدف به قدر کافی خوب نیست. روش دوم، یعنی به کارگیری جستجوی محلی برای تمام نقاط جمعیت نمونه دارای دو مزیت می‌باشد. اولا، بخش‌های جذب کننده ناحیه شدنی دقیق‌تر بررسی می‌شوند و ثانیا، نقاط دفع شده شانس بیشتری برای حرکت به سمت نقاط بهینه‌ای که هنوز پیدا نشده‌اند، دارند. از این رو، در این حالت عملکرد الگوریتم به طور قابل توجهی بهبود می‌یابد اما در مقابل تعداد محاسبات تابعی و در نتیجه زمان اجرای الگوریتم افزایش می‌یابد.

در روش سوم، یعنی به کارگیری جستجوی محلی تنها برای نقطه بهینه جاری، تعداد محاسبات تابعی و دقت جواب‌ها در حالت تعادل قرار می‌گیرد. در این حالت، تعداد متوسط محاسبات و در نتیجه زمان اجرای الگوریتم قابل مقایسه با روش اول، در حالی که کیفیت جواب‌ها قابل مقایسه با روش دوم می‌باشند.

٢.٦ به کارگیری EM برای حل CDLP

همان طور که در بخش قبل مشاهده شد، روش EM بر روی مسایل بهینه‌سازی با فرم کلی (٦—١) به کار می‌رود. برای تبدیل مدل مسأله CDLP به فرم فوق‌الذکر، می‌توان از روش‌های مختلف استفاده کرد.
یکی از این روش‌های پرکاربرد در تبدیل مسایل بهینه‌سازی با محدودیت به مسایل بهینه‌سازی بدون محدودیت، روش تابع جریمه می‌باشد. در این روش، محدودیت‌ها با هزینه سنگینی به تابع هدف منتقل می‌شوند، به گونه‌ای که نقض محدودیت‌های هزینه سنگینی را در تابع هدف منجر شود و از این رو کمترین انحراف از محدودیت‌ها رخ دهد. این روش به طور مختصر در فصل اول توضیح داده شده است.
در به کارگیری روش EM برای حل CDLP، ما نیز از روشی مشابه با روش تابع جریمه استفاده کرده‌ایم. محدودیت‌هایی که در مسأله CDLP وجود دارند را می‌توان با یک جریمه زیاد به تابع هدف برد.
لازم به ذکر است که مسأله CDLP، ماکزیمم‌سازی است، لذا می‌بایست تغییرات لازم بر روی تعاریف در EM انجام شود. به جای این کار، می‌توان به جای ماکزیمم کردن تابع هدف، از مسأله مینمم‌سازی قرینه آن استفاده کرد. با این توضیحات، فرم مسأله CDLP برای حل به وسیله EM به صورت زیر خواهد بود:

$$\min_{q} \quad -f(q,p) + M[\|Aq-b\|^{2} + (\sum_{i=1}^{n-1} q_i - \gamma)^{2}]$$

که در آن p له زیر است: جواب مسا

$$\begin{aligned} \min_{p} \quad & f(q,p) \\ \text{s.t.} \quad & \\ & p \in SI, \\ & q_i \in [0,\gamma]. \end{aligned} \qquad (٦—٥)$$

که در آن M یک عدد بسیار بزرگ است، $\|.\|$ نرم اقلیدسی و سایر نمادها مشابه با صورت اصلی مسأله CDLP می‌باشند.
همان طور که دیده می‌شود، نقض شرایط SD و شرط $\sum_{i=1}^{n-1} q_i = \gamma$ هزینه سنگینی را

به تابع هدف تحمیل می‌کند. بنابراین، شدنی بودن جواب از این طریق می‌تواند حفظ شود. با رسیدن به فرمول فوق، می‌توان روش EM را پیاده‌سازی کرد. البته این تنها و لزوما بهترین روش برای حل مسایل بهینه‌سازی پیوسته نیست، اما می‌تواند حداقل از لحاظ زمان اجرا، برتری خوبی نسبت به بسیاری دیگر از روش‌های بهینه‌سازی پیوسته داشته باشد. نتایج پیاده‌سازی EM بر مسأله CDLP در بخش بعد خواهد آمد.

٣.٦ نتایج محاسباتی

در این بخش، نتایج محاسباتی حاصل از پیاده‌سازی روش الکترومغناطیس را بر روی نمونه‌های عددی از مدل اصلاح شده مسأله CDLP، یعنی مدل (٦—٥) ارائه می‌کنیم. مثال‌های عددی مورد استفاده، همان ۴ نمونه عددی DLP می‌باشد که در محاسبات با روش‌های TS و VNS مورد استفاده قرار گرفتند. تفاوت این نمونه‌ها، وجود پارامتر ثابت γ به جای β می‌باشد. در این محاسبه، مقدار کل توان دفاعی مدافع، $\gamma = ۱۰۰$ در نظر گرفته شده است. همچنین معیار توقف، مطابق با معیار توقف پیشنهادی توسط بیربیل و فنگ [۱]، یعنی حداکثر دفعات تکرار، تعیین شده است. حداکثر دفعات تکرار در محاسبات ما ۲۰۰ می‌باشد. به علاوه با توجه به حداکثر مقادیر تابع هدف که در هر ۴ نمونه کمتر از ۱۰۰۰ می‌باشد، مقدار M، مقدار جریمه، را ۱۰۰۰۰ در نظر گرفته‌ایم. همچنین روش EM را ۱۰ مرتبه بر روی هر کدام از نمونه‌ها به کار برده‌ایم. نتایج محاسباتی در جدول ۴ ارائه شده است.

علاوه بر اعمال روش EM، برای مشاهده کارایی این روش، یک الگوریتم جستجوی تصادفی نیز برای حل CDLP به کار گرفته شد. این روش از یک نمونه‌گیری اولیه تصادفی از ناحیه شدنی و سپس تکرار فرآیند جستجوی موضعی همانند فاز دوم الگوریتم EM (بخش ٣.١.٦) تشکیل شده است. این روش را با انتخاب حداکثر زمان اجرای ۵۰۰۰ ثانیه بر روی CDLP به کار برده‌ایم. نتایج محاسباتی در جدول ۵ آمده است.

