بهینه‌سازی ترکیبیاتی بر مبنای توابع ارزیابی

توسط

غلامرضا حفاری

رساله ارائه شده به عنوان بخشی از ملزومات برای دریافت درجه

کارشناسی ارشد هوش‌مصنوعی

زیرنظر

دکتر سعید باقری شورکی

خرداد ۱۳۸۱

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

تهران

قدردانی

در این‌جا لازم است از استاد گرامی آقای دکتر سعید باقری شورکی که در تمام مراحل انجام این پروژه با مساعدت‌ها و راهنمایی‌های بی‌دریغ خود مرا یاری کردند، تشکر کنم. نظرات عمیق او در مورد مباحث هوش مصنوعی و تشویق‌های ایشان طی مراحل تحقیق، همواره انگیزهٔ من در ادامه دادن کار بوده است؛ او با کمال صبوری و متانت (پس از شنیدن صحبتهای من) ایده‌هایی دقیق و موشکافانه می‌داد که گره‌گشای کار در مراحل دشوار بود.

ضمن تشکر از همهٔ اساتیدم در دانشکده مهندسی کامپیوتر دانشگاه صنعتی شریف بخصوص دکتر منصور جم‌زاد رئیس گروه هوش مصنوعی، بر خود لازم می‌دانم که بطور ویژه از پروفسور کارو لوکس و دکتر روزبه توسرکانی سپاسگزاری نمایم. حضور در کلاس درس حسابگری زیستی (ارائه شده توسط پروفسور لوکس) باعث آشنایی من با مباحث پیشرفته و بسیار جالب هوش‌مصنوعی شد و انگیزه تحقیق در زمینه‌های مطرح‌شده در فصل ۲ گشت. همچنین حضور در کلاس درس بهینه‌سازی ترکیبیاتی (ارائه شده توسط دکتر توسرکانی) باعث آشنایی من با پایهٔ ریاضی بهینه‌سازی ترکیبیاتی شد و انگیزه اصلیِ تعریف این تز بشمار می‌رود. حینِ مطالعهٔ برنامه‌ریزی پویا و فرایندهای تصمیم‌گیری مارکوف، پروفسور Tisitsiklis کتاب بسیار عالی خود را (Neuro-Dynamic Programming) برای من ارسال نمود و پروفسور Moore اشکالات من را در مورد STAGE (از طریق مکاتبات شخصی) برطرف کرد؛ از این دو نفر نهایتِ تشکر را دارم.

تحصیل در دانشکده مهندسی کامپیوترِ دانشگاه شریف، سبب آشنایی من با افرادی بود که از دوستان گرانقدر من محسوب می‌شوند؛ از رامین حلاوتی بخاطر بحثهای مفید در مورد مباحث زیستی و سید ابراهیم هاشمیان بخاطر بحثهای سودمند درسی و غیر درسی کمال تشکر را دارم. همچنین از یاشار گنجعلی، حسین سیادت‌زاده و سالم دریساوی قدردانی می‌نمایم، شاید اگر پشتیبانی مالیِ آنها نبود آماده کردن چنین گزارشی هرگز میسر نمی‌شد. از محمد آبام و هم‌کلاسیهایم نیز بخاطر در اختیار گذاشتن امکانات کامپیوتری متشکرم، اغلب نتایج تجربی بدست آمده در فصل ۵، از اجرای برنامه‌ها طی روزهای متوالی روی کامپیوترهای موجود در آزمایشگاه هوش‌مصنوعی و ماشینِ Ada در آزمایشگاه پردازش موازی بدست آمده است.

در پایان از بزرگترین معلمان زندگی‌ام، یعنی مادر و پدرم که والاترین مفهوم زندگی یعنی عشق را به من آموختند، سپاسگزارم و خداوند را به خاطر این هدایای گران‌بهایش شکرگزارم.

بهینه‌سازی ترکیبیاتی بر مبنای توابع ارزیابی

چکیده

تحقیات اخیر، موفقیت استفاده از روشهای یادگیری تقویتی در حل مسائل بهینه‌سازی ترکیبیاتی را نشان داده است. ایدهٔ اصلی این روشها، یادگیری تابع ارزیابی (نزدیک) به بهینه و سپس استفاده از آن جهت پیدا کردن جواب (نزدیک) به بهینه می‌باشد. الگوریتم STAGE —که توسط Boyan و Moore ابداع شد— مهمترین الگوریتم در این زمینه می‌باشد. در این رساله، ما روی مسئله bin-packing که یک مسئله NP-Complete مشهور است، تمرکز خواهیم کرد. ابتدا ساختار سطح هزینه آن را بررسی کرده و ساختار دره بزرگ را برای نحوهٔ پراکندگی بهینه‌های محلی آن تحقیق می‌کنیم، نتیجه علت عملکرد خوب STAGE در حل این مسئله را روشن خواهد ساخت. سپس بر اساس نتایج آزمایشات روی مسئله bin-packing، کارایی استفاده از الگوریتم‌های جستجوی محلی مختلف و ساختارهای یادگیری متفاوت را در STAGE تحلیل خواهیم نمود. علاوه بر این، نتایج آزمایشات بعضی قضایای بهینه‌سازی ترکیبیاتی را تأیید خواهد کرد.

واژه‌های کلیدی: یادگیری تقویتی، بهینه‌سازی ترکیبیاتی، توابع ارزیابی، ساختار سطح هزینه

فهرست مندرجات

لیست اشکال

لیست جداول

فصل ۱

مقدمه

در عصر صنعت، انسانها انجام کار فیزیکی را به ماشینها محوّل کردند و اکنون، در عصر اطلاعات سعی می‌شود که انجام کار ذهنی به کامپیوترها سپرده شود؛ کارهایی از قبیل: کنترل ترافیک، زمانبندی تولید محصولات کارخانه، برنامه‌ریزی برای سرمایه‌گذاری و حتی بازیهایی مثل شطرنج. این‌گونه کارها[۱] جزء مسائل تصمیم‌گیری[۲] می‌باشند:

- کار با اتخاذ یک تصمیم صورت نمی‌گیرد بلکه باید دنباله‌ای از تصمیمات در طول زمان برای انجام آن کار در نظر گرفته شود.
- نتیجه هر تصمیم ممکن است وابسته به متغییرهای کنترلی باشد که خارج از حیطهٔ کامپیوتر قرار دارند؛ و
- هدف نهایی— که با اصطلاحاتی همچون جریان ترافیک، سود اقتصادی و برد در بازی بیان می‌شود— بصورت پیچیده‌ای وابسته به تعامل تصمیمات و نتایج تصادفی آنهاست.

[۱] tasks
[۲] decision making

در چنین مسائلی، سیاستهای تصمیم‌گیری بهینه در حالت کلی ناشناخته هستند و حتی برای خبره‌های انسانی نیز مشکل است که سیاستهای تصمیم‌گیری خوب و معقولانه‌ای را در درون نرم‌افزار تعبیه نمایند. محققان هوش‌مصنوعی راه‌حل زیر را برای برخورد با این مسائل پیشنهاد می‌کنند:
یک الگوریتم تصمیم‌گیری می‌تواند بصورت خودکار سیاستهای مؤثر برای یک کار تصمیم‌گیری را یاد بگیرد، به این طریق که کار را شبیه‌سازی کند و آماری در مورد اینکه کدام تصمیمات منجر به نتایج نهایی خوبی شده‌اند و کدام تصمیمات منجر به نتایج نهایی خوبی نشده‌اند را در خود نگه دارد.

شاخه یادگیری تقویتی[۳]، که این تز مبتنی بر آن است، مفاهیم اصلی و مبانی نظری این متدولوژی را بیان می‌کند.

۱—۱ انگیزه: یادگیری توابع ارزیابی

در هوش‌مصنوعی داده‌ساختار[۴] پایه‌ای برای تصمیم‌گیری در فضاهای حالت[۵] بزرگ، تابع ارزیابی است [۹]. کدام حالت [۶] برای رسیدن به یک حالت نهایی[۷] نزدیک و ارزان باید انتخاب شود؟ تابع ارزیابی، مؤلفه‌های[۸] هر حالت را به یک عدد حقیقی نگاشت می‌کند که نشان‌دهندهٔ خوب بودن حالت می‌باشد. بطور مثال بازی شطرنج را در نظر بگیرید؛ یک تابع ارزیابی کلاسیک برای این مسئله، جمع کردن وزن‌دار موقعیت مهره‌های مختلف در صفحه می‌باشد. انتخاب تابع ارزیابی مناسب مستقیماً روی نتایج بدست آمده از جستجو تأثیر می‌گذارد؛ بطور مثال در دامنه‌های مختلفی نظیر الگوریتمهای

[۳] reinforcement learning
[۴] data structure
[۵] state spaces
[۶] state
[۷] goal state
[۸] features

برنامه‌ریزی و کنترل (A^*)، انجام بازی (alpha_beta) و بهینه‌سازی ترکیبیاتی (تپه‌نوردی [۹] و SA [۱۰]). توابع ارزیابی معمولاً توسط خبره‌های انسانی بدست می‌آیند؛ مثلاً در بعضی برنامه‌های انجام بازی شطرنج، وزنها توسط خبره‌های انسانی بدست آمده‌اند. شطرنج‌باز کامپیوتری Deep Blue (ساخته‌شده توسط شرکت IBM که کاسپارف، قهرمان شطرنج جهان را در سال ۱۹۹۷ شکست داد) از یک تابع ارزیابی استفاده می‌کرد که دارای ۸۰۰۰ پارامتر قابل تنظیم بود. مقادیر اولیه این پارامترها توسط یک روال [۱۱] بطور خودکار بدست آمده بود و سپس با راهنمایی یک خبره انسانی تنظیم شدند. تنظیم پارامترها بطور مشابهی در مسائل ترکیبیاتی نظیر مسئله فروشنده دوره‌گرد (TSP) و کار طراحی مدارات VLSI نیز صورت گرفته‌اند. در چنین مسائلی، فضای جستجو شامل تعداد زیادی جوابهای کاندید معتبر است و تابع هدف [۱۲]— که کیفیت جواب نهایی را مشخص می‌کند— خود می‌تواند در قالب تابع ارزیابی فرایند جستجو را راهنمایی کند. اما اگر تابع هدف دارای تعداد زیادی بهینه محلی [۱۳] باشد و یا در مناطقی مقداری ثابت داشته باشد آنگاه نمی‌تواند به فرایند جستجو کمک مؤثری نماید. بنابراین، برای گرفتن نتایج خوب و بهینه، طراحان الگوریتم وقت زیادی را صرف اضافه کردن جملات خطا و دیگر جملات به تابع هدف می‌نمایند. بدیهی است که روشهای خودکار برای ساخت تابع ارزیابی هم باعث صرفه‌جویی در وقت و تلاش انسانها و هم باعث بالا رفتن کارایی جستجو می‌شود.

[۹] hill climbing
[۱۰] simulated annealing
[۱۱] procedure
[۱۲] objective function
[۱۳] local optimum

۱ — ۲ وعده‌های یادگیری تقویتی

یادگیری تقویتی (RL) بستری مناسب برای یادگیری تابع ارزیابی در یک فرایند تصمیم‌گیری ترتیبی فراهم می‌کند. روشهای استاندارد RL فرض می‌کنند مسئله قابل فرموله‌شدن بصورت فرایند تصمیم‌گیری مارکوف [۱۴] (MDP) است. MDP مدلی برای سیستم‌های دینامیکی قابل کنترل است و از آن بصورت گسترده‌ای در تئوری کنترل، هوش مصنوعی و تحقیق در عملیات استفاده می‌شود [۲۹].

در بخش ۲ مدل MDP را توصیف خواهم کرد. نکته کلیدی اینجاست که برای هر MDP، تابع ارزیابی مخصوصی بنام تابع مقدار بهینه [15] وجود دارد [۹]. این تابع که با $V^*(x)$ نشان داده می‌شود، جایزه [۱۶] درازمدتِ قابل دستیابی از هر حالت x را (با اتخاذ تصمیمات بهینه) پیش‌بینی می‌نماید. V^* تابع ارزیابی ایده‌آل می‌باشد: یک جستجوی حریصانه [۱۷] بر مبنای V^* فقط با در نظر گرفتن یک حرکت بعدی، تصمیمات بهینه در درازمدت را مشخص می‌نماید.

الگوریتمهای محاسبه V^* در صورتی که فضای MDP کوچک باشد (حدود کمتر از $۱۰^۷$ حالت) کاملاً شناخته شده هستند، بنابراین V^* می‌تواند توسط یک جدول جستجو [۱۸] بیان شود. در MDPهای کوچک، اگر مدل گذر [۱۹] (که توزیع حالات بعدی نتیجه شده از اعمال کنش [۲۰] را بیان می‌کند) مشخص باشد، V^* توسط الگوریتمهای کلاسیک مانند برنامه‌ریزی پویا [۲۱] یا برنامه‌ریزی

[۱۴] Markov decision process

[۱۵] optimal value function

[۱۶] reward

[۱۷] greedy

[۱۸] lookup table

[۱۹] transition model

[۲۰] action

[۲۱] dynamic programming

خطی [۲۲] بدست می‌آید [۲۹]. در MDPهای کوچک، وقتی که مدل گذر صریحاً مشخص نباشد باید V^* را از روی مسیرهای [۲۳] بدست آمده از شبیه‌سازی محیط ساخت؛ در این حالت روشهای جدیدِ Q-learning [۳۳] و $TD(\lambda)$ [۳۰] قابل اعمال هستند. این الگوریتمها برنامه‌ریزی پویا را بصورت غیرهم‌زمان [۲۴] و افزایشی [۲۵] بکار می‌برند و تحت شرایط مناسبی می‌توان نشان داد که به V^* همگرا می‌شوند [۶].

اما وضعیت برای فرایندهای تصمیم‌گیری بزرگ متفاوت است. این فرایندها دارای فضای حالت با ابعاد بزرگ هستند، لذا نگهداری V^* در جدول جستجو امکان‌پذیر نمی‌باشد، مسئله‌ای که به نفرین بُعد [۲۶] [۵] معروف است. یک رویکرد برای فرار از این مشکل تخمین V^* توسط یک تخمین‌زننده تابع، مانند شبکه‌عصبی و یا رگرسیون خطی می‌باشد. ترکیب یادگیری تقویتی و تخمین‌زننده تابع به برنامه‌ریزی پویا—عصبی[۲۷] [۶] و یا تخمین تابع مقدار[۲۸] [۱۰] معروف است و موفقیتهای چشمگیری را در دامنه‌هایی همچون کنترل آسانسور [۱۳]، زمانبندی کارها [۳۵] و انجام بازی backgammon [۳۱] بدست آورده است. با این وجود، این پیاده‌سازیها دارای محاسبات سنگینی هستند و برای رسیدن به کارایی بالا نیاز به هزاران و حتی میلیونها مسیر شبیه‌سازی‌شده دارند. همچنین هنگامی که از یک تخمین‌زننده عمومی تابع (مانند شبکه عصبی) بجای جدول جستجو استفاده شود، اثبات همگرایی اغلب روشهای مبتنی بر برنامه‌ریزی پویا امکان‌پذیر نیست. شاید قوی‌ترین شرط همگرایی برای تخمین تابع مقدار این گزاره باشد [۳۲]: برای یک MDP با سیاست تصمیم‌گیری ثابت،

---

[۲۲] linear programming

[۲۳] trajectories

[۲۴] asynchron

[۲۵] incremental

[۲۶] carse of dimensionality

[۲۷] neuro-dynamic programming

[۲۸] value function approximation

الگوریتم $TD(\lambda)$ می‌تواند برای محاسبه دقیق تقریب خطی تابع مقدار بکار رود. اما فرض سیاست تصمیم‌گیری ثابت بسیار محدودکننده است، این قضیه برای STAGE نیز صادق است (الگوریتمی عملی برای بهینه‌سازی سراسری [۲۹] که در بخشهای بعدی معرفی می‌شود).

۱ـ۳ نگاه کلی به ساختار رساله

در این رساله هدف اصلی مطالعه و تحلیل الگوریتمهای بهینه‌سازی مبتنی بر یادگیری تقویتی برای مسائل ترکیبیاتی می‌باشد. ما الگوریتم STAGE را که الگوریتمی موفق در حل مسائل ترکیبیاتی بوده است، بطور تجربی تحلیل خواهیم کرد و ساختار سطح هزینه مسائل ترکیبیاتی را مورد بررسی قرار می‌دهیم. رساله به صورت زیر مرتب شده است:

فصل ۲ مروری کلی بر روشهای بهینه‌سازی نوین ارائه می‌کند. در این فصل روشهای مبتنی بر تکرار برای بهینه‌سازی سراسری مورد بحث قرار می‌گیرند و الگوریتمهای الهام‌گرفته از طبیعت مرور می‌شوند. بویژه من سعی می‌کنم مروری کلی بر SA، الگوریتمهای تکاملی، جمعیت مورچگان [۳۰] و محاسبات مولکولی [۳۱] ارائه دهم.

فصل ۳ تعریفی رسمی از فرایندهای تصمیم‌گیری مارکوف و تخمین تابع مقدار ارائه می‌کند. همچنین در این فصل ساختارهای مختلف تخمین تابع مانند شبکه عصبی، شبکه‌های CMAC و توابع پایه‌ای شعاعی را مرور خواهیم کرد و بویژه ALM را به عنوان روشی فازی برای تخمین تابع معرفی می‌کنیم.

---

[۲۹] global

[۳۰] ant colony

[۳۱] molecular computing

در فصل ۴ ساختار سطح هزینه مسئله bin-packing مورد بررسی قرار می‌گیرد و ساختار دره بزرگ [۳۲] که توسط Boese و دیگران برای شکل سطح هزینه مسائل ترکیبیاتی (بطور ویژه برای TSP و مسئله تقسیم‌بندی گراف) داده شده است، برای مسئله bin-packing تحقیق می‌شود. سپس الگوریتم STAGE مرور خواهد شد و مباحث نظری آن مطرح می‌شوند. این الگوریتم بر مبنای روشهای موجود در یادگیری تقویتی توسط boyan و Moore طراحی شده است و کارایی بسیار خوبی در حل طیف وسیعی از مسائل بهینه‌سازی ترکیبیاتی از خود نشان داده است.

در فصل ۵ بر اساس نتایج تجربی حاصل از حل مسائل ترکیبیاتی سعی می‌کنیم اثر استفاده از الگوریتمهای جستجوی محلی مختلف در STAGE را بررسی کنیم. همچنین اثر استفاده از ساختارهای مختلف یادگیری بر عملکرد STAGE را تحلیل خواهیم کرد. نتایج این فصل بطور عمده از حل مسئله bin-packing بدست آمده‌اند. همچنین در این فصل بطور اصولی ساختار سطح هزینه مسائل بهینه‌سازی مورد بررسی قرار می‌گیرند. این امر به درک علت عملکرد موفقیت‌آمیز STAGE در حل مسائل ترکیبیاتی کمک می‌کند.

فصل ٦ به جمع‌بندی نتایج بدست آمده از رساله و به کارهای جدیدی که در آن انجام شده است، می‌پردازد. همچنین زمینه‌های تحقیقاتی آتی معرفی می‌شوند که بطور عمده مربوط به بکارگیری ALM در شاخه یادگیری تقویتی و کاربرد STAGE در دیگر دامنه‌های بهینه‌سازی می‌باشد.

---

[۳۲] big valley

فصل ۲

مروری کلی بر روشهای نوین بهینه‌سازی

هدف بهینه‌سازی پیدا کردن یک جواب برای یک مسئله است به‌گونه‌ای که این جواب معیاری از خوب بودن را بیشینه یا کمینه نماید. بصورت رسمی با داشتن یک تابع شایستگی [۱] $f : x \to R$ روی یک دامنهٔ بسته X، هدف بدست آوردن $x \in X$ است که از تمام عناصر دیگر دامنه شایسته‌تر باشد که به این x بهینه سراسری گفته می‌شود. برای بسیاری از مسائل، پیدا کردن چنین جوابی در زمان چند جمله‌ای[۲](نسبت به اندازهٔ مسئله ورودی) امکان‌پذیر نیست (چنین مسائلی NP-Hard هستند [۱۹]). در چنین مواردی از یک ابتکار [۳] برای بدست آوردن یک جواب خوب استفاده می‌شود.

بهینه‌سازی سراسری [۴] مربوط به بدست آوردن بهترین جواب ممکن برای یک مسئله است. چون در حالت کلی الگوریتمی برای رسیدن به این هدف وجود ندارد، روشهای بهینه‌سازی مبتنی

[۱] fitness

[۲] polynomial

[۳] heuristic

[۴] global optimization

بر ابتکار به خدمت گرفته می‌شوند که از جمله این روشها می‌توان به SA، الگوریتمهای تکاملی و الگوریتمهای مبتنی بر رفتار مورچگان اشاره کرد.

در بعضی موارد پیدا کردن یک بهینه محلی [5] نیز قابل قبول می‌باشد. روشهای جستجوی محلی [6] کاملاً قابل فهم هستند و پیدا کردن بهینه محلی حتی برای مسائلی که بدست آوردن بهینه سراسری آنها مشکل است می‌تواند بصورت کارایی انجام شود. ترکیب الگوریتمهای جستجوی محلی و سراسری نیز به عنوان یک رویکرد کارا برای بهینه‌سازی سراسری پیشنهاد شده است.

از طرفی پیدا کردن جواب بهینه با استفاده از محاسبات مولکولی نیز مورد توجه قرار گرفته است. در این روش مسئله بصورت زنجیره‌ای از مولکولها، بویژه DNA، مدل می‌شود و سپس حجم وسیعی از این مولکولها برای حل مسئله بکار گرفته می‌شوند که قدرت این روش ناشی از انجام محاسبات بطور موازی در یک محیط زیستی می‌باشد.

الگوریتمهایی که از رفتار مورچگان برای حل مسائل استفاده می‌کنند، از این ایده استفاده می‌کنند که تعاملهای ابتدایی بین عناصر ساده می‌تواند منجر به نمایش یک رفتار هوشمندانه از کل سیستم شود. مورچگان موجوداتی نابینا و ناشنوا هستند و دارای رفتارهایی کاملاً ابتدایی می‌باشند. با این وجود برای مسیریابی به سمت مقصد سعی می‌کنند از مسیری با حداقل مسافت استفاده کنند. با توجه با ساختار کاملاً ساده مورچگان و تعامل ابتدایی آنها با یکدیگر، مشاهدهٔ رفتار هوشمندانه آنها در انتخاب کوتاهترین مسیر، انگیزه طراحی الگوریتمهای بهینه‌سازی بر مبنای این ایده بوده است.

[5] local optimum
[6] local search

۲—۱ بهینه‌سازی سراسری تکراری

الگوریتمهای بهینه‌سازی مبتنی بر تکرار (IGO) [۷] از یک جستجوی تکراری برای بدست آوردن جواب (نزدیک به) بهینه استفاده می‌کنند. خانواده این الگوریتمها در قالب کلی شکل ۲—۱ بیان می‌شود.

Iterative Global Optimization (IGO)

1: **for** $i = 0$ to $+\infty$

2: given the current solution s_i, generate a new random trial solution s'

3: decide whether to set $s_{i+1} = s_i$ or $s_{i+1} = s'$

4: **if** a stopping condition is satisfied

5: **return** the best solution found

شکل ۲—۱: شِمای کلی الگوریتمهای بهینه‌سازی سراسریِ تکراری.

معمولاً s' در خط ۲ با یک روش از پیش تعیین شده— و غالباً تصادفی— از روی s_i بدست می‌آید، بعبارتی $s' \in N(s_i)$ است که $N(s_i)$ نشان دهنده مجموعهٔ جوابهای همسایه s_i می‌باشد. تابع هزینه f همراه با N، یک سطح هزینه را روی توپولوژی همسایگی تعریف می‌کند.

بعضی روشها که از تغییر چارچوب کلی IGO بدست می آیند عبارتند از:

- IGO با نزول حریصانه [۸]
- IGO با تپه‌نوردی [۹]

---

[۷] iterative global optimization

[۸] greedy descend

[۹] hill climbing

• IGO با چند نقطه شروع [۱۰]

در نزول حریصانه جواب کاندید s' تنها هنگامی انتخاب می‌شود که هزینه کمتری نسبت به s_{i+1} داشته باشد. ضعف اساسی این روش این است که در نقاط بهینه محلی گیر می‌افتد. تپه‌نوردی اجازه می‌دهد که در بعضی مواقع حرکتهای غیر بهبود دهنده [۱۱] (یا حرکت به سمت بالا) نیز انجام گیرد، یعنی $f(s_{i+1}) > f(s_i)$ با این امید که در نهایت به جوابی خوب برسد. SA که در بخشهای بعدی بررسی می‌شود، نیز در این دسته الگوریتمها قرار می‌گیرد [۷]. IGO با چند نقطه شروع، الگوریتم بهینه‌سازی را نه فقط از یک نقطه بلکه از نقاط مختلفی شروع می‌کند. انتخابِ نقاط اولیه جستجو می‌تواند تصادفی یا هوشمندانه باشد. در فصلهای بعد خواهیم دید که کیفیت جواب نهایی بدست آمده از اجرای الگوریتمِ جستجوی محلی، (در بعضی مسائل) به نقطه شروع بستگی دارد که این امر منجر به ارائه الگوریتمهایی می‌شود که نقاط اولیه را بطور هوشمند برمی‌گزینند (در واقع این ایده، ایدهٔ اصلی الگوریتم STAGE است).

۲—۲ الگوریتم فرا ابتکاری بر مبنای رفتار مورچگان

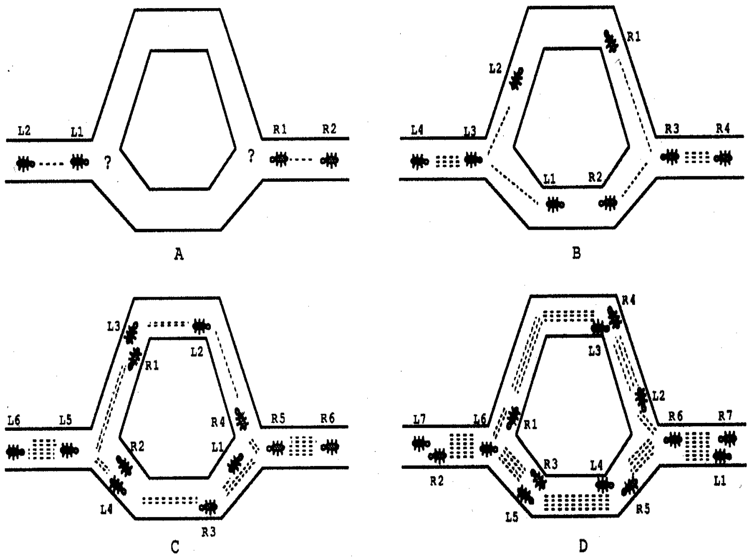
الگوریتم فرا ابتکاری مبتنی بر رفتار مورچگان [۱۲](AGO) در اوایل دهه نود میلادی توسط دوریگو، مانیزو و کلرنی معرفی گردید ([۱۶] و [۱۸]). این الگوریتم از رفتار مورچگان در طبیعت الهام گرفته شده است. کاربرد این الگوریتم در حل مسائل بهینه‌سازی ترکیبیاتی و نتایج حاصل از آن نشان دهنده کارایی بالای این الگوریتم است.

---

[۱۰] multi start

[۱۱] dissimproving

[۱۲] ant colony optimization



شکل ۲—۲: شکلهای بالا نشان‌دهندهٔ چگونگی پیدا کردنِ کوتاهترین مسیر توسط مورچه‌های واقعی می‌باشد. a) مورچه‌ها به یک نقطه تصمیم‌گیری می‌رسند. b) بعضی از مورچه‌ها مسیر بالا و بعضی مسیر پایین را انتخاب می‌کنند، این انتخاب تصادفی است. c) چون مورچه‌ها با سرعتی تقریباً ثابت حرکت می‌کنند، مورچه‌هایی که مسیر پایینی (کوتاهتر) را انتخاب می‌کنند زودتر از مورچه‌هایی که مسیر بالا (بلندتر) را انتخاب می‌کنند به نقطه تصمیم‌گیریِ دوم می‌رسند. d) فرومون با نرخ بیشتری بر روی مسیر کوتاهتر انباشته می‌شود. تعداد خطوط بریده بریده تقریباً متناسب با مقدار فرومون انباشته‌شده می‌باشد.

مورچه‌ها با اینکه فاقد قدرت بینایی‌اند، می‌توانند کوتاهترین مسیر از منبع تغذیه تا لانه خویش را با استفاده از مواد شیمیایی که در هنگام حرکت از خود بر جای می‌گذارند (فرومون [۱۳])، پیدا کنند [۱۷]. مورچه‌ها در هنگام حرکت با بر جای گذاشتن فرومون، فرومونهای باقیمانده از سایر مورچه‌ها را (بصورت تصادفی) دنبال می‌نمایند. از نظر مورچه‌ها مسیری مطلوب‌تر است که مقدار فرومون بیشتری داشته باشد. طریقه یافتن کوتاهترین مسیر با استفاده از فرومون در شکل ۲—۲ نشان داده شده است.

حالت A را در شکل ۲—۲ در نظر بگیرید. مورچه‌ها به یک دوراهی رسیده‌اند و مجبورند که تصمیم بگیرند به سمت بالا حرکت کنند یا به پایین. در این لحظه هیچ پیش زمینه‌ای در مورد بهترین

[۱۳] pheromone

انتخاب وجود ندارد و مورچه‌ها مسیر حرکت خود را بصورت کاملاً تصادفی انتخاب می‌نمایند. می‌توان انتظار داشت که بطور متوسط، نیمی از مورچه‌ها مسیر بالا و نیمی دیگر مسیر پایین را انتخاب کنند. این فرایند هم برای مورچه‌هایی که از چپ به راست در حرکتند (و با حرف L نشان داده شده‌اند) و هم برای مورچه‌هایی که از سمت راست به چپ در حرکتند (و با حرف R نشان داده شده‌اند) صدق می‌کند. با فرض مساوی بودن سرعت حرکت مورچه‌ها، وقایعی که پس از حالت A روی می‌دهد در قسمتهای B و C شکل ۲—۲ نشان داده شده است. تعداد خطهای تیره بطور تقریبی متناسب با مقدار فرومونی است که مورچه‌ها در هنگام حرکت بر روی زمین باقی گذاشته‌اند. از آنجا که مسیر پایینی کوتاهتر از مسیر بالایی بوده است بنابراین تعداد بیشتری مورچه می‌توانند این مسیر را در واحد زمان طی کنند که این امر موجب انباشته شدن سریع فرومون در این مسیر می‌گردد. بتدریج اختلاف فرومون دو مسیر زیاد می‌شود و پس از مدتی اختلاف فرومون در دو مسیر به اندازه کافی بزرگ می‌شود تا بر تصمیم مورچه‌های جدید در انتخاب مسیر، تأثیرگذار باشد. این مطلب در قسمت D شکل ۲—۲ نشان داده شده است. از حالا به بعد مورچه‌ها ترجیح می‌دهند تا مسیر پایین را انتخاب نمایند. این فرایند با یک بازخور [۱۴] مثبت ادامه می‌یابد، یعنی اینکه افزایش انتخاب مسیر موجب تجمع فرومون بیشتری در این مسیر و تجمع فرومون بیشتر باعث افزایش انتخاب مسیر می‌گردد. پس از مدتی تمام مورچه‌ها مسیر کوتاهتر را برای ادامه حرکت خود انتخاب می‌کنند.

اولین الگوریتم بهینه‌سازی بر مبنای رفتار مورچه‌ها بر اساس همین ایده ساده ابداع گردید. این الگوریتم که سیستم مورچه‌ها [۱۵] (AS) نام داشت نخستین بار در حل مسائلی مانند TSP[۱۶] و QAP[۱۷]

---

[۱۴] feedback

[۱۵] ant system

[۱۶] travelling salesman problem

[۱۷] quadratic assignment problem

بکار گرفته شد. در الگوریتم AS از تعدادی مورچه مصنوعی[۱۸] استفاده می‌شود که این مورچه‌ها از طریق مبادله اطلاعات با یکدیگر در ساختن جواب همکاری می‌کنند. مورچه‌ها در طول یالهای گراف مسئله از گرهی به سمت گره دیگر حرکت می‌کنند و با باقی گذاشتن فرومون بر روی یالها به مبادله اطلاعات با یکدیگر می‌پردازند.

۲—۳ الگوریتم Simulated Annealing

همانگونه که اشاره شد نقطه ضعف اصلی الگوریتم نزول حریصانه، گیر افتادن در نقاط بهینه محلی می‌باشد. رویکردهای جایگزین اجازه می‌دهند که در بعضی مواقع حرکت به سمت بالا یا حرکتهای غیر بهبود دهنده نیز انجام گیرد، یعنی $f(s_{i+1}) > f(s_i)$ با این هدف که از نقاط کمینه محلی فرار کنیم. شاید مهمترین و معروفترین تکنیک بر این اساس، SA باشد که اولین بار توسط Kirkpatrick و دیگران [۲۶] در سال ۱۹۸۳ ابداع شد.

SA در خط ۳ شکل ۲—۱ این‌گونه عمل می‌کند: اگر $f(s') \le f(s_i)$ آنگاه $s_{i+1} = s'$ و در غیر اینصورت اگر $f(s') > f(s_i)$ باشد آنگاه با احتمال $e^{\frac{-[f(s')-f(s_i)]}{T_i}}$، s_{i+1} برابر با s' قرار می‌گیرد. در اینجا $T_i > 0$ مقدار دما در تکرار iام الگوریتم می‌باشد و دنباله‌ای از دماها، برنامه دمایی [۱۹] نامیده می‌شود که روش بهینه‌سازی را کنترل می‌نماید. SA جوابهای واقعاً خوبی در عمل داده است و از لحاظ تئوری نیز ثابت شده است که جواب بهینه سراسری را در زمان بینهایت می‌دهد به این شرط که دارای برنامه دمایی صحیح باشد و از هر حالتی بتوان به حالت دیگری از فضای جواب با اعمال دنباله‌ای از کنشها رسید. با این وجود این روش برای اینکه بتواند با سایر روشهای بهینه‌سازی رقابت

[۱۸] artificial ant

[۱۹] temperatute schedule

SA Algorithm Template

0: $s_0 \leftarrow$ random solution in S

1: **For** $i = 0$ to $M - 1$

2: choose $s' \leftarrow a$ random element from $N(s_i)$

3: **if** $f(s') < f(s_i)$

4: $s_{i+1} \leftarrow s'$

5: **else**

6: $s_{i+1} \leftarrow s'$ with probability $e^{\frac{-[f(s')-f(s_i)]}{T_i}}$

7: otherwise $s_{i+1} \leftarrow s_i$

8: **return** s_M

8a. **retrun** s_i, $0 \leq i \leq M$, such that $f(s_i)$ is minimum

شکل ۲—۳: شمای کلی الگوریتم simulated annealing با حداکثر M گام (زمانی).

کند نیاز به گرفتن زمان زیادی از CPU دارد.

الگوریتم SA با جزئیات بیشتر در شکل ۲—۳ نشان داده شده است. یک تکرار از خط ۱ تا ۷ یک گام را تشکیل می‌دهد و یک اجرا از الگوریتم شامل دنباله‌ای از گام‌هاست. جواب کاندید s' در هر مرحله بطور تصادفی بر اساس اختلاف $f(s') - f(s_i)$ و دمای T_i پذیرفته می‌شود. هنگامی که $T_i = \infty$ باشد تمام جوابهای کاندید s' پذیرفته می‌شوند و هنگامی که $T_i = 0$ باشد فقط جوابهایی پذیرفته می‌شوند که بهبود حاصل نمایند، یعنی $f(s') \leq f(s_i)$ باشد. بنابراین قالب نشان داده شده در شکل ۲—۳ کلی است، به این معنا که در خود هم الگوریتم نزول حریصانه ($T_i = 0$) و هم الگوریتم گام تصادفی [۲۰] ($T_i = \infty$) را در بر می‌گیرد.