نمونه	۱	۲	۳	۴
بهترین مقدار تابع هدف	۱۱۷	۹۲	۱۵۳	۱۰۳
میانگین مقادیر تابع هدف	۱۱۲/۴	۸۹/۵	۱۳۸/۲	۱۰۰/۴
بدترین مقدار تابع هدف	۹۹	۸۴	۱۱۲	۹۳
میانگین زمان محاسبات (ثانیه)	۳۷۵/۴	۳۰۲/۳	۳۲۱/۸	۳۷۲/۳

جدول ۴: نتایج محاسباتی روش EM بر روی چهار نمونه عددی از CDLP

نمونه	۱	۲	۳	۴
بهترین مقدار تابع هدف	۸۹	۸۴	۹۳	۹۰
حداکثر زمان محاسبات (ثانیه)	۵۰۰۰	۵۰۰۰	۵۰۰۰	۵۰۰۰

جدول ۵: نتایج محاسباتی روش جستجوی تصادفی بر روی چهار نمونه عددی از CDLP

فصل ۷

نتیجه‌گیری

در این بخش ابتدا می‌خواهیم نتایج حاصل از پیاده‌سازی دو روش TS و VNS را بر روی مسئله DLP با هم مقایسه کنیم. سپس به بررسی نتایج حاصل از به کارگیری روش EM برای حل CDLP و مقایسه آن‌ها با روش جستجوی تصادفی خواهیم پرداخت و در انتها پیشنهادهایی برای مطالعات بعدی ارائه خواهیم کرد.

۱.۷ بررسی نتایج به دست آمده

در فصل دوم این تحقیق، حل مسئله DLP با روش TS مورد مطالعه قرار گرفت. در جدول شماره ۱، نتایج حاصل از پیاده‌سازی این روش بر روی چهار نمونه عددی از DLP که به طور تصادفی تولید شده بودند، نشان داده شد. در این محاسبات الگوریتم TS بر روی هر کدام از نمونه‌ها ۲۰ مرتبه اجرا شد و بهترین مقدار تابع هدف به دست آمده، مقدار متوسط تابع هدف، بدترین مقدار تابع هدف به دست آمده و متوسط زمان محاسبات در ۲۰ مرتبه محاسبه، به ترتیب در جدول شماره ۱ نشان داده شدند.
در فصل چهارم، یک روش حل بر مبنای الگوریتم VNS ارائه شد. ۴ نمونه عددی حل شده توسط الگوریتم VNS پیشنهادی، دقیقا همان ۴ نمونه عددی تصادفا تولید

شده در بخش قبلی و برای به کارگیری روش TS هستند.
در آن بخش، ابتدا متوسط زمان محاسبات که در پیاده‌سازی الگوریتم TS بر روی هر یک از ۴ نمونه به دست آمد و در جدول شماره ۱ مطرح شد، به عنوان حداکثر زمان محاسبات و معیار توقف الگوریتم VNS برای حل همان نمونه در نظر گرفته شد. همچنین هر نمونه ۲۰ مرتبه توسط این روش حل شده است. نتایج حاصل در جدول شماره ۲ ارائه گشت.
با مقایسه جدول‌های شماره ۱ و ۲، مشاهده می‌شود که اگرچه در دو نمونه اول و چهارم، بهترین جواب حاصل از هردو روش یکسان است، ولی در دو نمونه دیگر بهترین جواب حاصل هر ۴ نمونه، مقدار متوسط جواب حاصل از پیاده‌سازی TS بهتر از مقدار مشابه برای روش VNS است. همین طور کمترین مقدار تابع هدف حاصل از پیاده‌سازی TS، برای هر ۴ نمونه بهتر از VNS است. این مشاهدات نشان می‌دهد که در زمان مشابه، کارایی روش TS برای حل DLP بهتر از روش VNS است.
سپس در ادامه آن بخش، حداکثر زمان محاسبات و معیار توقف برای پیاده‌سازی الگوریتم VNS بر روی هر ۴ نمونه عددی معادل ۱۰۰۰ ثانیه در نظر گرفته شد. این کار برای مشاهده کارایی روش VNS در زمان‌های طولانی‌تر است. نتایج حاصل در جدول شماره ۳ ارائه گشتند و این بار نیز، هر نمونه ۲۰ مرتبه توسط روش VNS حل شده است. با مقایسه جدول‌های شماره ۱ و ۳، مشاهده می‌شود که در دو نمونه اول و سوم، بهترین جواب حاصل از هر دو روش TS و VNS یکسان‌اند در حالی که در دو نمونه دیگر بهترین جواب حاصل از VNS بهتر از TS است. همچنین، متوسط مقادیر تابع هدف حاصل از پیاده‌سازی VNS به طور قابل ملاحظه‌ای از متوسط مقادیر تابع هدف حاصل از پیاده‌سازی TS، بهتر است. به علاوه، برتری قابل ملاحظه‌ی روش VNS نسبت به TS، در مقایسه بدترین مقدار تابع هدف حاصل از پیاده‌سازی VNS و TS، به طور آشکارتری دیده می‌شود.
این مشاهدات نشان می‌دهد که در زمان‌های طولانی‌تر، کارایی روش VNS به طور قابل توجهی افزایش می‌یابد و در مقایسه با روش TS، برتری آن نمایان می‌شود.
در فصل ششم، نتایج حاصل از پیاده‌سازی روش EM بر روی مسئله CDLP، در جدول شماره ۴ ارائه شده است. ۴ نمونه عددی به کار رفته در این محاسبات،

همان‌هایی هستند که برای پیاده‌سازی روش TS در بخش آخر از فصل ۳ تولید شدند، با این تفاوت که در این مثال‌ها، به جای استفاده از پارامتر β از پارامتر γ، یعنی کل توان دفاعی، استفاده کردیم. همچنین در هر ۴ نمونه، فرض کردیم $\gamma = ۱۰۰$.
با مقایسه جدول شماره ۴، با جدول‌های ۱، ۲ و ۳، مشخص می‌شود که جواب‌های مدل CDLP همگی به طور قابل توجهی بهتر از جواب‌های مدل DLP می‌باشند. متوسط زمان محاسبات نسبت به روش TS افزایش یافته است. این امر بدیهی به نظر می‌رسد، چرا که در CDLP جستجو در یک فضای پیوسته صورت می‌گیرد.
در ادامه آن بخش، نتایج حاصل از پیاده‌سازی یک روش جستجوی تصادفی را بر روی مسئله CDLP در جدول شماره ۵ ارائه کردیم.در این جا نیز نمونه‌های عددی، همان ۴ نمونه عددی قبل هستند. در این محاسبه، معیار توقف را حداکثر زمان محاسبات و معادل ۵۰۰۰ ثانیه در نظر گرفتیم و محاسبات را تنها ۱ مرتبه بر روی هرکدام از نمونه‌های عددی انجام دادیم.
از مقایسه جدول‌های ۴ و ۵، مشاهده می‌شود که جواب‌های روش EM برتری قابل توجهی هم از نظر مقدار تابع هدف و هم از نظر زمان محاسبات نسبت به روش جستجوی تصادفی دارند. حتی بدترین جواب‌های حاصل از EM در ۲۰ مرتبه به محاسبه بر روی هر نمونه نیز بهتر از بهترین مقدار تابع هدف حاصل از به کارگیری روش جستجوی تصادفی برای حل CDLP می‌باشند.
این مشاهدات، کارایی مناسب روش EM برای حل CDLP را هم از لحاظ کیفیت مقدار تابع هدف و هم از لحاظ زمان محاسبات، به خوبی نشان می‌دهند.

۲.۷ مطالعات آینده

همان طور که گفته شد، مسئله DLP یک مسئله مکان‌یابی نسبتا جدید است [۲۸]. بهترین روش حل ارائه شده تا پیش از این، روش جستجوی ممنوعه [۱] بوده است. در این تحقیق، یک الگوریتم ابتکاری جدید بر مبنای روش VNS برای حل مسئله DLP ارائه شد. در ادامه مسئله جدید CDLP را معرفی کردیم. در این مسئله که

تعمیمی از DLP است، ما مجموعه جواب‌های شدنی را از مجموعه $\{۰,۱\}^{n-۱}$ به یک بازه تغییر دادیم. این تغییر، در واقع باعث پیوسته‌سازی متغیرهای مسئله CDLP شد. سپس، مسئله CDLP را با روش EM، که یک روش بهینه‌سازی سرتاسری است، حل کردیم.

در مطالعات آینده می‌توان مدل‌هایی دیگر از روش VNS را برای حل DLP به منظور بهبود مقادیر تابع هدف و زمان محاسبات به کار برد.

همچنین، می‌توان مسئله CDLP را به روش‌های بهینه‌سازی سرتاسری دیگر حل و نتایج آن را با نتایج فعلی مقایسه کرد. روش‌هایی از قبیل روش «مربع بزرگ، مربع کوچک[۱]»، روش VNS پیوسته و غیره.

---

[۱] BSSS

کتاب‌نامه

[1] Birbil, S., Fang, S. C., An Electromagnetism-like Mechanism for Global Optimization, *Jouranal of Global Optimization*, 263-282, (2003).

[2] Bondy, J.A., Murty, U.S.R. *Graph Theory With Applications*, Springer, (1992).

[3] Burke, E.K., Kendall, G., *Search Methodologies*, Springer, (2005).

[4] Calami, J., Vincente, L., Generating Quadratic Bilevel Programming Problems, *Acm Transactions*, 103-119, (1994).

[5] Cowan, E.W., *Basic Electromagnetism,* Academic Press, (1988).

[6] Demirhan, M., Birbil, S., A geometric Partitioning Metaheuristic for Global Optimization, *Journal of Global Optimization*, 415-435, (1999).

[7] Drezner, Z., Competitive Location Strategies for Tow Facilities, *Regional Science and Urban Economics*, 485-493, (1982).

[8] Eaton, B.C., Lipsey, R.G., The Principle of Minimum Differentiation Reconsidered, *Review of Economic Studies*, 27-48, (1975).

[9] Fletcher, R., Reeves, C., Function Minimization by Conjugate Directions, *Computer Journal*, 149-154, (1964).

[10] Freund, R.M., *Penalty and Barrier Methods for Constrained Optimization*, Massachusetts Institute of Technology, (2004).

[11] Glover, F., Kochenberg, G., *Handbook of Metaheuristics*, Kluwer, (2003).

[12] Glover, J.K., Laguana, M., Genetic Algorithms and Tabu Search, *Computer and Operation Research*, 111-134, (1995).

[13] Hakimi, S.L., On Locating New Facilities in A Competitive Environment, *European Journal of Operational Reasearch*, 29-35, (1983).

[14] Hanafi, S., Ferville, A., An efficient Tabu Search Approach for the 0-1 Multidimentional Knapsack Problem, *European Journal of Operational Reasearch*, 659-677, (1998).

[15] Hauyer, W., Neumaier, A., Global Optimization by Multilevel Coordinate Search, *Journal of Global Optimization*, 331-355, (1999).

[16] Hertz, A., Taillard, E., *A Tutorial on Tabu Search*, (2001).

[17] Hotelling, H., Stability in Competition, *The Economic Journal*, 34-38, (1991).

[18] Karkazis, J., Facility Location in A Competitive Environment, *European Journal of Operational Research*, 294-304, (1989).

[19] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs,* Springer, (1994).

[20] Mladenovic, N., Hansen, P., Variable Neighbourhood Search: Methods and Applications, *Invited Survey*, Springer, 319-360, (2008).

[21] Moreno Perez, J.A., Verdegay, J.L., Fuzzy Location Problems on Networks, *Fuzzy Sets and Systems*, 393-405. (2004).

[22] Okabe, A., Suzuk, A., Stability of Competition for a Larg Number of Firms on A Bounded Tow Dimensional Space, *Environment and Planning*, 1067-1082, (1997).