[۲۰] random walk

معمولاً مهمترین پارامترهای پیاده‌سازی الگوریتم SA ساختار همسایگی N و برنامه دمایی $T_0, T_1, \ldots, T_{M-1}$ می‌باشد. این دو پارامتر با یکدیگر مصالحه [21] بین کاویدن [22] فضای جواب و تقید [23] به ادامه دادن یک مسیر جستجو را مشخص می‌کنند. وجود همسایه‌های بیشتر باعث کاوش بیشتر فضای جواب در یک گام می‌شود و در نتیجه باعث کاهش تعداد گامهای لازم برای رسیدن به جواب بهینه می‌گردد. از طرفی وجود همسایه‌های بیشمار موجب می‌گردد هنگامی که دما پایین است SA برای مدت زمان تقریباً طولانی‌تری در یک جواب باقی بماند. به طریق مشابه دمای بالاتر باعث می‌شود فضای زیادی از جوابها کاوش شود چونکه اغلبِ جوابهای کاندید پذیرفته می‌شوند اما دمای پایین از فاصله گرفتن از جوابهای کم‌هزینه جلوگیری می‌نماید.

قسمتی از جذابیت SA، ناشی از شباهت آن به ترمودینامیک است؛ جواب بهینه باید در دمای $T = 0$ بدست آید مشروط بر اینکه دما به اندازه کافی آهسته کم شود. در تمام اثباتهایی که برای بدست آوردن جواب بهینه در بی‌نهایت ذکر شده‌اند، فرض شده است که برنامه دما، بصورت یکنوا و غیر نزولی (یا سرد شونده) به سمت دمایی نزدیک به صفر پیش می‌رود (لذا در انتهای اجرا، فقط حرکتهایی که در جهت بهبود جواب هستند پذیرفته می‌شوند) [۷].

۲–۴ الگوریتمهای بهینه‌سازی بر مبنای تکامل

الگوریتمهای تکاملی [24] (EA) به عنوان یک روش بهینه‌سازی سراسری مورد توجه قرار گرفته‌اند. این دسته از الگوریتمها با نگه داشتن یک جمعیت از genotype نمونه‌ها، یک نمونه‌برداری اولیه بصورت

[21] trade off
[22] exploration
[23] exploitation
[24] evolutionary algorithms

یکنواخت از کل فضای جستجو انجام می‌دهند. نسلهای بعدی از نسلهای قبلی با اعمال عملگرهای ژنتیکیِ انتخاب [۲۵] (بردن جمعیت به سمت نواحی از جمعیت فعلی که امید بیشتری برای یافتن جواب بهینه دارند)، جهش [۲۶] و ترکیب مجدد [۲۷] بدست می آیند [۲۷].

بصورت دقیق‌تر، جوابهای امکان‌پذیر [۲۸] یک مسئله بهینه‌سازی توسط مجموعه‌ای از پارامترها (که genotype نامیده می‌شود) نشان داده می‌شوند. این پارامترها ممکن است دارای مقادیر حقیقی یا گسسته باشند که این وابسته به پیوسته یا گسسته بودن متغیرهای مسئله‌ای است که باید حل شود. در هر نسل، هر genotype ارزیابی می‌شود و به آن یک مقدار شایستگی [۲۹] اختصاص داده می‌شود. بر اساس خوب بودن genotypeها (که بر اساس شایستگی بیان می‌شود) بعضی از آنها برای تولید دوباره انتخاب می‌شوند. genotypeهای بهتر شانس بیشتری برای تولید مثل دارند، لذا ژن آنها از تکثّر بیشتری در نسل بعدی برخوردار است. به این صورت، بر اساس اطلاعات آماری که از نسلهای قبلی بدست آمده است، نسلهای بعدی به سمت نواحی امیدوارکننده‌تر برده می‌شوند.

یک نسل جدید، با اعمال عملگرهای ژنتیکی بر genotypeهای انتخاب شده از نسل قبل بوجود می آید. در الگوریتم ژنتیک، عملگرهای ژنتیکی عبارتند از: جهش، کپی‌برداری و ترکیب مجدد. ترکیب مجدد معمولاً عبارتست از انتخاب دو والد و سپس کپی کردن بعضی پارامترها از والد اول و کپی کردن بقیه پارامترها از والد دوم. جهش باتغییرات تصادفی کوچک روی genotypeهای بدست آمده صورت می‌گیرد. قالب کلی الگوریتم در شکل ۲—۴ نشان داده شده است.

[۲۵] selection
[۲۶] mutation
[۲۷] recombination
[۲۸] feasible solutions
[۲۹] fitness

Standard Evolutionary Algorithm

```
randomly generate an initial population of size M
repeat
    evaluate each genotype in the population
    select genotypes for reproduction, based on fitness
    perform recombination and mutation on selected genotypes
    replace part or all of population with new genotypes
until good enough solution found, or out of time
```

شکل ۲—۴: الگوریتم تکاملی پایه.

۲—۴—۱ steady state **در برابر** generational

توصیف شکل ۲—۴ سؤالات زیادی را بدون جواب می‌گذارد: چه تعداد از افراد جدید باید به یکباره تولید شوند و چگونه عملیات جایگزینی افراد قدیم با افراد جدید صورت می‌گیرد؟ در اغلب موارد تمام افراد جمعیت به یکباره جایگزین می‌شوند و بنابراین به تعداد P فرد جدید در هر نسل تولید می‌شود. برای تأکید بر اینکه جمعیت در نسلهای مختلف متمایز است الگوریتم تکاملی که از این شِما استفاده می‌کند generational EA نامیده می‌شود. یک صورت رایجِ دیگر از این الگوریتم، نگهداری بهترین جواب یافت شده در نسل قدیم و انتقال آن به نسل جدید می‌باشد. به این صورت فرایند جستجو هیچگاه بهترین جواب دیده شده را از دست نمی‌دهد.

در برابر روش generational، EAهای حالت پایدار فقط یک فرد را در هر نسل جایگزین می‌کنند. فردی که باید جایگزین شود معمولاً بدترین فرد نسل است. EA حالت پایدار، تا زمانی که جواب بهتری بدست نیاورده است، جوابهای خوب را جایگزین نمی‌کند و لذا از خود پایداری خوبی

Evolutionary Algorithm with Local Search

0: randomly generate an initial population of size M

1: **repeat**

2: evaluate each genotype (or locally optimized genotype)

3: select genotypes for reproduction, based on fitness

4: perform recombination and mutation on selected genotypes

5: perform local search on members of the population

6: **until** good enough solution found, or out of time

شکل ۲–۵: شِمای کلیِ الگوریتم تکاملی + جستجوی محلی.

نشان می‌دهد.

۲–۴–۲ الگوریتمهای تکاملی + جستجوی محلی

یک گسترش مرسوم EA، استفاده از الگوریتمهای جستجوی محلی در آنهاست (EA + LS). جستجو کننده محلی، الگوریتمی شبیه نزول حریصانه و یا گام تصادفی است که فقط برای یافتن بهینه محلی تلاش می‌کند؛ این روشها هیچ‌گونه تلاشی برای یافتن بهینه سراسری انجام نمی‌دهند و این مسئولیت را بر دوش جمعیت و عملگرهای ژنتیکی می‌گذارند. قالب کلی این الگوریتمها در شکل ۲–۵ آمده است.

جستجوی محلی نقشی مکمل در جستجوی سراسری EA ایفا می‌کند. این الگوریتم قسمت زیادی از فضای جستجو را کاوش نمی‌کند زیرا که بستر جذب [۳۰] فضای جستجو در مقایسه با جوابهای

[۳۰] basin of attraction

موجود در این فضا بسیار کمتر است. یک فایدهٔ استفاده از جستجوی محلی، بدست آوردن آمارهایی بهتر از کل فضا می‌باشد.

۲—۴—۳ تکامل داروینی و لامارکی

الگوریتمی که در شکل ۲—۵ مطرح شد، مشخص نکرد که روی نتیجه جستجوی محلی چه کاری صورت می‌گیرد (هر چند که در هر صورت از مقدار شایستگی جواب بدست آمده استفاده می‌شود). به دو طریق می‌توان با جواب بدست آمده از الگوریتم جستجوی محلی برخورد کرد. راه اول، جایگزینی نقطه شروع جستجوی محلی با جواب بدست آمده از آن می‌باشد؛ به این معنی که جواب بدست آمده روی genotype اولیه کپی می‌شود. این استراتژی، تکامل لامارکی نام دارد چون اجازه می‌دهد خصوصیات یاد گرفته شده از محیط، روی خصوصیات ژنتیکی تأثیر بگذارد. راه دوم استفاده از تکامل داروینی است که در آن جواب نهایی بدست آمده از الگوریتم جستجوی محلی از بین می‌رود و فقط از مقدار شایستگی آن استفاده می‌شود؛ بعبارتی خصوصیات یاد گرفته شده از محیط، روی ژنها تأثیر نمی‌گذارد.

۲—۴—۴ بهینه‌سازی بر مبنای Co-Evolution

الگوریتم ژنتیک Holland، دسته‌ای از الگوریتمهای ابتکاری برای حل مسائل بهینه‌سازی را مشخص می‌کند [۲۲]؛ با این ویژگی جالب که این دسته از الگوریتمها بر مبنای مدلی زیستی و طبیعی بدست آمده‌اند. یک مدل جالبِ دیگر تکامل در طبیعت، Co-Evolution می‌باشد. Co-Evolution یک دینامیک مهم در طبیعت است که بر مبنای آن گونه‌های زیستی مختلف، با تکامل خود باعث کمک به

تکامل در دیگر گونه‌ها می‌شوند. از این ایده می‌توان برای بدست آوردن الگوریتمهای ابتکاری که باعث تکامل یکدیگر می‌شوند، استفاده نمود. اگر الگوریتمهای ابتکاری مختلف را به عنوان گونه‌های زیستی در یک اکولوژی در نظر بگیریم، آنگاه مفهوم شایستگی در فرایند تکامل، بصورت قدرت گونه در حل نمونه‌های مسئله نمود پیدا می‌کند، بعبارتی تنوع در فضای نمونه‌های مسئله باعث رشد الگوریتمهای ابتکاری می‌شود که در حل زیرمجموعه‌ای از نمونه‌های مسئله بسیار قدرتمند عمل می‌کنند. Belew و دیگران از این تکنیک برای حل مسئله تقسیم‌بندی گراف استفاده کرده‌اند و نتایج جالبی را گزارش داده‌اند [۴]. آنها از الگوریتم تکاملی برای حل مسئله تقسیم‌بندی گراف استفاده کردند اما نکتهٔ اصلی اینجا بود که خود این الگوریتم دارای پارامترهای زیادی است که باید تنظیم می‌شد. آنها از Co-Evolution برای بدست آوردن این پارامترها استفاده کردند، به این طریق که الگوریتمی که دسته‌ای از مسائل را حل می‌کرد دارای شایستگی بیشتری بود و مسئله‌ای که الگوریتمهای کمتری می‌توانستند آن را حل کنند نیز دارای شایستگی بالا بود. بعد از گذشتِ مدتی از اجرای این روش، مسائل به زیر مجموعه‌هایی تقسیم‌بندی شدند و برای هر یک از این زیرمجموعه‌ها، الگوریتمی تکاملی بدست آمد که فوق‌العاده خوب عمل می‌کرد.

۲—۵ محاسبات مولکولی

علم محاسبات زیستی، از روشهای محاسباتی که در علم کامپیوتر وجود دارد، استفاده می‌کند تا اطلاعات زیستی بدست آمده از طریق تکنولوژی مدرنِ مولکولی را تحلیل کند. محاسبات مولکولی (بویژه DNA) این ارتباط را برعکس نموده است؛ بجای محاسبات درباره رشته‌های DNA (همانند آنچه که در محاسبات زیستی اتفاق می‌افتاد)، محاسبات DNA از رشته‌های DNA برای حل مسائل

محاسباتی بهره می‌بَرد [۱۵].

تاسال ۱۹۹۴، Adelman به دنبال راهی بود که از قابلیتهای طبیعی DNA برای ذخیره داده و محاسبه استفاده کند. کامپیوترهای متعارف از مدارات الکترونیکی ریز بر روی چیپهای سیلیکونی برای نگهداری اطلاعات (بصورت ۰ها و ۱ها) استفاده می‌کنند، در حالی که کامپیوترهای DNA اطلاعات را بصورت الگویی از مولکولها که در امتداد یک رشته قرار گرفته‌اند (A،C،T،G) [۳۱]، نشان می‌دهند؛ بطوری‌که واکنشهای زیست—شیمیایی (به کمک آنزیمها) جایگزین الکتریسیته و نرم‌افزار می‌شوند [۲]. Adelman سعی کرد که مسئله دور هامیلتونی را در گراف [۳۲] (که یک مسئله NP-Complete است) با محاسبات DNA حل کند. یک الگوریتم برای حل مسئله دور هامیلتونی در شکل ۲—۶ داده شده است.

General Algorithm for the Hamiltonian Path Problem

1: generate random paths through the graph.

2: keep only those paths that start and end with the correct vertices.

3: if the graph has n vertices, only keep paths with n vertices.

4: keep only the paths that enter all cities at least once.

5: remaining paths are solutions.

شکل ۲—۶: یک الگوریتم برای حل مسئله دور هامیلتونی.

Adelman از یک گراف با ۷ رأس و ۱۳ یال استفاده کرد. هر رأس با یک رشته DNA با طول ۲۰ مولکول (بصورت یکتا) مشخص شد [۱]. یالها نیز با رشته‌های DNA با طول ۲۰ نمایش داده شدند اما طوری طراحی شدند که نیمه اول آنها (۱۰ مولکول اول) مکمل نیمه دوم رأس مبدأ و نیمه

[۳۱] لازم به ذکر است که این چهار حرف، مخفف نام چهار باز پایه می‌باشند که بازهای A و T مکمل هم هستند و تمایل دارند که با هم پیوند تشکیل دهند. بطور مشابه بازهای C و G نیز مکمل یکدیگر بوده و تمایل دارند که با هم پیوند تشکیل بدهند.

[۳۲] پیدا کردن دوری در یک گراف که از همه گره‌ها فقط یک بار بگذرد و دارای حداقل وزن باشد

دوم آنها (۱۰ مولکول دوم) مکمل نیمه اول رأس مقصد باشد. تعداد زیادی کپی از این رشته‌های DNA یالی و رأسی ساخته شد تا حالات ممکن مختلف برای ترکیب یالها و رأسها با هم اتفاق بیفتد؛ سپس روشهای زیست—مولکولی برای جدا کردن رشته‌هایی از مولکولها (شامل DNAهای رأسها و یالها) که خواص جواب معتبر را داشتند و دارای هزینه کمتری بودند، انتخاب شدند. در شکل ۲—۷ روش کلی محاسبات DNA برای حل مسئله نشان داده شده است.

Typical DNA Computing Methodology

1: encoding:
map problem instance onto a set of biological molecules and molecular biology protocols.

2: molecular operations:
let molecules react to form all potential solutions.

3: extraction/detection:
use molecular biology protocols to extract result in molecular form.

شکل ۲—۷: فرایند حل مسئله با استفاده از محاسبات DNA.

محاسبات DNA دارای مزیتهایی نسبت به محاسبات در کامپیوترهای متعارف می‌باشد: سرعت، انجام محاسبات بطور موازی و حافظه بالا. آزمایشات Adelman نقطه عطفی در انجام محاسبات محسوب می‌شود؛ او نشان داد که ایدهٔ اولیه درنظرگیری تمام فضای جستجو، حتی با موازی‌سازی جستجو (توسط فراهم کردن تعداد زیادی زنجیره DNA و بدست آوردن تمام ترکیبهای ممکن برای جواب بصورت موازی) هنوز برای مسائل NP-Complete عملی نیست چون با بالا رفتن اندازه مسئله، بطور نمایی تعداد زنجیره‌های DNA لازم نیز زیاد می شود. محاسبات DNA برای سرعت بخشیدن به حل مسائل tractable، با ارزش به نظر می‌رسد، همچنین برای پیاده‌سازی الگوریتمهایی که نیاز به در نظر گرفتن بخشی از فضای جوابها را دارند. از طرف دیگر اهمیت این روش می‌تواند در انجام محاسبات توسط مدل جدیدی از محاسبه می‌باشد.

۲–۶ بحث

در این فصل سعی کردیم مروری کلی بر روشهای نوین بهینه‌سازی که اغلب الهام گرفته از طبیعت می‌باشند، ارائه دهیم. ابتدا ساختار کلی این الگوریتمها را تحت عنوان بهینه‌سازی سراسری مبتنی بر تکرار (IGO) مد نظر قرار دادیم. سپس الگوریتم SA را مطرح کردیم و دیدیم که الگوریتمهای جستجوی محلی نزول حریصانه و گام تصادفی می‌توانند به عنوان حالت خاصی از آن محسوب شوند. سپس به بیان الگوریتم فراابتکاری بهینه‌سازی بر مبنای رفتار مورچه‌ها پرداختیم و دیدیم که چگونه تعامل ساده بین عناصر ساده می‌تواند منجر به بروز رفتار هوشمندانه از کل سیستم شود (در اینجا ظهور رفتار هوشمندانه از کل سیستم در بهینه‌سازی تابع تحقق پیدا می‌کند). الگوریتمهای بهینه‌سازی مبتنی بر تکامل و Co-Evolution نیز دسته‌ای بزرگ از الگوریتمهای بهینه‌سازی (چه در حالت پیوسته و چه در حالت گسسته) را در بر می‌گیرند. اهمیت این دسته از الگوریتمها در این است که سعی می‌کنند از طبیعت تقلید کنند و در عمل نیز نشان داده شده است که نتایج خوبی را از خود نشان داده‌اند. محاسبات DNA، عمل جستجو برای بدست آوردن جواب را در یک محیط زیستی انجام می‌دهد و بخاطر موازی‌سازی در انجام محاسبات، به عنوان یک روش تقریباً سریع در جستجو مطرح شده است.

فصل ۳

یادگیری تقویتی و تخمین تابع مقدار

در این فصل به معرفی چارچوب فرایند تصمیم‌گیری مارکوف می‌پردازیم. انگیزه ما در مطرح کردن چنین بحثی، معرفی یادگیری تقویتی و بحث تخمین تابع مقدار می‌باشد. در فصلهای آتی خواهیم دید که چگونه جستجو در فضای جوابهای یک مسئله بهینه‌سازی ترکیبیاتی را می‌توان در قالب مبحث یادگیری تقویتی مطرح نمود، سپس الگوریتمی برای بهینه‌سازی بر مبنای این ایده معرفی می‌شود. در ادامه این فصل به معرفی چند تخمین‌زننده مناسب برای توابع که بکارگیری آنها در یادگیری تقویتی منجر به بدست آمدن نتایج خوبی شده است، می‌پردازیم. در انتها نیز روش یادگیری فعال [۱] (ALM) را به عنوان یک روش یادگیری و تخمین‌زننده فازی معرفی می‌کنیم، با این امید که ترکیبب آن با یادگیری تقویتی نتایج خوبی را در بر داشته باشد.

---

[۱] active learning method

۳—۱ فرایند تصمیم‌گیری مارکوف

فرض کنید فضای جستجو بصورت یک فرایند تصمیم‌گیری مارکوف [۲] (MDP) باشد که بصورت زیر تعریف شده است:

- مجموعه‌ای از حالات X، شامل مجموعه‌ای از حالات اولیه $S \subset X$
- مجموعه‌ای محدود از کنشها A
- تابع پاداش $R : X \times A \rightarrow R$ که $R(x,a)$ پاداش بلافصل حاصل از انجام کنش a در حالت x است.
- مدل گذر $P : X \times X \times A \rightarrow R$ که $P(x'|x,a)$ احتمال رسیدن به حالت x' از انجام کنش a در حالت x را می‌دهد [۹].

تصمیمات در یک MDP توسط یک سیاست نشان داده می‌شود: $\pi : X \rightarrow A$ که هر حالت را به یک کنش نگاشت می‌کند (یا بطور کلی‌تر به یک توزیع احتمالی روی کنشها نگاشت می‌کند). فرض می‌کنیم سیاست پایدار [۳] باشد، یعنی در طول مدت شبیه‌سازی تغییر نکند. برای هر سیاست پایدار π، تابع مقدار سیاست [۴] $V^\pi(x)$ بصورت امید ریاضی [۵] پاداش دراز مدتِ انباشته شده با شروع از حالت x و دنبال کردن سیاست π تعریف می‌شود:

$$V^\pi(x) = E\{\sum_{t=0}^{\infty} \gamma^t R(x_t, \pi(x_t)) | x_0 = x\} \qquad (۱)$$

---

[۲] Markov decision process

[۳] stationary

[۴] policy value function

[۵] expected

در اینجا γ ضریب تنزیل[۶] می‌باشد که نظر ما را در مورد پاداش درازمدت و پاداش کوتاه‌مدت بیان می‌کند. با فرض محدود بودن مقدار پاداش، V^π برای هر مقدار $\gamma < 1$ تعریف شده است؛ برای حالت $\gamma = 1$، V^π هنوز تعریف شده است مشروط بر اینکه هر مسیر [۷] خاتمه‌پذیر باشد[۸]. بعبارتی هر مسیر به یک حالت جذب کننده [۹] برسد که تمام پاداشهای بعد از آن ۰ باشد. هر MDP با $\gamma < 1$ را می‌توان به یک MDP معادل با $\gamma = 1$ تبدیل نمود، لذا در این تز فرض می‌کنیم $\gamma = 1$ باشد؛ یعنی به پاداشهای درازمدت و کوتاه‌مدت اهمیت یکسانی می‌دهیم.

تابع مقدار سیاست V^π در این سیستم خطی از معادلاتِ Bellman (برای پیش‌بینی) صدق می‌کند:

$$\forall x, V^\pi(x) = R(x, \pi(x)) + \gamma \sum_{x' \in X} P(x'|x, \pi(x)) V^\pi(x') \tag{۲}$$

جواب یک MDP سیاست بهینه π^* می‌باشد که بطور همزمان $V^\pi(x)$ را در هر حالت x بیشینه می‌کند. برای هر MDP یک سیاست بهینه قطعی[۱۰] وجود دارد [۹]، تابع مقدار سیاست برای π^*، تابع مقدار بهینه است و در معادلات Bellman برای کنترل صدق می‌کند:

$$\forall x, V^*(x) = \max_{a \in A} [R(x, a) + \gamma \sum_{x' \in X} P(x'|x, a) V^*(x')] \tag{۳}$$

با داشتن تابع مقدار V^*، می‌توان سیاست بهینه را بازیابی نمود، برای این منظور از هر حالت x هر کنشی که مقدار بیشینه در معادله ۳ بدهد، انتخاب می‌شود [۵]. الگوریتمهای محاسبه V^* برای حالتی که فضای حالت MDP کوچک باشد (تقریباً کمتر از $۱۰^۷$ حالت گسسته) شناخته شده هستند؛ بنابراین

---

[۶] discount factor

[۷] trajectory

[۸] مسیر در گراف تصمیم‌گیری شامل تعدادی یال است که یک گره را از طریق گره‌های میانی به گرهی دیگر وصل می‌کند.

[۹] absorbing

[۱۰] deterministic

V^* می‌تواند توسط یک جدول جستجو بیان شود. در MDPهای کوچک، اگر اطلاعات صریحی از مدل گذر $P(x'|x,a)$ و تابع پاداش $R(x,a)$ داشته باشیم آنگاه V^* بصورت دقیق توسط الگوریتمهای کلاسیک مانند برنامه‌ریزی خطی، تکرار سیاست [۱۱]، تکرار سیاست بهبود یافته [۱۲] یا تکرار مقدار [۱۳] بدست می آید [٦]. در MDPهای کوچک وقتی که مدل گذر مشخص نباشد، ابتدا باید V^* را از روی نمونه مسیرهای بدست آمده از شبیه‌سازی فرایند ساخت؛ در این حالت روشهای یادگیری تقویتی (RL) اعمال می‌شوند. روشهای RL یا مبتنی بر مدل هستند که ابتدا یک مدل تجربی از فرایند ساخته و سپس الگوریتمهای کلاسیک گفته شده را بکار می‌برند و یا مستقل از مدل هستند که در اینصورت V^* مستقیماً تخمین زده می‌شود، مانند الگوریتم $TD(\lambda)$، Q-Learning و SARSA [۹]. تقریباً تمامی این الگوریتمها برنامه‌ریزی پویا [۱۴] را بصورت غیر همزمان [۱۵] و افزایشی [۱٦] بکار می‌برند و تحت شرایط مناسبی می‌توان نشان داد که همگی به سمت تابع مقدار بهینه همگرا می‌شوند [٦].

اما برای مسائل تصمیم‌گیری با ابعاد بزرگ وضعیت کاملاً متفاوت است. این مسائل دارای فضای حالت با ابعاد بزرگ می‌باشند، لذا بیان V^* بصورت جدول امکانپذیر نیست— مشکلی که تحت عنوان نفرین بُعد معروف است —. یک روش طبیعی برای برخورد با این مشکل، درنظر گرفتن حالات بصورت برداری با مؤلفه‌های حقیقی و سپس استفاده از یک تخمین‌زننده تابع برای بیان V^* (روی فضای بردارها) می‌باشد. این مبحث تحت عنوان تخمین تابع مقدار [۱۷] (VFA) توسط بویان و دیگران در سال ۱۹۹۵ مطرح شد [۱۰].

---

[۱۱] policy iteration
[۱۲] modified policy iteration
[۱۳] value iteration
[۱۴] dynamic programming
[۱۵] asynchron
[۱٦] incremental
[۱۷] value function approximation

۳—۲ الگوریتم $TD(\lambda)$ خوش بینانه برای تخمین تابع مقدار

Tesauro الگوریتم $TD(\lambda)$ متعلق به Sutton— که این الگوریتم برای تخمین V^{π} برای یک سیاست ثابت طراحی شده بود— را تغییر داد و آنرا برای یادگیری تابع مقدار بهینه V^* و سیاست بهینه بکار برد، این الگوریتم در شکل ۳—۱ نشان داده شده است. تغییر بسیار ساده است: بجای تولید نمونه مسیرها با شبیه‌سازیِ یک سیاست ثابت π، مسیرهای نمونه با سیاست μ که نسبت به تابع مقدارِ تخمینی فعلی V حریصانه است، بدست می آیند:

$$\mu(x) = arg\max_{a\in A}[R(x,a) + \gamma\sum_{x'\in X} P(x'|x,a)V(x')] \qquad (۴)$$

معمولاً حرکتهای غیر حریصانه مکاشفه‌ای نیز انجام می‌شود اما این حرکتها در بازی backgammon، بخاطر تصادفی بودن ذاتی مسئله لازم نبودند. الگوریتم تغییر یافته، $TD(\lambda)$ خوش‌بینانه نام دارد چون اطلاعات کمی در مورد همگرایی آن وجود دارد. هنگامی که $\lambda = 0$ باشد الگوریتم شبیه به برنامه‌ریزی پویای بی‌درنگ[۱۸] (RTDP) می‌شود، جز اینکه RTDP مقادیر نتیجه را در هر حالت با پشتیبانی کامل[۱۹] (میانگین روی تمام نتایج ممکن، مانند روش تکرار مقدار) تخصیص می‌دهد؛ $TD(0)$ نیز مقادیر نتیجه را با ذخیره نمونه[۲۰] تخصیص می‌دهد (یادگیری تنها با مشاهده یک نتیجه). با اعمال $TD(\lambda)$ خوش‌بینانه همراه با یک شبکه عصبی پرسپترون چند لایه، برنامه Tesauro تابع ارزیابی را یاد گرفت که از خود خصوصیات حرفه‌ای نشان می‌داد.

ترکیب $TD(\lambda)$ خوش‌بینانه و شبکه عصبی به دیگر مسائل مانند کنترل آسانسور و زمانبندی کار نیز اعمال شد. ذکر این نکته حائز اهمیت است که وقتی $TD(\lambda)$ همراه با تخمین‌زننده تابعی بکار

[۱۸] realtime dynamic programming

[۱۹] full backup

[۲۰] sample backup

Optimistic $TD(\lambda)$ for value function approximation

Given:

- a simulation model for MDP X;
- a function approximator $V(x)$ parametrized by wector weight $\mathbf{w}$;
- a sequence of step sizes $\alpha_1, \alpha_2, \ldots$ for incremental weight updating; and
- a parameter $\lambda \in [0, 1]$

Output: a weight vector $\mathbf{w}$ such that $V(x) = V^*(x)$

$\mathbf{w} \leftarrow 0$ (or an arbitrary initial estimate)

for n = 1, 2, ... **do**

  using the greedy policy for the current evaluation function $V(x)$, generate a trajectory from a start state in X to termination: $x_0 \rightarrow x_1 \ldots \rightarrow x_T$, record the reward $r_0, r_1, \ldots, r_T$ in each step

  update the weights a V from the trajectory as follows:

  for i = T **downto** 0 **do**

    if i = T

      $targ_i = r_T$ (the terminal reward)

    else

      $targ_i = r_i + \lambda targ_{i+1} + (1 - \lambda)V(x_{i+1})$

    update V's weights by delta rule: $\mathbf{w} \leftarrow \mathbf{w}\ \alpha_n(targ_i - V(x_i))\nabla_w V(x_i)$

شکل ۳–۱: الگوریتم $TD(\lambda)$ خوش‌بینانه بدون تخفیف، برای یک MDP که دارای (یک) حالت جذب‌کننده می‌باشد. این الگوریتم به روزسازی را پس از خاتمه هر مسیر انجام می‌دهد .

برده شود، هیچ تضمینی برای بهینه بودن جواب نمی‌دهد. برای مسئلهٔ یادگیری پیش‌بینی (تخمین V^π) و یادگیری کنترل (تخمین V^*) سؤالهای مهم عبارتند از: (۱) آیا الگوریتم‌های موجود همگرا می‌شوند؟ و (۲) اگر همگرا می‌شوند، نتایج تخمین تا چه حد خوب خواهند بود؟ در بخشهای بعدی تخمین‌زننده‌هایی را که همراه با الگوریتم $TD(\lambda)$ نتایج خوبی داشته‌اند، بررسی می‌کنیم.

۳—۳ تخمین‌زننده‌های تابع

در این بخش مروری کلی بر تخمین‌زننده‌هایی خواهیم داشت که عموماً در مبحث یادگیری تقویتی بکار رفته‌اند و منجر به نتایج خوبی شده‌اند. ابتدا دسته کلی تخمین‌زننده‌های تابع با معماری خطی را بررسی می‌کنیم و سپس به معرفی شبکه‌های عصبی پرسپترونِ چند لایه، شبکه‌های CMAC و توابع پایهٔ شعاعی [۲۱] خواهیم پرداخت.

۳—۳—۱ تخمین‌زننده‌های با معماری خطی

در بیشتر کاربردهای تئوری تخمین، داده‌های آزمایشی در قالب زوج مرتبهای (x_i, y_i) (که خود x_i می‌تواند بردار باشد) داده می‌شوند و ما دنبال ساختن تابع $y = f(x)$ هستیم که ارتباط بین داده‌ها را بیان کند. دو شرط اساسی برای ایجاد یک تخمین‌زننده کارا وجود دارد؛ ابتدا ضروریست که معماری تخمین به اندازه کافی غنی [۲۲] باشد بطوریکه بتواند تابعی تخمینی نزدیک به تابع واقعی بدهد. دوم، پس از انتخاب معماری تخمین، باید روشی آموزشی برای بدست آوردن پارامترهای تخمین ابداع کرد بطوریکه این پارامترها با دقت خوبی بدست آیند. دو معماری کلی برای تخمین تابع وجود دارد: خطی

[۲۱] radial basis functions
[۲۲] rich

و غیر خطی. تخمین‌زننده‌هایی مثل CMAC، چندجمله‌ای‌ها و توابع پایه‌ای شعاعی با مراکز ثابت جزٔ معماری خطی بحساب می آیند و شبکه‌های پرسپترون چندلایه جزٔ معماری غیر خطی محسوب می‌شوند.

معماری خطی دارای فرم کلی زیر می‌باشد:

$$V(x,\beta) = \sum_{k=1}^{K} \beta_k \Phi_k(x) \qquad (5)$$

که $\beta_1, \beta_2, \ldots, \beta_K$ مؤلفه‌های بردار ضرایب β می‌باشند و Φ_k توابع پایه‌ای شعاعی ساده و با مقدار حقیقی می‌باشند. بطور مثال اگر $\Phi_k(x) = x^{k-1}$ باشد آنگاه V تبدیل به چندجمله‌ای می‌شود. در معماری خطی، هدف آموزش و بدست آوردن ضرایب (بردار β) بصورتی است که جمع مجذور خطاها (بین مقادیر واقعی و مقادیر پیش‌بینی شده) کمینه شود:

$$\beta^* = arg \min_{\beta \in R^k} \sum_{i=1}^{K} (y_i - \beta \cdot \Phi_i)^2 \qquad (6)$$

پیدا کردن بردار ضرایب بهینه β^* یک مسئله کمینه‌سازی است که می‌توان آنرا با تکنیکهای جبر خطی بدست آورد. اطلاعات لازم برای محاسبه β^*، ماتریس A (با ابعاد $K \times K$) و بردار b (با طول K) می‌باشد:

$$A = \sum_{i=1}^{N} \Phi_i \cdot \Phi_i^T \quad , \quad b = \sum_{i=1}^{N} \Phi_i \cdot y_i \qquad (7)$$

با داشتن این اطلاعات، ضرایب را می‌توان بصورت زیر بدست آورد:

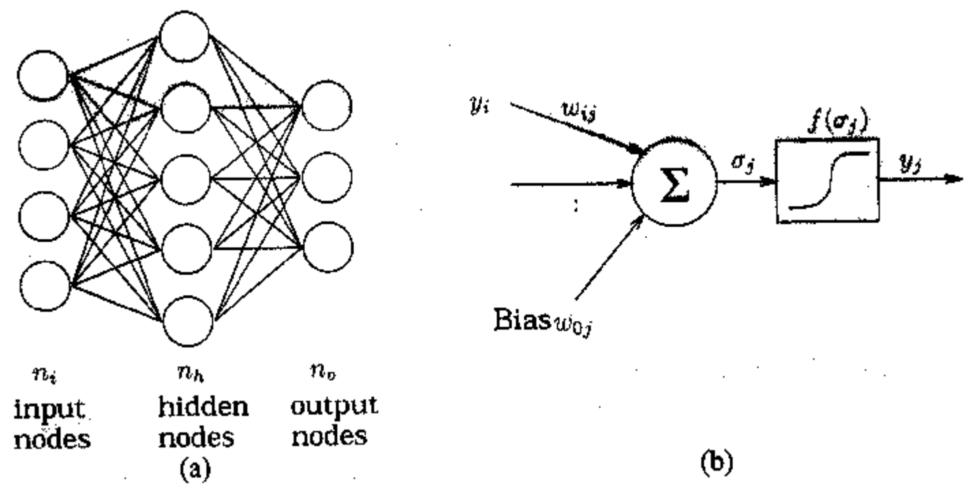
$$\beta^* = A^{-1} \cdot b \qquad (8)$$

در حقیقت اگر گرادیان تخمین‌زننده بر حسب بردار پارامترها (β) خطی باشد آنگاه می‌توان از معادلات ۷ و ۸ بردار ضرایب بهینه را بدست آورد (که در این صورت معماری، خطی نامیده می‌شود)، در غیر

این صورت باید از روشهای آموزش تکرار مبتنی بر گرادیان (مانند انتشار خطا در شبکه‌های عصبی) برای بدست آوردن بردار ضرایب استفاده نمود.