[23] Papadimitrian, C., *Computational Complexity*, Addison Wesley, (1994).

[24] Reeves, C.R., *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, (1993).

[25] Sakawa, M., Kato, K., *An Interactive Multi-objective Optimization*, Plenum Press, (1993).

[26] Solis, F.J., Wets, R., Minimization by Random Search Techniques, *Mathemaatics of Operatins Research*, 19-30, (1981).

[27] Torn, A., Viitanen, S., Topographical Global Optimization Using Presampled Points, *Journal of Global Optimization*, 267-276, (1994).

[28] Uno, T., Katagiri, H., Single- and Multi-objective Defensive Location Problem on a Network, *European Journal of Operational Research*, 76-84, (2008).

[29] Wendell, R.E., McKelvey, R.D., New Perspective in Competitive Location Theory, *European Journal of Operational Research*, 29-35, (1998).

پیوست

کد برنامه VNS در C++

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<time.h>

#define n 199
#define m 199

int q[n],A[m][n],ones[n]={0},zeros[n]={0},q_dij[n+1]={0},
    pathestimate_i[n+1],pathestimate_c[n+1],fac[n+1],
    alpha,beta;

int graph[n+1][n+1];   // adjacent matrix of the network
int source_i,source_c;   //invador vertex , core vertex
int num_of_vertices;
float b[m];
```

```
void rand_q();
void read();
int feasible(int *);
void change(int *,int *,int);
int one_enumerator(int *);
int zero_enumerator(int *);
void make_feasible(int*,int*);
```

```
void modification(int*,int*);
void first_improvement(int*,int*);
void rand_q_k(int*,int*,int);
int inequality_check(int*,int);
long int f(int *);

 class dijkstra   /* Dijkstra algorithm used in calculating
                     the objective function value */
 {
   private:

    int predecessor_i[n+1],predecessor_c[n+1],mark_i[n+1],
        mark_c[n+1];
   public:
    int minimum_i();
    int minimum_c();
    void initialize();
    void printpath_i(int);
    void printpath_c(int);
    void algorithm();
    void output();
 };
```

```
void dijkstra::initialize() /* initializing Dijkstra
                               algorithm */
{
  for(int i=0;i<num_of_vertices;i++)
  {
     mark_i[i]=mark_c[i]=fac[i]=0;
     pathestimate_i[i]=pathestimate_c[i]=999;
     predecessor_i[i]=predecessor_c[i]=0;
  }
  pathestimate_i[source_i]=0;
  pathestimate_c[source_c]=0;
}
void dijkstra::algorithm() /* main procedure of Dijkstra
                              algorihtm */
{
  initialize();
  int count=0;
  int i,u_i,u_c;
  while(count<num_of_vertices)
  {
     u_i=minimum_i();
     u_c=minimum_c();
     ++count;
     mark_i[u_i]=1;
     mark_c[u_c]=1;
```

```
for(i=0;i<num_of_vertices;i++)
{
   if(graph[u_i][i]>0)   // labeling verteices
                         // considering the invader
                         // vertex as the source vertex
   {
      if(mark_i[i]!=1)
      {
            if(pathestimate_i[i]>pathestimate_i[u_i]+
               graph[u_i][i])
            {
         pathestimate_i[i]=pathestimate_i[u_i]+
                           graph[u_i][i];
         predecessor_i[i]=u_i;
            }
      }
   }
   if(graph[u_c][i]>0)  // labeling verteices
                        // considering the core vertex
                        // as the source vertex
   {
      if(mark_c[i]!=1)
      {
            if(pathestimate_c[i]>pathestimate_c[u_c]+
               graph[u_c][i])
```

```
                {
            pathestimate_c[i]=pathestimate_c[u_c]+
                                graph[u_c][i];
            predecessor_c[i]=u_c;
                }
          }
        }
      }
      fac[u_i]=fac[predecessor_i[u_i]]+
            q_dij[u_i];  // counting the number of defensive
                         // facilities in the path from
                         // the invader to each vertex
  }
}

int dijkstra::minimum_i()
{
  int min=999;
  int i,t;
  for(i=0;i<num_of_vertices;i++)
  {
     if(mark_i[i]!=1)
     {
        if(min>=pathestimate_i[i])
        {
```

```
            min=pathestimate_i[i];
            t=i;
         }
      }
   }
    return t;
}
int dijkstra::minimum_c()
{

   int min=999;
   int i,t;
   for(i=0;i<num_of_vertices;i++)
   {
      if(mark_c[i]!=1)
      {
         if(min>=pathestimate_c[i])
         {
            min=pathestimate_c[i];
            t=i;
         }
      }
   }
   return t;
} // End of Dijkstra algorithm
```

```
void main()   /* Main frame of the VNS method */
{
  time_t time1,time2;
  int feas_out[n],mod_out[n],rand_out[n],
      main_q[n],first_imp_out[n];
  int nei_counter,kmax,sum,duration;
  int timemax=60;
  clrscr();
  for(int row=0;row<m;row++)  // entering the matrix A of
                              // restrictions
  {
     cout<<" row " <<row+1<<" of matrix A:\n";
     for(int column=0;column<n;column++)
        cin>>A[row][column];
  }
  cout<<"\nenter vector b:\n";  // entering the right-hand
                                // vector "b"
  for(int scaler=0;scaler<m;scaler++)
     cin>>b[scaler];

  read();  // entering the adjacent matrix of the network
           // and the initial energy of the invader and
           // the defensive  capacity of each defensive
           // facility
  rand_q();  // generating the first solution randomly
```

```
for(int j=0;j<n;j++)
   main_q[j]=q[j];
if(!feasible(q))  // checking the feasiblity of
                  // the first randomly generated solution
                  // and making it feasible if it's not
{
   make_feasible(q,feas_out);
   for(int i=0;i<n;i++)
      main_q[i]=feas_out[i];
}
modification(main_q,mod_out); // modifying the first
                              // feasible solution
for(int i=0;i<n;i++)
   main_q[i]=mod_out[i];

time1=time(NULL);

duration=0;
while(duration<timemax)  // checking the calculation
                         // time as termination criteria
{
   nei_counter=1;
   kmax=one_enumerator(main_q);
   while(nei_counter<kmax+1)  // neighborhood change
                              // procedure
```