۳—۳—۲ شبکه پرسپترون چندلایه

شبکه‌های پرسپترون چند لایه[۲۳] (MLP) برای بسیاری از کارهای مدل‌سازی و تقریب تابع بکار می‌روند، این شبکه‌ها دارای قابلیت تعمیم بالایی برای داده‌هایی هستند که از قبل دیده نشده‌اند و می‌توانند توابع غیر خطی را یاد بگیرند. یک شبکه MLP با n_i واحد ورودی، n_h واحد پنهان و n_o واحد خروجی در قسمت (a) شکل ۳—۲ نشان داده شده است، در قسمت (b) شکل ۳—۲ نمایی از



شکل ۳—۲: (a) شبکه پرسپترون چند لایه با یک لایه پنهان (b) اجزاء یک گره با جزئیات بیشتر.

[۲۳] multi-layer perceptron

الگوی ورودی در قالب بردار x به واحد ورودی شبکه داده می‌شود، سپس واحد خروجی y_j توسط یک فرایند دو مرحله‌ای محاسبه می‌شود:

$$\sigma_j = \sum_i w_{ij} y_i + w_{\circ j} \; , y_j = f(\sigma_j) \tag{۹}$$

که w_{ij} وزن اتصالی است که خروجی واحد i را به ورودی واحد j وصل می‌کند. معمولاً تابع غیر خطی $f(\sigma_j)$ را تابع سیگموئید انتخاب می‌کنند:

$$f(\sigma_j) = \frac{۱}{۱ - exp(-\sigma_j)} \tag{۱۰}$$

که مشتق آن بصورت زیر است:

$$\frac{\partial f(\sigma_j)}{\partial \sigma_j} = y_j(۱ - y_j) \tag{۱۱}$$

فرض کنید خروجی دلخواه y^* باشد و خروجی واقعی شبکه y باشد. خطا را مجموع مجذور خطاهای خروجی‌ها (روی N داده آموزشی) بصورت زیر تعریف می‌کنیم:

$$E = \frac{۱}{۲} \sum_{p=۱}^{N} \sum_{j=۱}^{n_\circ} (y^*_{pj} - y_{pj})^۲ \tag{۱۲}$$

می‌توان از روش نزول گرادیان (در قالب الگوریتم back propagation) برای تنظیم (آموزش) وزنهای شبکه استفاده کرد. با توجه به دادهٔ آموزشی p، تغییرات وزن بصورت زیر است:

$$\Delta_p w_{ij} = \eta \delta_{pj} y_{pi} \tag{۱۳}$$

که η نرخ یادگیری است. اگر واحد خروجی از تابع سیگموئید استفاده کند:

$$\delta_{pj} = (y^*_{pj} - y_{pj}) y_{pj} (۱ - y_{pj}) \tag{۱۴}$$

و اگر در لایه پنهان نیز از تابع سیگموئید استفاده شود:

$$\delta_{pj} = y_{pj}(۱ - y_{pj}) \sum_k \delta_{pk} w_{jk} \tag{۱۵}$$

که واحد k در لایه بعدی است. نقطه ضعف اصلی شبکه‌های MLP این است که به تعداد تکرار زیادی برای یادگیری احتیاج دارند.

۳—۳—۳ شبکه‌های با توابع پایه شعاعی

معماری روش توابع پایه شعاعی، در حالت کلی غیر خطی می‌باشد؛ خروجی شبکه بصورت زیر است:

$$f(x) = \sum_{j=1}^{N} \alpha_j \Phi(||X - X_j||) \qquad (16)$$

که X بردار ورودی و α_j ضرایب وزنی متناظر با هر داده آموزشی X_j می‌باشد. $\Phi(.)$ تابع پایه شعاعی یا تابع هسته‌ای با مرکز X_j می‌باشد که اثر هر بردار آموزشی روی مقدار خروجی را بیان می‌کند. تابعِ پایه شعاعی، هر تابعی می‌تواند باشد بشرطی که مقدار حقیقی نامنفی تولید کند و دارای مقدار کمینه در صفر باشد و با نزدیک شدن به بینهایت به سمت صفر میل کند، مانند $e^{-\frac{x^2}{2}}$. با انتخاب تابع $\Phi(.)$ و داشتن یک مجموعه از داده‌های آموزشی (x_j, y_j)، $j = 1 \dots N$، می‌توان یک فرمول بسته برای محاسبه α_j ارائه داد (چون مراکز ثابت هستند)[۱۱].

شبکه تخصیص منبع

بجای قرار دادن توابع پایه در هر نقطهٔ آموزشی، می‌توان آنها را تنها در نقاط ضروری در نظر گرفت؛ شبکه تخصیص منبع [۲۴] (RAN) تنها موقعی یک تابع پایه‌ای تخصیص می‌دهد (یا یک نرون به لایه پنهان اضافه می‌کند) که الگوی ورودی جدید [۲۵] باشد، یعنی از مراکز فعلی بسیار دور باشد یا

[۲۴] resource allocating network

[۲۵] novel

خطای زیادی تولید کند. وقتی تابع پایه‌ای جدیدی تخصیص نیابد، مراکز و ضرایب به روز می‌شوند. Anderson این ساختار (RAN) را طوری تغییر داد که نرونهای لایه پنهان دوباره از اول تخصیص داده می‌شوند (به جای اینکه تنها یک نرون جدید به این لایه اضافه کند) [۱۱]. او این الگوریتم را با الگوریتم Q-Learning برای حل مسئله حفظ تعادل پاندول معکوس [۲۶] بکار برد. ساختار RAN در شکل ۳—۳ نشان داده شده است.

اغلب در RAN از تابع گاوسی [۲۷] استفاده می‌شود، خروجی شبکه هنگامی که K تابع پایه‌ای داشته باشیم عبارتست از:

$$f(x) = \alpha_\circ + \sum_{k=۱}^{K} \alpha_k \Phi_k(x) \qquad (۱۷)$$

که تابع پایه‌ای شعاعی اینگونه است:

$$\Phi_k(x) = exp[-\frac{۱}{۲{\sigma_k}^۲}||x-\mu_k||^۲] \qquad (۱۸)$$

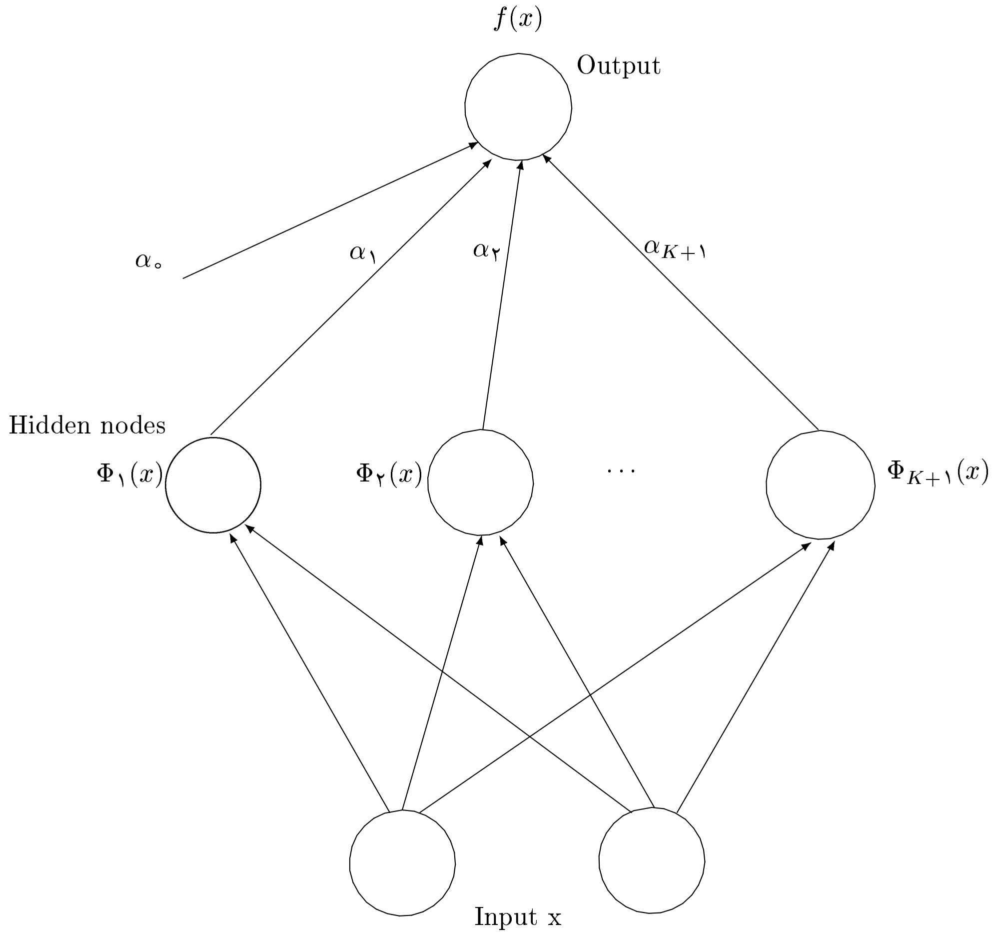
که μ_k مرکز (یا میانگین) و σ_k عرض (یا انحراف استاندارد) تابع گاوسی می‌باشد. پارامترهای شبکه عبارتند از: $\mathbf{w} = [\alpha_\circ, \ldots, \alpha_K, \mu_۱^T, \ldots, \mu_K^T]^T$.

الگوریتم یادگیری این شبکه بصورت شکل ۳—۴ است که در آن $a_i = \nabla_w f(x)$ گرادیان تابع $f(x_i)$ نسبت به بردار ضرایب w می‌باشد و η نرخ یادگیری است. پارامتر ϵ_i مقدار مرزی [۲۸] فاصله است که وقتی فاصله از آن بیشتر شد یک تابع پایه جدید تخصیص داده می‌شود. این مقدار به تدریج از مقدار اولیه ϵ_{max} به ϵ_{min} کاهش می‌یابد (با نرخ $\gamma < ۱$). مقدار مرزی خطا e_{min} با توجه به دقت خواسته شده از شبکه تعیین می‌شود. در نهایت پارامتر K عرض تابع پایه‌ای جدید را بر حسب فاصله

[۲۶] inverted pendulum problem

[۲۷] Gaussian

[۲۸] thredshold



شکل ۳–۳: ساختار شبکه تخصیص منبع که برای توابع گاوسی پیاده‌سازی شده است.

Training RAN Algorithm

$\epsilon_i \leftarrow \epsilon_{max}, \alpha_0 \leftarrow y_0$

for each observation (x_i, y_i)

$\quad \epsilon_i \leftarrow max(\epsilon_{max}\gamma^i, \epsilon_{min})$

$\quad$ error $e_i \leftarrow y_i - f(x_i)$

$\quad$ **if** $e_i > \epsilon_{min}$ and $||x_i - \mu_{nr}|| > \epsilon_i$

$\quad\quad$ allocate a new basis function with $\alpha_{k+1} \leftarrow e_i \;\; , \;\; \mu_{k+1} \leftarrow x_i \;\; , \;\; \sigma_{k+1} \leftarrow K||x_i - \mu_{nr}||$

$\quad$ **else**

$\quad\quad w_i \leftarrow w_{i-1} + \eta e_i a_i$

شکل ۳—۴: الگوریتم آموزش به یک شبکه تخصیص منبع.

میان بردار ورودی و نزدیکترین مرکز به آن μ_{nr} تعیین می‌کند.

۳—۳—۴ شبکه CMAC

شبکه CMAC [۲۹] اولین بار توسط Albus در سال ۱۹۷۱ به عنوان یک مدل پردازشگر اطلاعات پیشنهاد شد [۱۱]. CMAC یک ساختار کدکننده درشت‌دانه است که در آن هر نقطه ورودی یک مجموعه از فیلدهای درک [۳۰] را در هر بُعد از ورودی تحریک می‌کند.

فیلدهای درک [۳۱] توسط توابع کوانتیزه‌کننده تعیین می‌شوند. وقتی فیلدهای درکِ متناظر با یک

[۲۹] cerebellar model articulation controller

[۳۰] receptive field

[۳۱] فضای ورودی‌ها توسط توابع کوانتیزه‌کننده، در هر بُعد به بازه‌های مشخصی تقسیم می‌شوند که به هر یک از این بازه‌ها، فیلد درک گفته می‌شود.

تابع کوانتیزه‌کننده در همهٔ ابعاد ورودی ترکیب شوند یک فرامکعب [۳۲] در فضای چند بُعدی ورودی بدست می آید. این فرامکعب متناظر با یک عنصر [۳۳]، مشخص کننده مقدار خروجی است. چون معمولاً چند تابع کوانتیزه کننده وجود دارد، لذا تعدادی فرامکعب تحریک می‌شوند و مقدار خروجی از ترکیب وزن‌دار خروجی این فرامکعبها بدست می آید [۱۱].

نگاشتها در شبکه CMAC

هر گاه که یک بردار ورودی به شبکه CMAC داده شود، دو نگاشت اتفاق می‌افتد. نگاشت اول بردار حقیقی ورودی s را به یک بردار باینری x تبدیل می‌کند (توسط یک تابع غیرخطی مانند $\Phi(.)$)، یعنی $x = \Phi(s)$. نگاشت دوم که خطی است x را در بردار وزن w ضرب می‌کند و خروجی y را تولید می‌نماید. اثر ترکیبی این دو نگاشت بصورت $y = g(s)$ نوشته می‌شود. تابع نگاشت غیر خطی $\Phi(.)$ دارای سه مرحله است:

$$\underbrace{s \rightarrow m^* \rightarrow A^*}_{\Phi(.)} \rightarrow A_p^* \rightarrow y$$

در مدل زیستی، مجموعه m^* همان مجموعه سلولهای سنسوری است که بعضی از آنها توسط الگوی ورودی تحریک می‌شوند. سلولهای سنسوری فعال، باعث آتش کردن سلولهای متناظر در مجموعه A^* می‌شود. در اینجا m^* شامل فیلدهای درک و مجموعه A^* شامل فرامکعبها در فضای ورودی است که وزنهای دخالت کننده در مقدار خروجی را علامت‌گذاری می‌کند. چون A^* می‌تواند بسیار بزرگ باشد، یک نگاشت از این فضای بزرگ به فضای واقعی حافظه فیزیکی (که معمولاً بسیار کوچکتر است)، توسط A_p انجام می‌گیرد. نگاشت $s \rightarrow A^*$ را می‌توان به عنوان گام تعمیم‌دهی تصور نمود و

[۳۲] hypercube

[۳۳] component

نگاشت $A^* \to A_p^*$ را به عنوان گام کاهش حافظه در نظر گرفت.

می‌توان نگاشت را توسط عبارات ریاضی نشان داد؛ مقدار خروجی y با جمع وزنهای فعال در A^* بدست می آید:

$$y = g(s) = \sum_{l=0}^{K-1} w(f_{l1}, \ldots, f_{lN}) \quad (19)$$

که f_{lj} به بازه کوانتیزهٔ lام در بُعد jام ورودی اطلاق می‌شود. شاخص [34] بازهٔ کوانتیزه lام بصورت زیر بدست می آید:

$$f_{li} = f_i(s_i, l, k)$$

که $f_i(s_i, l, k)$ تابع کوانتیزه برای بعد i است:

$$f_i(s_i, l, k) = int(\frac{s_i}{tQ_i} + \frac{l}{k})$$

آخرین مرحله نگاشت مربوط به نگاشت A^* به A_p^* می‌باشد که این کار توسط hashing انجام می‌پذیرد. اکنون فرض می‌کنیم مجموعه‌های A^* و A_p^* یکسان هستند. معادله ۹۱ را بصورت برداری می‌توان نوشت:

$$y = w^T \cdot x$$

که x و w بردارهای ستونی با تعداد مؤلفه‌های $K \prod_{i=1}^{N} Q_i$ می‌باشند. بردار w شامل وزنها در A^* می‌باشد و بردار x در محلهای شاخص دارای مقدار ۰ و در بقیه جاها دارای مقدار ۱ می‌باشد. برای بدست آوردن مکان وزن فعال، از رابطه زیر استفاده می‌شود:

$$index_l = f_{l1} + \sum_{i=2}^{N} f_{li} \prod_{j=1}^{i-1} Q_j \quad (20)$$

[34] index

چون بردارهای w و x دارای مؤلفه‌های بسیار زیادی هستند و خیلی از آنها در بردار x صفر می‌باشند، بهتر است مستقیماً وزن مؤلفه‌های فعال را با هم جمع کنیم:

$$y = g(s) = \sum_{l=1}^{K-1} w_{[index_l]} \qquad (21)$$

آموزش در شبکه CMAC

تغییرات وزنها در شبکه CMAC در گام زمانی iام بصورت زیر محاسبه می‌شود:

$$\triangle w_i = \frac{\alpha}{K}(y^* - w_i^T \cdot x_i) \cdot x_i \qquad (22)$$

که همان الگوریتم LMS [35] می‌باشد. y^* مقدار خروجی مطلوب و α نرخ یادگیری است.

غیر خطی بودن تخمین توسط CMAC از غیر خطی بودن نگاشت بردار حقیقی ورودی s به بردار x نشأت می‌گیرد؛ تغییرات وزن در فضای x خطی است. بنابراین تمام اثباتهای همگرایی مطرح شده توسط Widrow و Hoff (برای عناصر وفقی خطی) در اینجا نیز قابل اعمال است [۱۱]. استفاده از الگوریتم بالا با شرط $0 < \alpha < 1$ همگرایی وزنها را به مقداری که میانگین مجذور خطا کمینه گردد، تضمین می‌کند.

چون خروجی شبکه CMAC را می‌توان بصورت $y = w^T \cdot x$ نوشت، W را می‌توان از رابطه زیر بدست آورد:

$$W = [\Phi_{N_O}^T \cdot \Phi_{N_O}]^{-1} \cdot \Phi_{N_O}^T \cdot y_{N_O} \qquad (23)$$

[35] least mean square

که

$$\Phi_{N_O} = \begin{bmatrix} x_1^T \\ \vdots \\ x_{N_O}^T \end{bmatrix} \quad , \quad y_{N_O} = \begin{bmatrix} y_1^* \\ \vdots \\ y_{N_O}^* \end{bmatrix} \tag{۲۴}$$

البته اگر تعداد مشاهدات (یا نمونه‌ها) N_O بزرگتر از تعداد ضرایب N_W باشد و معکوس $[\Phi_{N_O}^T \cdot \Phi_{N_O}]$ وجود داشته باشد.

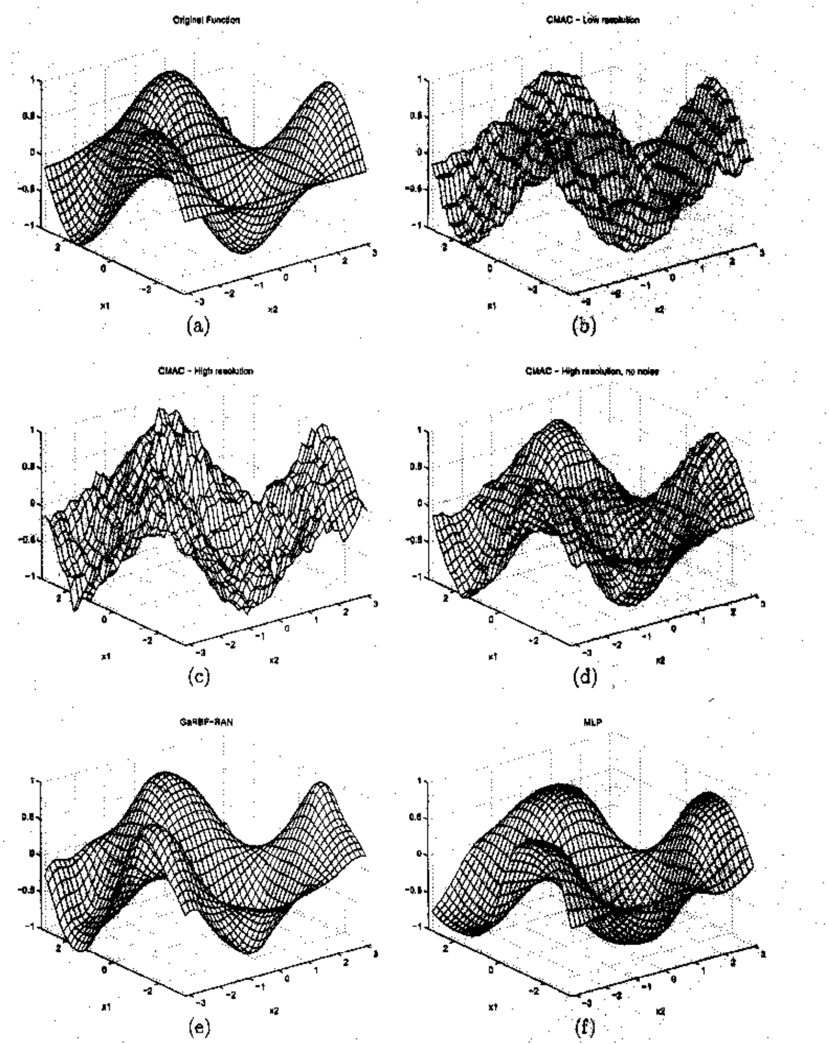
۳—۳—۵ مقایسه تخمین‌زننده‌های مختلف

تابع زیر را در نظر بگیرید:

$$y = sin(x_1)Cos(x_2) + n(\sigma_n) \tag{۲۵}$$

که در آن $n(\sigma_n)$ نویز گاوسی با میانگین صفر و انحراف استاندارد σ_n می‌باشد. سعی می‌کنیم با آموزش با سرپرستی[۳۶] این تابع را به هر کدام از تخمین‌زننده‌های بالا یاد بدهیم و مقایسه کارایی تخمین‌زننده‌ها را با مقایسه تابع یاد گرفته شده توسط هر کدام از آنها، انجام دهیم. فضای ورودی دو بُعدی است و $-\pi < x_1, x_2 < \pi$، بنابراین $-1 < y < 1$. در شکل ۳—۵ می‌توانید شکل توابع یاد گرفته شده را ببینید. توابع یاد گرفته شده با شبکه پرسپترون چند لایه و توابع پایه‌ای شعاعی، حتی با وجود نویز هموار هستند در حالیکه شکل یاد گرفته شده توسط شبکه CMAC ناهموار (دندانه‌دار) است. برای اطلاعات بیشتر می‌توانید به [۱۱] مراجعه کنید.

[۳۶] supervised learning



شکل ۳–۵: شکل توابع یادگرفته‌شده توسط تخمین‌زننده‌های مختلف تابع. a) تابع اصلی (اولیه). b) CMAC با resolution پایین. c) CMAC با resolution بالا. d) CMAC با resolution بالا بدون نویز اضافی. e) تابع شعاعی پایه که تابع پایه گاوسی است. f) شبکه پرسپترون چندلایه.

۳—۴ روش یادگیری فعال

روش یادگیری فعال[۳۷](ALM) بر مبنای این ایده است که بخاطر سپردن اعداد برای مغز انسان مشکل است اما نگهداری رفتار عمومی تابع کاری آسان است. در این روش، فرض می‌شود یادگیری انسان تشکیل شده است از یادگیری سیستم‌های ساده‌ای که تک‌ورودی—تک‌خروجی هستند و رفتار کلی یک سیستم با ترکیب اثر این سیستم‌های ساده بیان می‌شود.

در این روش، رفتار خروجی بر اساس هر یک از ورودی‌ها در نظر گرفته می‌شود، سپس سعی می‌شود یک مسیر هموار (که بیانگر رفتار خروجی بر اساس آن ورودی است) از داده‌های نمونه گذرانده شود. برازاندن این منحنی توسط روشی به نام IDS[۳۸] صورت می‌گیرد که روشی فازی است. منحنی تولید شده حاوی دو اطلاع بسیار مهم است: بیان رفتار خروجی بر اساس ورودی و وابستگی خروجی به این ورودی. حال اگر بخواهیم خروجی متناظر با یک بردار ورودی که از قبل دیده نشده است را پیدا کنیم ابتدا باید خروجی‌های میانی را بر اساس سیستم‌های تک‌ورودی—تک‌خروجی بدست آوریم و سپس بر اساس وابستگی خروجی به هر کدام از ورودی‌ها (که این اطلاع از IDS بدست آمده است) این خروجی‌های میانی را (بطور فازی) ترکیب و خروجی کلی را بدست آوریم. برای اطلاعات بیشتر می‌توانید به [۳] مراجعه کنید.

[۳۷] active learning method

[۳۸] ink drop spread

۳—۵ بحث

در این فصل مدل تصمیم‌گیری مارکوف و سپس الگوریتمهای یادگیری تقویتی برای پیدا کردن سیاست بهینه (یا نزدیک به بهینه) را مرور کردیم. دیدیم هنگامی که تعداد حالات بسیار زیاد باشد، بهتر است که تابع ارزیابی را بجای جدول جستجو با یک تابع تخمینی نشان بدهیم. برای توابع تخمینی، معماریهای خطی و غیر خطی را بررسی کردیم و نهایتاً الگوریتم یادگیری فعال را معرفی نمودیم. این الگوریتم مبتنی بر دیدِ فازی است و از فرایند یادگیری مغز انسان الهام گرفته شده است.

هدف من از باز کردن بحث یادگیری تقویتی و فرایند تصمیم‌گیری مارکوف، مرور یک الگوریتم بهینه‌سازی برای مسائل ترکیبیاتی در فصل بعد است. اگر حل مسائل بهینه‌سازی را بصورت جستجو در یک فضای گسسته در نظر بگیریم، آنگاه به تشابه پیدا کردن تابع ارزیابی بهینه (و نهایتاً سیاست بهینه) و پیدا کردن جواب بهینه مسئله ترکیبیاتی پی خواهیم برد. در فصل بعد الگوریتم STAGE را که بر پایه این ایده شکل گرفته است، مرور می‌کنیم و اثر پارامترهای مختلف را بر کارایی آن تحلیل می‌کنیم.

فصل ۴

یادگیری تقویتی در بهینه‌سازی ترکیبیاتی

می‌توان فرایند بهینه‌سازی یک مسئله ترکیبیاتی را بصورت جستجو در یک فضای حالات در نظر گرفت. رئوس (حالات)، جوابهای امکانپذیر [1] هستند و یالها، متناظر با اعمال کنش روی یک حالت و رسیدن (تبدیل) به حالت دیگر می‌باشند. پیمایش تمام جوابهای کاندید (حالات) زمان زیادی مصرف می‌کند (که این به خاصیت NP-Hard بودن این مسائل برمی‌گردد). اگر بتوان بطور هوشمند تنها قسمتی از فضای جواب را جستجو کرد که دارای جواب (نزدیک به) بهینه می‌باشد، آنگاه در مدت زمان کمی، جوابی خوب بدست آمده است. بنابراین مسئله اصلی انتخاب این زیرمجموعه از حالات می‌باشد. با معرفی تابع ارزیابی — که معیاری برای خوب بودن هر حالت بر اساس خصوصیات آماری آن حالت را بیان می‌کند — می‌توان زیرمجموعه حالات با ارزش را انتخاب نمود. در واقع الگوریتم STAGE نیز بر پایه همین ایده است.

در این فصل ابتدا تشریح خواهیم کرد که چگونه می‌توان حل یک مسئله بهینه‌سازی ترکیبیاتی را بصورت جستجو در یک فضای جواب در نظر گرفت. سپس در ادامهٔ کار Boese و دیگران در تحلیل

[1] feasible

ساختار هزینه برای مسائل تقسیم‌بندی گراف و TSP، این کار را برای مسئله bin packing انجام می‌دهیم. نتایج کار این انگیزه را تقویت می‌کنند که آیا می شود الگوریتمی ابتکاری ارائه داد که با داشتن تجربه از نقاط بهینه محلی قبلی، نقاط بهینه جدید و کم‌هزینه‌تری را پیشنهاد کند؟ در عمل خواهیم دید که این حرف با معرفی الگوریتم STAGE تحقق می‌یابد؛ این الگوریتم در سال ۹۸ توسط Boyan و Moore ارائه شد. الگوریتم STAGE از نقاط بهینه محلی قبلی استفاده می‌کند و بطور هوشمند نقاط جدیدی پیشنهاد می‌کند که پیش‌بینی می‌شود با اجرای یک الگوریتم جستجوی محلی از این نقاط، به جوابهای کم‌هزینه‌تری برسد.

۴–۱ فضای جستجو در مسائل ترکیبیاتی

یک مسئله بهینه‌سازی ترکیبیاتی شامل یک تابع هدف و مجموعه‌ای از نمونه‌های مسئله می‌باشد. هر نمونه مسئله توسط مجموعه‌ای از متغیرها و مجموعه‌ای از قیود بین آنها مشخص می‌شود. یک جواب کاندید [۲] برای نمونه مسئله عبارتست از انتصاب مقادیر به متغیرهای نمونه مسئله. یک جواب امکانپذیر [۳] عبارتست از جواب کاندیدی که قیود را برآورده سازد. جواب بهینه سراسری، جواب امکانپذیری است که تابع هدف را بهینه کند.

می‌توان فرایند بهینه‌سازی یک مسئله ترکیبیاتی را بصورت جستجو در یک فضای حالات در نظر گرفت؛ حالات (رئوس) جوابهای امکانپذیر هستند و یالها، متناظر با اِعمال کنش روی یک جواب و رسیدن به یک جواب دیگر می‌باشند. فرایند جستجو برای پیدا کردن جواب بهینه عبارتست از شروع از یک جواب تصادفی اولیه و طی مسیری (اعمال کنشهایی) که جواب اولیه را به جواب بهینه تبدیل

[۲] candidate solution

[۳] feasible solution

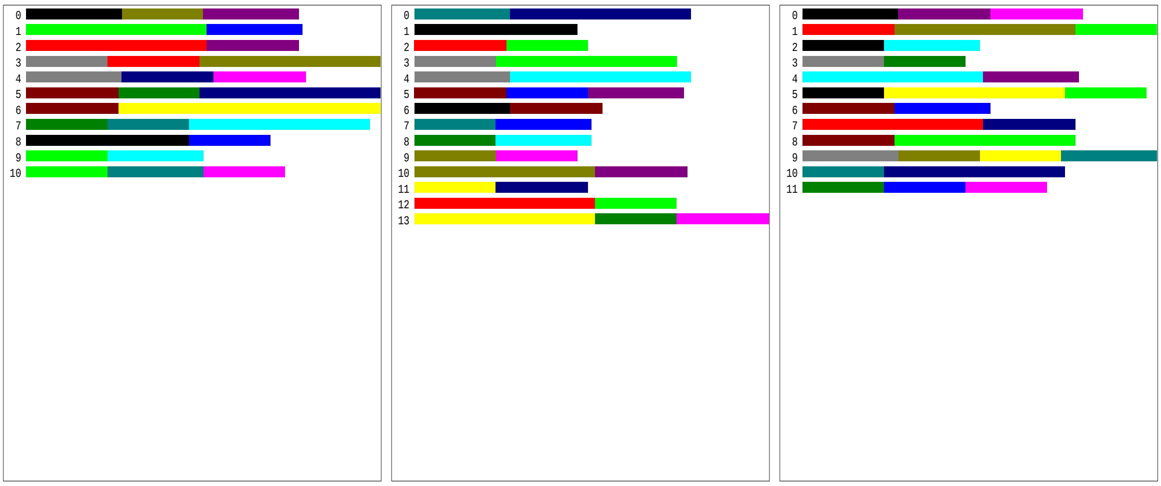
می‌کند، این مطلب را با ذکر یک مثال توضیح می‌دهیم.

فرض کنید می‌خواهیم مسئله bin-packing را به اینصورت مدل کنیم؛ در این مسئله یک لیست از n بسته $L = (a_1, \ldots, a_n)$ و یک عدد C بعنوان ظرفیت هر یک از n سطل به ما داده می‌شود و هدف قرار دادن بسته‌ها در حداقل تعداد سطلها می‌باشد. بعبارتی می‌خواهیم بسته‌ها را به m زیرمجموعه افراز کنیم $B_1, \ldots, B_m$ بطوریکه برای هر B_j داشته باشیم: $\sum_{a_i \in B_j} s(a_i) \leq C$. این مسئله NP-Complete است و موارد کاربرد زیادی در زندگی روزمره دارد [۱۲]. شکل ۴—۱ یک نمونه از مسئله را با ۳۰ سطل نشان می‌دهد؛ در حالت قرار دادن بهینه در سطلها، این بسته‌ها در ۹ سطل قرار می‌گیرند بطوریکه از تمام حجم سطلها استفاده می‌شود.

برای در نظرگیری این مسئله بصورت فضای جستجو، حالت (رأس) را به اینصورت تعریف می‌کنیم: انتصاب اعداد ۱ تا n به بسته‌ها که $b(a_i) = j$ نشان دهنده قرار گرفتن بسته a_i در سطل jام می‌باشد:

$$b(a_1) = j_1, \ldots, b(a_n) = j_n$$

در ضمن این جواب کاندید باید امکانپذیر باشد، یعنی مجموع حجم بسته‌های قرار گرفته در یک سطل باید کوچکتر یا مساوی حجم سطل باشد. کنش (یال) را در گراف جستجو این گونه تعریف می‌کنیم: برداشتن یک بسته از یک سطل و انداختن آن در سطلی دیگر بطوریکه محدودیت ظرفیت حفظ شود. به این ترتیب هر حالت حداکثر دارای $(n-1)^2$ حالت همسایه خواهد بود.



شکل ۴–۱: سه بهینه محلی مسئله bin-packing. هر مستطیل رنگی نشان دهندهٔ یک بسته است و طول آن متناسب با وزن بسته می‌باشد.

۴–۱–۱ جستجوی محلی

می‌توان بطور تصادفی یک جواب اولیه از فضای جستجو را انتخاب کرد و سپس بطور حریصانه [۴] از بین حالات همسایه آن، حالتی را انتخاب کرد که مقدار تابع هدف را بهبود ببخشد. اگر این استراتژی را در پیش بگیریم، در نهایت به حالتی می‌رسیم که مقدار تابع هدف در آن حالت از این مقدار در همسایه‌های آن، بهتر است. اصطلاحاً به این حالت یک بهینه محلی [۵] گفته می‌شود، سه بهینه محلی برای مسئله bin-packing در شکل ۴–۱ دیده می‌شود. ممکن است تصور کنیم هر بهینه محلی می‌تواند با تقریبب خوبی به جواب بهینه سراسری نزدیک باشد، اما این مطلب درست نیست.

[۴] greedy

[۵] local optimum

مجموعهٔ جوابهای بهینه محلی، دارای خاصیتی است که ما آن را «فاجعه حد مرکزی [۶]» می‌نامیم. فاجعه حد مرکزی هنگامی اتفاق می‌افتد که همگی بهینه‌های محلی بجز تعداد بسیار کمی (بصورت معکوسِ نمایی)، دارای کیفیت میانگین هستند (یعنی دارای هزینه‌ای نزدیک به میانگین هزینه یک جواب بهینه محلی هستند). با بزرگ شدن اندازه مسئله، اگر هزینه جوابهای بهینه محلی تقریباً بصورت نرمال توزیع شده باشد و اگر تعداد جوابهای بهینه محلی بصورت نمایی رشد کند آنگاه این پدیده اتفاق می‌افتد. اگر اختلاف هزینه بین جواب محلی میانگین و جواب بهینه سراسری را به یک نرمال کنیم آنگاه انحراف استاندارد هزینه‌های بهینه محلی به سمت صفر میل می‌کند و همه (بجز تعداد بسیار کمی) هزینه‌ای تقریباً برابر با بهینه محلی میانگین خواهند داشت[۷] [۷].

ذکر دو نکته در اینجا ضروری به نظر می‌رسد. اولاً Boyan به این مطلب توجه نمود که کیفیت جواب بدست آمده از الگوریتم جستجوی محلی به نقطه اولیه (شروع) بستگی دارد [۹]. بعبارتی اگر نقطه اولیه بطور مناسبی انتخاب شود می‌توان انتظار داشت که جواب نهایی تا حد خوبی به جواب بهینه نزدیک باشد. او سعی کرد توسط یک تابع ارزیابی به هر حالت عددی نسبت دهد که نشان دهنده کیفیت جواب بدست امده از اجرای یک الگوریتم جستجوی محلی باشد، این اساس الگوریتم STAGE است که در بخشهای بعدی مرور خواهیم کرد.