```
{
   rand_q_k(main_q,rand_out,nei_counter);
                  // choosing a random solution in
                  // the "nei_counter"th neighborhood
                  // of the current solution
   if(!feasible(rand_out))  // making feasible
   {
      make_feasible(rand_out,feas_out);
      for(int i=0;i<n;i++)
         rand_out[i]=feas_out[i];
   }
   first_improvement(rand_out,first_imp_out);
                  // finding the first better solution
                  // in the first neighborhood
                  // of the current solution
   if(f(main_q)<f(first_imp_out))
   {
      for(int i=0;i<n;i++)
         main_q[i]=first_imp_out[i];
      nei_counter=1;
   }
   else
      nei_counter++;
}
time2=time(NULL);
```

```
      duration=time2-time1;
   }
   modification(main_q,mod_out); // modifying the current
                                 // solution
   for(int l=0;l<n;l++)
      main_q[l]=mod_out[l];
   clrscr();     /* printing the outputs */
   cout<<"Best Location is:";
   for(int s=0;s<n;s++)
      cout<<main_q[s];
   cout<<"\n";
   cout<<"Best Objective Function Value is:";
   cout<<f(main_q);
   cout<<"\n";
   cout<<"Calculation Time is =";
   cout<<duration;
   cout<<"s";
   getch();
}
void rand_q()  /* generating the first solution randomly */
{
   randomize();
   for(int i=0;i<n;i++)
      q[i]=random(2);
}
```

```
int feasible(int *p)  /* checking the feasiblity of a vector */
{
  int s;
  for(int j=0;j<m;j++)
  {
     s=0;
     for(int i=0;i<n;i++)
        s=s+A[j][i]*p[i];
     if(s>b[j])
        return 0;
  }
  return 1;
}
long int f(int *p)  /* calculating the objective function
                       value of a solution */
{
  int left_energy[n],min;
  dijkstra dij;
  for(int i=0;i<n;i++)
     q_dij[i]=p[i];
  dij.algorithm();
  for(i=0;i<n;i++)  // calculating the remained energy for
                    // the invader
     left_energy[i]=alpha-(pathestimate_i[i]+beta*fac[i]);
```

```
  i=0;
  while(left_energy[i]<0 && i<n)
     i++;
  if(i==n)
     return pathestimate_c[i];
  min=pathestimate_c[i];
  for(int j=i;j<n;j++)
  {
     if(left_energy[j]>=0)
        if(pathestimate_c[j]<min)
           min=pathestimate_c[j];
  }
  return min;
}
void change(int *in,int *out,int t) /* inverting the
                                       "t"th component of
                                       a vector */
{
  for(int i=0;i<n;i++)
     out[i]=in[i];
  out[t]=!out[t];
}
int zero_enumerator(int *p) /* enumerating zero-components
                               of a vector */
{
```

```
  int i=0,j=0;
  for(int l=0;l<n;l++)
     zeros[l]=0;
  while(i<n)
  {
     if(!p[i])
        zeros[j++]=i;
     i++;
  }
  return j;
}
int one_enumerator(int *p) /* enumerating one-components
                              of a vector */
{
  int i=0,j=0;
  for(int l=0;l<n;l++)
     ones[l]=0;
  while(i<n)
  {
     if(p[i])
        ones[j++]=i;
     i++;
  }
  return j;
}
```

```
void make_feasible(int *in,int *out)  /* making an unfeasible
                                         solution feasible */
{
  double  f_out,max=0;
  int j,change_out[n],q_max[n],q_temp[n];
  for(int l=0;l<n;l++)
     q_max[l]=in[l];
  while(feasible(q_max)==0)
  {
     max=0;
     j=one_enumerator(q_max);
     for(int i=0;i<j;i++)
     {
        change(q_max,change_out,ones[i]);
        f_out=f(change_out);
        if(f_out>=max)
        {
           for(l=0;l<n;l++)
              q_temp[l]=change_out[l];
           max=f_out;
        }
     }
     for(l=0;l<n;l++)
        q_max[l]=q_temp[l];
  }
```

```
  for(l=0;l<n;l++)
     out[l]=q_max[l];
}
void modification(int *in,int *out) /* modifying a solution
                                       (adding facilities
                                       if possible) */
{
  int t=0,s,change_out[n],q_mod[n];
  s=zero_enumerator(in);
  for(int k=0;k<n;k++)
     q_mod[k]=in[k];
  while(t<s)
  {
     change(q_mod,change_out,zeros[t]);
     if(feasible(change_out))
     {
        for(int l=0;l<n;l++)
           q_mod[l]=change_out[l];
        t++;
     }
     else
        t++;
  }
  for(int d=0;d<n;d++)
     out[d]=q_mod[d];
```

```
}
void first_improvement(int *in,int *out)
                /* finding the first better solution in
                   the first neighborhood if there's any */
{
  int i=0,q_imp[n],change_out[n];
  long int cur_f;
  for(int l=0;l<n;l++)
     q_imp[l]=in[l];
  cur_f=f(q_imp);
  while(i<n)
  {
     change(q_imp,change_out,i);
     if((feasible(change_out))&&(f(change_out)>cur_f))
        {
           for(int k=0;k<n;k++)
              out[k]=change_out[k];
           break;
        }
     else
           i++;
  }
  for(int j=0;j<n;j++)
     out[j]=q_imp[j];
}
```

```
void rand_q_k(int *in,int *out,int k)
                /* generating a random solution in the
                   "k"th neighbourhood the current vector */
{
  int ran[n];
  int index[n]={0};
  for(int i=0;i<n;i++)
     ran[i]=in[i];
  randomize();
  index[0]=random(n+1);
  int l=1;
  while(l<k)
  {
     randomize();
     index[l]=random(n+1);
     while((inequality_check(index,l))==0)
        index[l]=random(n+1);
     l++;
  }
  for(i=0;i<k;i++)
     ran[index[i]]=!ran[index[i]];
  for(i=0;i<n;i++)
     out[i]=ran[i];
}
```

```
int inequality_check(int *in,int s)
{
  int j=0,ieq[n];
  for(int i=0;i<n;i++)
     ieq[i]=in[i];
  while(j<s)
  {
     if(ieq[s]==ieq[j])
        return 0;
     else
        j++;
  }
  return 1;
}
void read()  /* getting the adjacent matrix of the network,
                the initial energy of the invader and
                the defensive capacity of each facility */
{
   num_of_vertices = n+1;
  cout<<"enter the adjacent matrix of the network:\n";
  for(int i=0;i<num_of_vertices;i++)
  {
     cout<<"\nenter the weights for the row"<<i+1<<":\n";
     for(int j=0;j<num_of_vertices;j++)
     {
```

```
            cin>>graph[i][j];
            while(graph[i][j]<0)
            {
               cout<<"\nu should enter the positive valued
                       weights only\nenter the value again\n";
               cin>>graph[i][j];
            }
         }
      }
      cout<<"\n enter the initial energy of the invader:\n";
      cin>>alpha;
      cout<<"\n enter the defensive capacity of each facility:\n";
      cin>>beta;
      source_i=n;
      source_c=0;
}
```

کد برنامه اصلی EM در C++

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <math.h>

// This file contains the EM class layout.
#include "EM.h"

// This file is used insted of the weak random number
// generator of C++
#include "randnum.h"

// Auxiliary Vector Functions
#include "auxiliary.h"

// These files contain the functions used in
// 'func' method below.
#include "dixonszegofnc.h"

// The RAND is a random number generated by
// the function genrand() in file randnum.h
#define RAND genrand()
```

```
// Constructor
EM::EM ()
{
  GLOBALCOUNT = 0;
  best.objvalue = 1.0e+100;
  best.index = 0;
  best.count = 0;
}

// Destructor
EM::~EM ()
{
  delete (Par.LB);
  delete (Par.UB);

  for (int i = 0; i < Par.M; i++)
    {
      delete (ions[i].coord);
      delete (ions[i].Fi);
    }

  delete best.coord;
}
```

```
// Input
// The lines of the .in file is read and the parameters
// are assigned.
// The explanation of the parameters are given in file EM.h

int
EM::Input (char *infilename)
{
  int i;
  char str[80];
  FILE *fin;

  if ((fin = fopen (infilename, "r")) == NULL)
    {
      printf ("ERROR: INPUT FILE IS NOT ACCESSIBLE!");
    }

  strcpy (str, " ");
  while (strcmp (str, "FunctionNo:"))
    {
      fscanf (fin, "%s", str);
    }
  fscanf (fin, "%s", str);
  Par.FUNCNO = atoi (str);
```

```
while (strcmp (str, "NumberOfIterations:"))
  {
    fscanf (fin, "%s", str);
  }
fscanf (fin, "%s", str);
Par.MAXGEN = atoi (str);

while (strcmp (str, "NumberOfRuns:"))
  {
    fscanf (fin, "%s", str);
  }
fscanf (fin, "%s", str);
Par.NUMOFRUNS = atoi (str);

while (strcmp (str, "NumberOfDimensions:"))
  {
    fscanf (fin, "%s", str);
  }
fscanf (fin, "%s", str);
Par.N = atoi (str);

Par.UB = new long double[Par.N];
Par.LB = new long double[Par.N];
```

```
while (strcmp (str, "LowerBounds:"))
  {
    fscanf (fin, "%s", str);
  }
for (i = 0; i < Par.N; i++)
  {
    fscanf (fin, "%s", str);
    Par.LB[i] = (long double) atof (str);
  }

while (strcmp (str, "UpperBounds:"))
  {
    fscanf (fin, "%s", str);
  }
for (i = 0; i < Par.N; i++)
  {
    fscanf (fin, "%s", str);
    Par.UB[i] = (long double) atof (str);
  }
```

```
while (strcmp (str, "Optimum:"))
  {
    fscanf (fin, "%s", str);
  }
fscanf (fin, "%s", str);
Par.OPTIMUM = (long double) atof (str);

while (strcmp (str, "NumberOfIons:"))
  {
    fscanf (fin, "%s", str);
  }
fscanf (fin, "%s", str);
Par.M = atoi (str);

while (strcmp (str, "LocalCount:"))
  {
    fscanf (fin, "%s", str);
  }
fscanf (fin, "%s", str);
Par.LS = atoi (str);

while (strcmp (str, "Delta:"))
  {
    fscanf (fin, "%s", str);
  }
```

```
  fscanf (fin, "%s", str);
  Par.DELTA = (long double) atof (str);

  while (strcmp (str, "PertPar:"))
    {
      fscanf (fin, "%s", str);
    }
  fscanf (fin, "%s", str);
  Par.PERTPAR = (long double) atof (str);

  Reserve ();

  fclose (fin);
  return (1);
}

// Initialize m points are generated randomly
// from the feasible region.
int
EM::Initialize ()
{
  GLOBALCOUNT = 0;
  best.objvalue = 1.0e+100;
  best.index = 0;
  best.count = 0;
```

```
  for (int i = 0; i < Par.M; i++)
    {
      for (int j = 0; j < Par.N; j++)
  {
    ions[i].coord[j] = Par.LB[j] + (Par.UB[j] - Par.LB[j])*
      RAND;
    ions[i].Fi[j] = 0.0;
    ions[i].Q = 0.0;
  }
      ions[i].objvalue = func (ions[i].coord, i);
    }

  // One point among m points is selected to be the
  // perturbed point.
  p = rand () % Par.M;
  return (1);
}

// CalcF
// The total force on each point is calculated here.
int
EM::CalcF ()
{
  int i, j, k;
```

```
long double totdif = 0.0;
long double temp, dist, farthest = -1.0e+10;
long double *tempvec;
long double *aij;
aij = new long double[Par.N];
tempvec = new long double[Par.N];

// The point which is the farthest from the current best
// point is selected to be the perturbed point. (Proof)
for (i = 0; i < Par.M; i++)
  {
    // The total deviation from the current best objective
    // function value. This is used in the calculation of
    // charges below.
    totdif += ions[i].objvalue - best.objvalue;

    if (i != best.index)
{
  VECSUB (aij, ions[best.index].coord, ions[i].coord,
    Par.N);
  VECNORM (&dist, aij, Par.N);
  if (dist > farthest)
    {
      farthest = dist;
      p = i;
```

```
    }
}
  }

for (i = 0; i < Par.M; i++)
  {
    // The charge of each point is calculated.
    ions[i].Q =
exp (-1.0 * (Par.N * (ions[i].objvalue - best.objvalue)/
  totdif));

    // Total force on each point is initialized.
    for (k = 0; k < Par.N; k++)
ions[i].Fi[k] = 0.0;
  }

for (i = 0; i < Par.M; i++)
  {
    for (j = 0; j < Par.M; j++)
{
        // Attraction
  if ((ions[j].objvalue < ions[i].objvalue) && (i != j))
    {
      VECSUB (aij, ions[j].coord, ions[i].coord, Par.N);
```

```
    VECNORM (&dist, aij, Par.N);
    temp = ions[i].Q * ions[j].Q *
      (1.0 / (pow (dist, 2.0)));
    SCAMUL (temp, aij, Par.N);

          // The total force on the perturbed point
          // is reversed with certain probability.