ثانیاً الگوریتمهای جستجوی محلی مختلفی را می‌توان در نظر گرفت. در ساده‌ترین حالت تپه‌نوردی با بیشترین نزول [۸] به تمام حالات همسایه $N(x)$ یک حالت x نگاه می‌کند و حالتی را انتخاب می‌کند که بهترین مقدار تابع هدف را در بین همسایه‌ها دارد (البته این مقدار باید بهتر از مقدار

---

[۶] central limit catastrophe

[۷] این پدیده به این دلیل فاجعه «حد مرکزی» نامیده می‌شود که توزیع بهینه‌های محلی در نمونه مسائل عملی، تقریباً نرمال است و انحراف استاندارد این توزیع با بزرگ شدن اندازه مسئله به سمت صفر میل می‌کند (طبق قضیه حد مرکزی، میانگین هر متغییر تصادفی، با زیاد شدن نمونه‌ها به سمت توزیع نرمال با واریانس صفر میل می‌کند).

[۸] steepest descent hill climbing

تابع هدف در x نیز باشد). هنگامی که تعداد حالات همسایه $|N(x)|$ زیاد باشد، بررسی تمام عناصر این مجموعه کاری زمان‌بَر است و خوبست خود را به زیرمجموعه‌ای تصادفی از این حالات محدود کنیم. این بهانه‌ای برای معرفی الگوریتم تپه‌نوردی تصادفی [۹] می‌باشد. در این الگوریتم زیرمجموعه‌ای شامل تعداد مشخصی عنصر از عناصر همسایه [۱۰] بطور تصادفی انتخاب می‌شوند و سپس حالتی انتخاب می‌شود که مقدار تابع هدف در آن بهتر از بقیه حالات (و حالت x) باشد. استراتژی بعدی تپه‌نوردی با اولین بهبود [۱۱] می‌باشد. در این الگوریتم بصورت سیستماتیک تمام همسایه‌های حالت x در نظر گرفته می‌شوند و اولین جواب که مقدار تابع هدف در آن بهتر از مقدار تابع هدف در حالت x باشد به عنوان حالت بعدی در نظر گرفته می‌شود. در بخشهای بعدی اثر زیرالگوریتمهای جستجوی محلی مختلف را بر کارایی STAGE بررسی خواهیم کرد.

۴—۱—۲ استفاده از مشخصه‌های دیگرِ حالت

جستجوی محلی معادل است با «تلاش برای پیدا کردن قله اورست در مه غلیظ در حالت سرگیجه»[۹]. هنگامی که کوه‌نورد می‌خواهد یک گام بردارد، برای تصمیم‌گیری به ارتفاع سنج خود توجه می‌کند. اما فرض کنید که کوه‌نورد به اطلاعات اضافی نیز دسترسی دارد، مانند مختصات (x, y) محل خود، شیب کوه در آن نقطه و ارتفاع نزدیکترین درخت. این مشخصه‌های اضافی موجب می‌شوند که کوه‌نورد گام‌های خود را با آگاهی بیشتر و کاراتر بردارد. در بهینه‌سازی نیز چنین مشخصه‌های اضافی می‌توانند بسیار مفید باشند. به عنوان مثال مشخصه‌های اضافی که می‌توان در مسئله bin-packing در نظر گرفت عبارتند از:

---

[۹] stochastic hill climbing

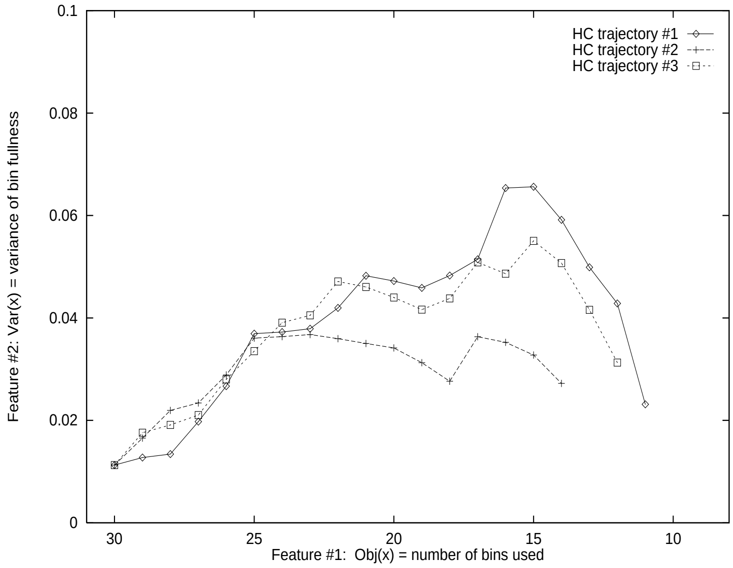
[۱۰] که به تعداد این همسایه‌ها، Patience گفته می‌شود.

[۱۱] first improvement hill climbing

- variance در پر شدن سطلها.
- variance تعداد بسته در هر سطل.
- نسبت متوسط حجم بزرگترین بسته به کوچکترین بسته در هر سطل.
- $\vdots$

چنین مشخصه‌های اگر با وزن مناسب ترکیب شوند، تابع هدفی می‌سازند که نه تنها می‌تواند نشان‌دهنده خوب بودن یک حالت به عنوان جواب نهایی باشد بلکه می‌تواند نشان‌دهنده خوب بودن حالت به عنوان حالتِ اولیه (شروع) برای یک الگوریتم جستجوی محلی نیز باشد.

برای روشن شدن موضوع به شکل ۴—۲ توجه کنید. این شکل سه مسیر تپه‌نوردی که منجر به بدست آمدن سه جواب بهینه محلی می‌شود را نشان می‌دهد. هر حالت مشاهده شدهٔ x_i بصورت یک نقطه در صفحه دوبعدی رسم شده است که محور اول $Obj(x_i)$ و محور دوم واریانس در پرشدگی سطلها (متناظر با جواب x_i) می‌باشد. هر سه مسیر از حالت اولیه نشان داده شده در شکل ۴—۱ شروع شده‌اند: هر سه مسیر از گوشه سمت چپ و پایین شروع شده‌اند که به معنای واریانس کم و $Obj = ۳۰$ می‌باشد؛ در ابتدا چون بیشتر سطلها خالی هستند واریانس کم است. مسیرها از سمت راست گسترش می‌یابند چون قبول هر کنش یک واحد از تعداد سطلهای مورد نیاز می‌کاهد. مقدار واریانس ابتدا افزایش می‌یابد (هنگامی که بعضی سطلها بیشتر از بقیه پر می‌شوند) و سپس شروع به کاهش می‌کند (در انتهای مسیر که تقریباً همهٔ سطلها پر می‌شوند). مسیرها در بهینه محلی ۱۱، ۱۴ و ۱۲ خاتمه یافته‌اند.



شکل ۴–۲: مسیرهای بدست آمده با الگوریتم تپه‌نوردی تصادفی در فضای مشخصه‌های مسئله .bin-packing

نکته کلیدی شکل ۴–۲ این است که مشخصه واریانس می‌تواند برای پیش‌بینی نتیجه جستجوی محلی از یک حالت مفید باشد. مسیرهای تپه‌نوردیی که از حالات با واریانس زیادتر می‌گذرند در انتها نیز تمایل دارند که به جوابهای بهتری برسند. این موضوع عاقلانه به نظر می‌رسد: حالات با واریانس بیشتر دارای تعداد بیشتری سطل تقریباً خالی هستند که راحت‌تر می‌توان آنها را خالی نمود. این یک نمونه از دانشی است که ما قصد داریم به تابع ارزیابی اضافه کنیم تا الگوریتم جستجوی محلی ما را یاری نماید.

۴–۲ ساختار سطح هزینه مسائل ترکیباتی

با توجه به مباحث بخش قبل، فرض کنید می‌خواهیم یک تابع ارزیابی بدست آوریم که الگوریتم جستجوی محلی را راهنمایی کند، بعبارتی این تابع ارزیابی طوری حالات را ارزش‌دهی کند که گویی

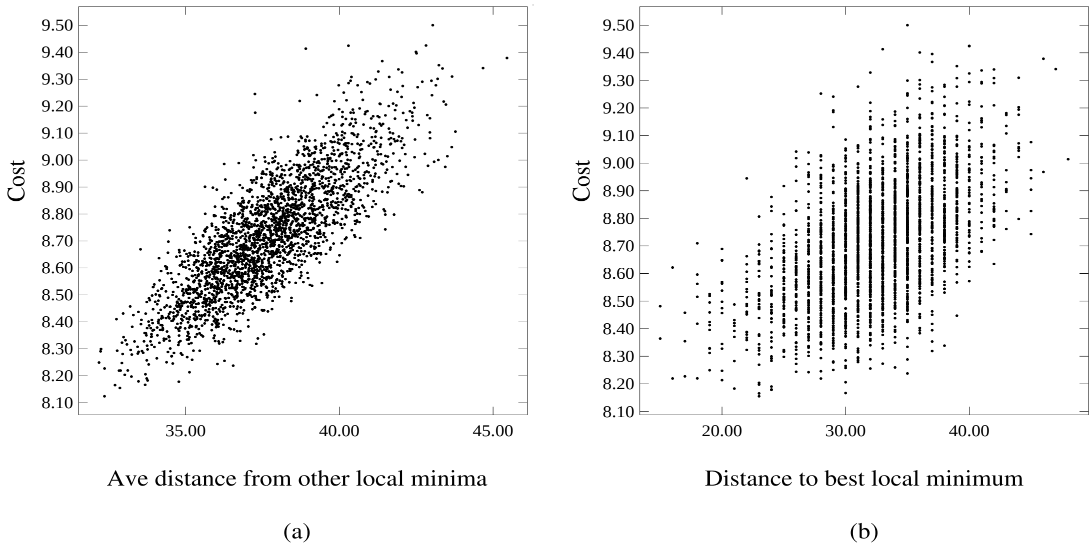
از نقاط بهینه محلی بدست آمده تا حال حاضر، برای پیشنهاد هوشمندانه یک نقطه شروع (برای الگوریتم جستجوی محلی) استفاده می‌کند. در این بخش علاقمندیم شهودی از شکل (ساختار) سطح هزینه مسائل ترکیبیاتی بدست آوریم که در اینصورت منجر به انتخاب خوب یک ساختار تخمین‌زننده برای تابع ارزیابی خواهد شد. در حقیقت با مشاهده رابطهٔ نقاط بهینه محلی در ساختار سطح هزینه، می‌توانیم ساختاری برای تابع ارزیابی پیشنهاد کنیم و سپس با شبیه‌سازی مونت—کارلو و با بدست آوردن مسیرهای آموزشی، پارامترهای این ساختار را با یک الگوریتم یادگیری با ناظر (یا سرپرستی) تعیین کنیم.

Boese و دیگران این کار را برای مسائل TSP و تقسیم‌بندی گراف انجام دادند [۸]. آنها ۲۵۰۰ جواب بهینه محلی را بدست آوردند و سپس طبق تعریف عملگر همسایگی N، شکلهای جالب و با معنایی ترسیم نمودند. شکلهای ۴—۳، ۴—۵ و ۴—۶ هزینه یک جواب را در مقابل فاصله متوسط تا سایر جوابها (۲۴۹۹ جواب دیگر) و هزینه یک جواب را در مقابل فاصله تا بهترین جواب بهینه، به ترتیب برای مسائل TSP با ۱۰۰، TSP با ۵۰۰ شهر و تقسیم‌بندی گراف نشان می‌دهند.

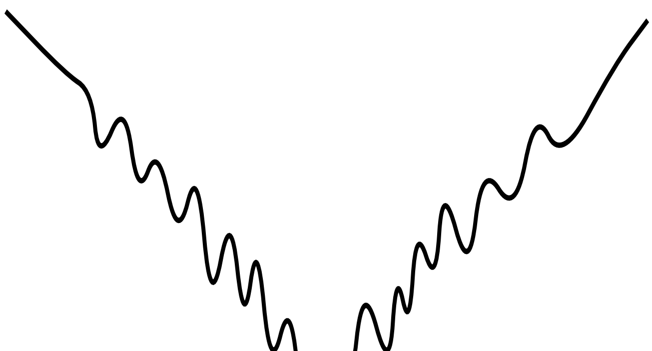
از این شکلها بر می‌آید که جوابهای بهینه محلی کم‌هزینه‌تر به بهینه سراسری نزدیکترند و بهینه سراسری به نوعی به مرکز مجموعه جوابهای بهینه محلی نزدیکتر می‌باشد که این باعث معرفی مفهوم دره بزرگ [۱۲] برای ساختار جوابهای بهینه محلی می‌شود (شکل ۴—۴).

یک راه استفاده از این اطلاعات، این است که در الگوریتم جستجوی تکراری، نقطه شروع جدید را نزدیک به میانگین نقاط بهینه محلی بدست آمده تا حال حاضر قرار دهیم. در عمل نیز سعی می‌کنیم ساختاری هموار و نزدیک به شکل دره بزرگ برای تابع ارزیابی پیشنهاد کنیم و سپس با

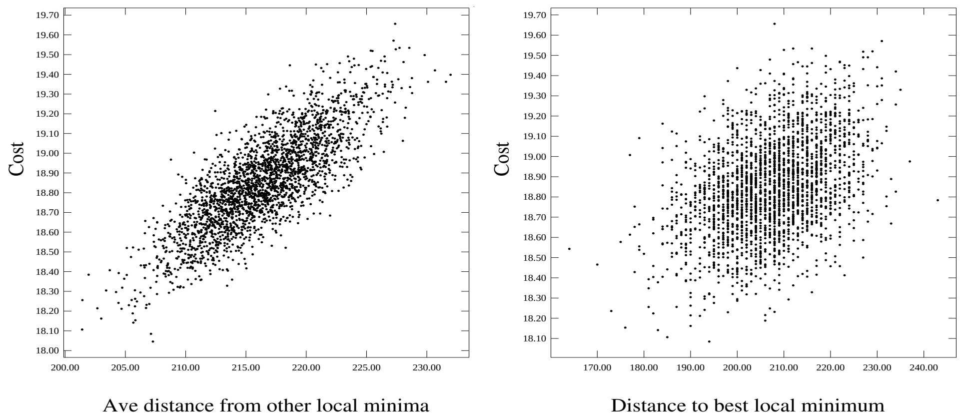
[۱۲] big valley



شکل ۴–۳: تحلیل ۲۵۰۰ بهینه محلی تصادفی برای یک نمونه از مسئله TSP با ۱۰۰ شهر. هزینه یک تور (محور عمودی) در مقابل (a) فاصله میانگین تا بقیه جوابها و (b) فاصله تا بهترین بهینه محلی رسم شده است.

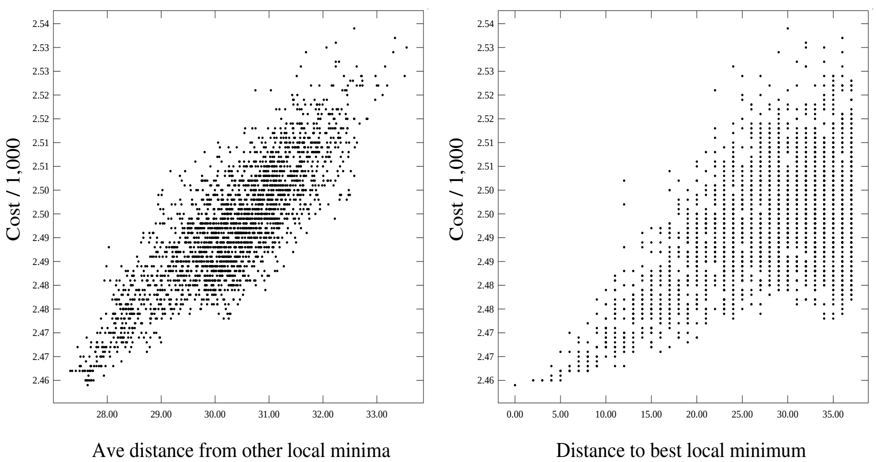


شکل ۴–۴: ساختار شهودی دره بزرگ برای تصور چگونگی پراکندگی بهینه‌های محلیِ یک مسئله ترکیبیاتی.



(a) (b)

شکل ۴–۵: تحلیل ۲۵۰۰ بهینه محلی تصادفی برای یک نمونه از مسئله TSP با ۵۰۰ شهر. هزینه یک تور (محور عمودی) در مقابل (a) فاصله میانگین تا بقیه جوابها و (b) فاصله تا بهترین بهینه محلی رسم شده است.



(a) (b)

شکل ۴–۶: تحلیل ۲۵۰۰ بهینه محلی تصادفی برای یک نمونه از مسئله تقسیم‌بندی گراف. هزینه یک تقسیم‌بندی (محور عمودی) در مقابل (a) فاصله میانگین تا بقیه جوابها و (b) فاصله تا بهترین بهینه محلی رسم شده است.

یک الگوریتم یادگیری با سرپرستی، پارامترهای این ساختار را مشخص کنیم (مثلاً استفاده از تابع چندجمله‌ای درجه دوم و سپس تنظیم ضرایب).