    if (i != p)
{
  VECADD (tempvec, ions[i].Fi, aij, Par.N);
}
    else
{
  if (RAND < Par.PERTPAR)
    {
      SCAMUL (RAND, aij, Par.N);
      VECADD (tempvec, ions[i].Fi, aij, Par.N);
    }
  else
    {
      SCAMUL (RAND, aij, Par.N);
      VECSUB (tempvec, ions[i].Fi, aij, Par.N);
    }
}
```

```
    VECASGN (ions[i].Fi, tempvec, Par.N);
  }
      // Repulsion
else if ((i != best.index) && (i != j))
  {
    VECSUB (aij, ions[j].coord, ions[i].coord, Par.N);
    VECNORM (&dist, aij, Par.N);

    temp = ions[i].Q * ions[j].Q *
      (1.0 / (pow (dist, 2.0)));
    SCAMUL (temp, aij, Par.N);

          // The total force on the perturbed point
          // is reversed with certain probability.
    if (i != p)
{
  VECSUB (tempvec, ions[i].Fi, aij, Par.N);
}
    else
{
  if (RAND < Par.PERTPAR)
    {
      SCAMUL (RAND, aij, Par.N);
      VECSUB (tempvec, ions[i].Fi, aij, Par.N);
    }
```

```
      else
        {
          SCAMUL (RAND, aij, Par.N);
          VECADD (tempvec, ions[i].Fi, aij, Par.N);
        }
    }

        VECASGN (ions[i].Fi, tempvec, Par.N);
      }

  }      //j

    }        //i

  delete (tempvec);
  delete (aij);

  return (1);
}

// Move
// Each point except the current best point
// is moved in the direction of total force.
```

```
int
EM::Move ()
{
  int i, j;
  long double norm, Tiny;

  for (i = 0; i < Par.M; i++)
    {
      if (i != best.index)
  {
    Tiny = RAND;

          // Normalize the total force on each point
          // to maintain the feasibility.
    VECNORM (&norm, ions[i].Fi, Par.N);
    SCAMUL ((1.0 / norm), ions[i].Fi, Par.N);

    for (j = 0; j < Par.N; j++)
      {
        if (ions[i].Fi[j] >= 0)
    {
      ions[i].coord[j] = ions[i].coord[j] +
        Tiny * ions[i].Fi[j] *
        (Par.UB[j] - ions[i].coord[j]);
    }
```

```
        else
    {
      ions[i].coord[j] = ions[i].coord[j] +
        Tiny * ions[i].Fi[j] *
        (ions[i].coord[j] - Par.LB[j]);
    }
      }      //j
  }
    }        //i

  for (i=0; i < Par.M; i++)
    ions[i].objvalue = func (ions[i].coord, i);
  return (1);
}

// Local
// Simple random local search algorithm.
void
EM::Local ()
{
  ION TempIon;
  TempIon.coord = new long double[Par.N];
```

```
TempIon.Fi = new long double[Par.N];

long double ThresHold, Tiny, Tiny1;
long double *TempVec;
int count;
TempVec = new long double[Par.N];

VECSUB (TempVec, Par.UB, Par.LB, Par.N);
VECMAX (&ThresHold, TempVec, Par.N);

ThresHold *= Par.DELTA;

int i, j;
int FLAG;

// The current best point is assigned to be point i.
// Local search is applied only to
// the current best point.
i = best.index;

for (j = 0; j < Par.N; j++)
  {
    count = 0;
    Tiny1 = RAND;
```

```
    while (count < Par.LS)
{
  ionasgn (&TempIon, &ions[i]);
  Tiny = RAND;
  FLAG = 1;

        // Make a tiny increment in
        // the corresponding dimension.
  if (Tiny1 > 0.5)
    {
      if (TempIon.coord[j] + Tiny * ThresHold < Par.UB[j])
  TempIon.coord[j] += Tiny * ThresHold;
      else
  FLAG = 0;
    }
        // Make a tiny decrement in
        // the corresponding dimension.
  else
    {
      if (TempIon.coord[j] - Tiny * ThresHold > Par.LB[j])
  TempIon.coord[j] -= Tiny * ThresHold;
      else
  FLAG = 0;
    }
```

```
    if (FLAG)
      {
        TempIon.objvalue = func (TempIon.coord, Par.M + 1);

              // Update the best values.
        if (TempIon.objvalue < best.objvalue)
    {
      ionasgn (&ions[i], &TempIon);
      best.objvalue = TempIon.objvalue;
      best.count = GLOBALCOUNT;
    }
        else
    {
      count = Par.LS - 1;
    }
      }
    count++;
  }
    }


  delete (TempVec);
  delete (TempIon.coord);
  delete (TempIon.Fi);
}
```

```
// Output
// Write the best objective function value
// and the best number of iterations to the
// output file, .out.
void
EM::Output (char *outfilename)
{

  FILE *fout;

  if ((fout = fopen (outfilename, "a")) == NULL)
    {
      printf ("ERROR: OUTPUT FILE IS NOT ACCESSIBLE!");
    }

  fprintf (fout, "%f \t %d \n", (float) best.objvalue,
             (int) best.count);

  fclose (fout);
}

// ************
// Auxiliary Functions
// ************
```

```
// Reserve
// This function is used to make necessary
// memory allocations.
int
EM::Reserve ()
{

  ions = new ION[Par.M];

  for (int i = 0; i < Par.M; i++)
    {
      ions[i].coord = new long double[Par.N];
      ions[i].Fi = new long double[Par.N];
    }

  best.coord = new long double[Par.N];

  return (1);
}

// ionasgn
// Assignment of two ions.
void
EM::ionasgn (ION * A, ION * B)
{
```

```
  // A = B
  for (int i = 0; i < Par.N; i++)
    {
      A->coord[i] = B->coord[i];
      A->Fi[i] = B->Fi[i];
    }

  A->objvalue = B->objvalue;
  A->Q = B->Q;

}

void EM::Basic(char *outstr)
{
  int runno = 0;
  int iteration = 0;
  time_t t;

  while (runno < Par.NUMOFRUNS)
    {

      sgenrand ((unsigned long) time (&t) + runno);

      Initialize ();
```

```
// The termination criterion is
// the maximum number of iterations.
iteration = 0;
while (iteration < Par.MAXGEN)
  {
    if (Par.LS != 0)
      Local ();
    CalcF ();
    Move ();
    // This part is used to compare the algorithm
    // with other methods. They use the following
    // termination criterion for the functions with
    // known global optima.
    if (Par.OPTIMUM!= 0.0)
      {
        if ((best.objvalue - Par.OPTIMUM) /
            fabs (Par.OPTIMUM) < 0.0001)
          iteration = Par.MAXGEN - 1;
      }
    else
      {
        if ((best.objvalue - Par.OPTIMUM) < 0.0001)
          iteration =  Par.MAXGEN - 1;
      }
```

```
            iteration++;
          }

        Output (outstr);

        runno++;
      }
}

long double EM::func(long double *x, int particleno)
{
  long double val;

  GLOBALCOUNT++;
  val = Fval (Par.FUNCNO, x, Par.N);

  if (val < best.objvalue && particleno < Par.M)
    {
      best.objvalue = val;
      best.count = GLOBALCOUNT;
      best.index = particleno;
      VECASGN(best.coord, ions[best.index].coord, Par.N);
    }
  return val;
}
```

کد تابع main برنامه EM در C++:

در این تابع اطلاعات مربوط به مسئله بهینه‌سازی از یک فایل ورودی خوانده می‌شود و پس از فراخوانی الگوریتم اصلی EM، نتیجه محاسبات در داخل یک فایل خروجی قرار می‌گیرد.

```
#include <stdio.h>
#include <string.h>
#include <math.h>

#include "EM.h"

// The file that contains the list of input files
// for the functions
#define ALLFILE "dixonszegofnc.in"

int main(int argc, char* argv[])
{
  char instr[50], outstr[50];
  FILE *allin;
  int NUMOFFILES, filecounter;

  EM *EMCPP;
  // A new instance of EM class is created.
  EMCPP = new EM;
```

```
// File is opened.
if ((allin = fopen (ALLFILE, "r")) == NULL)
  {
    printf ("ERROR: INPUT FILE IS NOT ACCESSIBLE!");
  }

fscanf (allin, "%s", instr);
// The first character in ALLFILE shows the number of
//  functions that are going to be solved
NUMOFFILES = atoi (instr);

filecounter = 0;

while (filecounter < NUMOFFILES)
  {
    fscanf (allin, "%s", instr);
    strcpy (outstr, instr);
    // Corresponding .in file for the function.
    strcat (instr, ".in");
    // Corresponding .out file for the function.
    strcat (outstr, ".out");

    EMCPP->Input (instr);
```

```
      // try - catch block is used to detect errors.
      try
        {
          EMCPP->Basic(outstr);
        }
      catch (char *str)
        {
          delete (EMCPP);
          printf ("Exception raised: %s \n", str);
          throw;
        }
      filecounter++;
    } //big while
  delete (EMCPP);
  fclose (allin);
  return 0;
}
```

Abstract

This dissertation considers a fairly new location problem called Defensive Location Problem (DLP). In DLPs, a decision maker locates defensive facilities in order to prevent her/his enemies from reaching an important site, called a core; for example, "a government of a country locates self-defense bases in order to prevent her/his aggressors from reaching the capital of the country." It is assumed that the region where the decision maker locates her/his defensive facilities is represented as a network and the core is a vertex in the network, and that the facility locater and her/his enemy are an upper and a lower level of decision makers, respectively. It is also assumed that all defensive facilities have equal defensive capacities, so DLPs are formulated as bi-level 0-1 programming problems [28]. Then, tow heuristic solving methods are proposed to solve DLP. One of them, which was introduced by Uno and Katagiri [28], is an algorithm based on Tabu Search Method while the other is based on Variable Neighborhood Search Method. A comparison between the efficiency of these tow solving algorithms is done by applying them on several numerical examples of DLPs. Next, we introduce a new development of DLP, named Continuous Defensive Location Problem (CDLP). In CDLPs, we assume that defensive facilities do not have equal defensive capacities necessarily and they can be equipped by the defender to have a capacity which is a real number in a continuous interval. We, therefore, can formulate CDLPs as continuous programming problems. Then we propose a heuristic solving algorithm in order to solve CDLPs, which is based on electromagnetism-like mechanism (EM). The efficiency of the proposed solving method is then shown by applying to some examples of the CDLPs.