۴–۲–۱ ساختار سطح هزینه مسئله bin-packing

در این بخش سعی می‌کنیم بطور دقیق‌تر ساختار سطح هزینه مسئله bin-packing را مورد بررسی قرار دهیم. در ابتدا به تعریف یک ساختار همسایگی N و فاصله میان دو جواب $d(x_1, x_2)$ نیاز داریم. ساختار همسایگی همان است که در بخش ۴–۱ توضیح داده شد، یعنی دو جواب x_1 و x_2 همسایه هستند اگر با تعویض جای یک بسته از یک سطل به سطلی دیگر، از x_1 به x_2 برسیم. فاصله میان دو جواب $d(x_1, x_2)$ عبارتست از حداقل تعداد عملگرهای تعویض لازم برای تبدیل x_1 به x_2. چون محاسبهٔ $d(x_1, x_2)$ از لحاظ زمانی طولانی است، از فاصله محدود شده [13] میان دو جواب x_1 و x_2 برای تخمین $d(x_1, x_2)$ استفاده می‌کنیم:

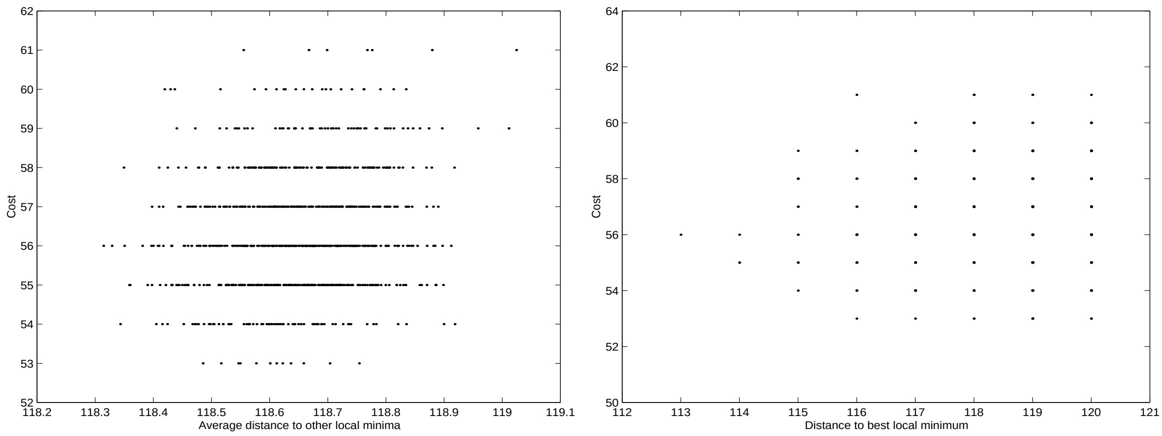
$$b(x_1, x_2) = num\_of\_bins - sim(x_1, x_2)$$

یعنی $b(x_1, x_2)$ عبارتست از اختلاف میان x_1 و x_2، یا بطور معادل تعداد کل سطلها منهای تشابه میان x_1 و x_2. تشابه میان دو جواب $sim(x_1, x_2)$ عبارتست از تعداد بسته‌هایی که در دو جواب در سطلهای یکسانی قرار گرفته‌اند.

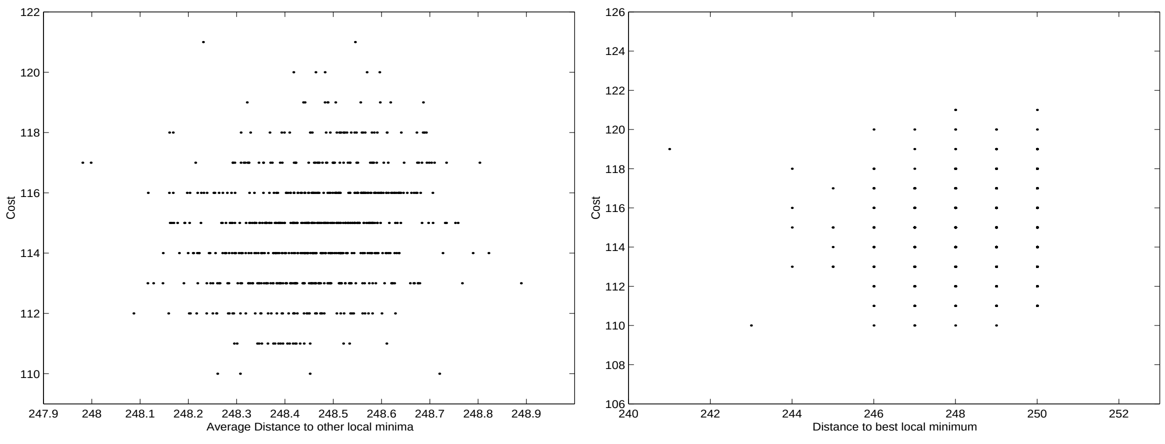
من برای دو نمونه از مسئله bin-packing یعنی u_250_00 و u_500_00 به تعداد ۴۰۰۱ جواب بهینه محلی را بدست آورده‌ام [14]؛ برای این منظور بصورت تصادفی ۴۰۰۱ نقطه شروع را انتخاب کرده‌ام و سپس با اجرای الگوریتم جستجوی محلی به ۴۰۰۱ جواب بهینه رسیده‌ام. شکلهای

[13] bounded distance

[14] تمام نمونه مسائل این تز از صفحه اینترنتی http://www.ms.ic.ac.uk/info.html برداشته شده‌اند.



شکل ۴—۷: پراکندگی ۴۰۰۱ نقطه بهینه محلی برای نمونه مسئله u_۲۵۰_۰۰. شکل سمت چپ فاصله متوسط هر نقطه تا بقیه نقاط (۴۰۰۰ نقطه دیگر) را در مقابل هزینه آن نقطه رسم می‌کند و شکل سمت راست فاصله هر نقطه تا بهترین جواب (یعنی جواب بهینه) را در مقابل هزینه رسم می‌کند.



شکل ۴—۸: پراکندگی ۴۰۰۱ نقطه بهینه محلی برای نمونه مسئله u_۵۰۰_۰۰. شکل سمت چپ فاصله متوسط هر نقطه تا بقیه نقاط (۴۰۰۰ نقطه دیگر) را در مقابل هزینه آن نقطه رسم می‌کند و شکل سمت راست فاصله هر نقطه تا بهترین جواب (یعنی جواب بهینه) را در مقابل هزینه رسم می‌کند.

۴—۷ و ۴—۸ فاصله متوسط یک جواب بهینه محلی تا بقیه جوابها و فاصله یک جواب بهینه محلی تا بهترین جواب را برای این نمونه مسئله‌ها نشان می‌دهد. همانطور که در این شکلها دیده می‌شود ساختار دره بزرگ برای مجموعه بهینه‌های محلی مسئله bin-packing تأیید می‌شود.

۴—۳ الگوریتم STAGE

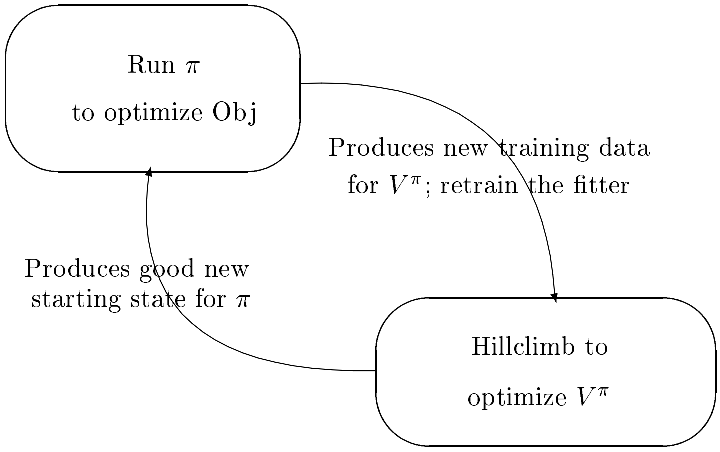
در این بخش الگوریتم STAGE برای مسائل ترکیبیاتی را مرور می‌کنیم. STAGE از یک ایده ساده استفاده می‌کند: کارایی یک الگوریتم جستجوی محلی به حالتی وابسته است که جستجو از آن آغاز شده است [۹]. می‌توان این مطلب را با نگاشتی از حالاتِ شروع x به نتایج مورد انتظار نشان داد:

$V^\pi(x) =$ بهترین نتیجه مورد انتظار از اجرای الگوریتم جستجوی محلی π با شروع از حالت x

که π یکی از الگوریتمهای جستجوی محلی مانند SA یا تپه‌نوردی می‌باشد. تابع ارزیابی یاد گرفته شده $V^\pi(F(x))$ نشان می‌دهد که حالت x به عنوان حالت شروع جستجوی محلی π به چه جوابی منتهی خواهد شد. برای پیدا کردن بهترین نقطه شروع باید V^π را روی مجموعه حالات X بهینه کنیم. این کار به سادگی با در نظر گرفتن V^π بجای Obj و با اعمال الگوریتم تپه‌نوردی تصادفی انجام می‌شود [۱۵].

الگوریتم STAGE چارچوبی برای یادگیری و استفاده از V^π روی یک نمونه مسئله فراهم می‌سازد. همانطور که در شکل ۴—۹ نشان داده شده است STAGE بین دو فازِ جستجوی محلی تغییر وضعیت می‌دهد: اجرای الگوریتم جستجوی اولیه π روی Obj و اجرای تپه‌نوردی روی V^π برای

[۱۵] توجه کنید که حتی اگر V^π نسبت به فضای مشخصه‌ها تابعی هموار باشد— مثلاً اگر V^π را با یک مدل ساده مثل چندجمله‌ای نشان دهیم— می‌تواند منجر به یک ساختار سطح پیچیده نسبت به ساختار همسایگی روی X شود.



شکل ۴—۹: نمودار حلقه اصلی در STAGE.

پیدا کردن نقاط شروع جدید خوب برای π. می‌توان STAGE را به عنوان یک رویکرد چندآغازه [۱۶] هوشمند در نظر گرفت.

توصیف الگوریتم در شکل ۴—۱۰ آورده شده است. در ادامه به شرح هر یک از پارامترهای X، S، π، Obj، $ObjBound$، F، Fit، N، Pat و $TotEvals$ می‌پردازیم:

- فضای جواب X.
- نقاط شروع $S \subset X$ (و یک روش برای تولید تصادفی نقاط S)
- π الگوریتم جستجوی محلی است که STAGE از آن یاد می‌گیرد. فرض می‌شود π دارای خاصیت مارکوف می‌باشد و مناسب [۱۷] است. در بخش بعد مفصل‌تر در این باره صحبت خواهیم کرد.

[۱۶] multi-start
[۱۷] proper

STAGE(X, S, π, Obj, ObjBound, F, Fit, N, Pat, TotEvals)

Given:

- a state space X
- starting states $S \subset X$ (and a method for generating a random state in X)
- a local search procedure π that is Markovian and proper
- an objective function, Obj: $X \to R$ to be minimized
- a lower bound on Obj, ObjBound $\in R$ (or -∞ if no bound is known)
- a featurizer F mapping states to real-valued features, F: $X \to R^D$
- a function approximator Fit
- a neighborhood structure N: $X \to 2^X$ and patience parameter Pat for running stochastic hill-climbing on V^π
- TotEvals, the number of states evaluations allotted for this run

1: **Initialize** the function approximator; let $x_0 \in S$ be a random starting state for search.
2: **Loop** until number of states evaluated exceeds TotEvals.

1. **Optimize Obj using** π. From x_0 run search algorithm π, producing a search trajectory $(x_0, x_1, \ldots, x_T)$.
2. **Train** V^π. For each point x_i on the search trajectory, define $y_i := \min_{j=i\ldots T} Obj(x_j)$, and add the pair $(F(x_i) \to y_i)$ to the training set for Fit. Retrain Fit, call the the resulting learned evaluation function V^π.
3. **Optimize** V^π **using hill-climbing.** Continuing from x_T, optimize $V^\pi(F(x))$ by performing a stochastic hill-climbing search over the neighborhood structure N. Cut off the search when either Pat consecutive moves produce no improvement or a candidate state z_{t_1} is predicted to be impossibly good, i.e. $V^\pi(z_{F(t+1)}) < ObjBound$. Denote this search trajectory by $(z_0, z_1, \ldots, z_t)$.
4. **Set smart restart state.** Set $x_0 = z_t$. But in the event that the V^π hill-climbing search accepted no moves (i.e. $z_t = x_T$) then reset x_0 to a new random starting state.

3: **Return** the best state found.

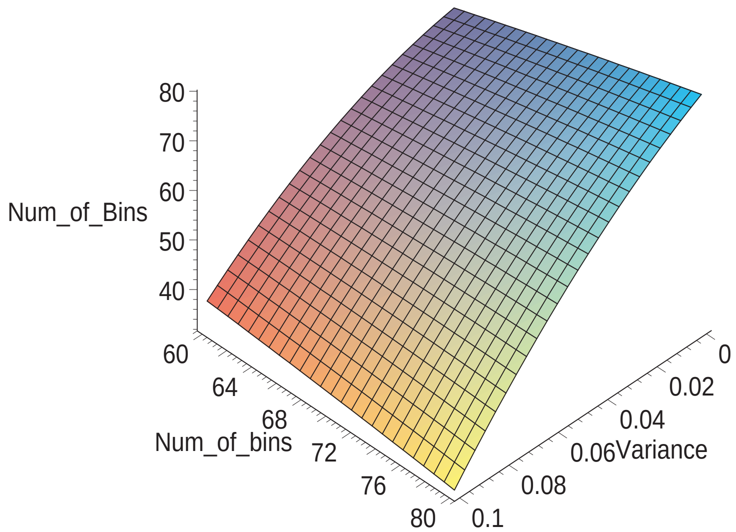
شکل ۴—۱۰: الگوریتم STAGE.

- تابع هدف $Obj : X \to R$ که باید (در اینجا) کمینه شود.
- $ObjBound$ یک کران پایین برای Obj است (یا $-\infty$ اگر این کران ناشناخته باشد).
- F (نگاشت حالت به مشخصه) که هر حالت را به یک بردار با مقادیر حقیقی نگاشت می‌کند: $F : X \to R^D$.
- Fit که یک تخمین‌زننده تابع است.
- ساختار همسایگی $N : X \to ۲^X$ و پارامتر Pat برای اجرای الگوریتم تپه‌نوردی تصادفی روی V^π.
- $TotEvals$ حداکثر تعداد ارزیابی‌های مجاز است.

خروجی STAGE یک حالت X است که دارای کمترین مقدار Obj بین حالات بررسی شده در طول اجراست. همچنین تابع ارزیابی یادگرفته‌شده نهایی V^π نیز تولید می‌شود که بررسی آن منجر به فهم این نکته می‌شود که چه ترکیبی از مشخصه‌ها باعث رسیدن به کارایی بالا می‌شوند، در شکل ۴—۱۱ می‌توانید تابع ارزیابی یاد گرفته شده توسط STAGE برای نمونه مسئله U250_00 را ملاحظه کنید. برای اطلاعات بیشتر می‌توانید به [۹] مراجعه کنید.

۴—۳—۱ مباحث نظری STAGE

در این بخش به ذکر نکاتی در مورد چگونگی انتخاب π و Fit می‌پردازیم. اجرای یک الگوریتم جستجوی محلی π با شروع از $x_0 \in X$، بصورت تصادفی دنباله‌ای از حالات را تولید می‌کند



شکل ۴–۱۱: نمونه‌ای از یک تابع ارزیابی که توسط STAGE یاد گرفته شده است.

$T = (x_0^T, x_1^T, \ldots)$ که مسیر [۱۸] نام دارد. در حالت کلی مسیر می‌تواند محدود یا نامحدود باشد، یعنی پس از تعداد مشخصی گام خاتمه پذیرد. برای هر مسیر خاتمه‌پذیر، برای هر $i > |T|$، x_i^T را معادل با یک حالت ویژه End در نظر می‌گیریم.

فرض کنید Γ مجموعه تمام مسیرهای ممکن روی X باشد؛ π هر مسیر را با یک احتمال مشخص تولید می‌کند و توسط همین احتمالات مشخص می‌شود.

تعریف ۱. روال جستجوی محلی یک تابع تصادفی است $\pi : X \to \Gamma$. با داشتن نقطه شروع $x \in X$ توزیع احتمالی که توسط آن π، $T \in \Gamma$ را تولید می‌کند با $P(T|x_0^T = x)$ نشان می‌دهیم.

تعریف ۲. روال جستجوی محلی π را مناسب [۱۹] گوییم اگر از هر نقطه شروع $x \in X$ نهایتاً بتوان با احتمال ۱ به حالت پایانی End رسید.

---
[۱۸] trajectory
[۱۹] proper

تعریف ۳. روال جستجوی محلی π دارای خاصیت مارکوف است اگر احتمال گذر از x_i به x_{i+1} به حالات دیده شدهٔ قبل از x_i بستگی نداشته باشد:

$$P(x_{i+1}^T = x_{i+1} | x_0^T = x_0, x_1^T = x_1, \ldots, x_i^T = x_i) = P(x_{i+1} | x_i)$$

تعریف ۴. روال جستجوی محلی π نسبت به Obj یکنوا است اگر برای هر مسیر $T \in \Gamma$ که π تعریف می‌کند، داشته باشیم:

$$Obj(x_0^T) \geq Obj(x_1^T) \geq \cdots$$

تعریف ۵. برای هر روال جستجوی محلی π و تابع هدف $Obj : X \to R$، تابع $V^\pi : X \to R$ را بصورت زیر تعریف می‌کنیم:

$$V^\pi = E[inf[Obj(x_k^T) | k = 0, 1, \cdots]]$$

که مقدار امید ریاضی روی توزیع مسیر $P(T | x_0 = x)$ محاسبه می‌شود.

تحت شرط زیر نشان می‌دهیم $V^\pi(x)$ تعریف شده است، یعنی مقدار عبارت تعریف ۵ وجود دارد.

قضیه ۱ . اگر Obj از طرف پایین کراندار باشد آنگاه $V^\pi(x)$ در هر حالت $x \in X$ و برای هر سیاست π تعریف شده است [۹].

STAGE تحمیل می‌کند که π دارای خاصیت مارکوف باشد. علت این است که STAGE می‌تواند داده‌های آموزشی زیادی را از یک مسیر نمونه استخراج کند چون هر نقطه (یا حالت) میانی

در مسیر را می‌توان به عنوان حالت اولیه در نظر گرفت و داده‌های آموزشی جدیدی بدست آورد. نکته مهم این است که در هر مسیر $T = (x_0, x_1, \ldots, x_i, x_{i+1}, \ldots)$ هر دنباله $(x_i, x_{i+1}, \ldots)$ با همان احتمالی تولید می‌شود که گویی π از x_i شروع شده است. اگر π یکنوا نیز باشد آنگاه تعریف V^π را بصورت جمع نیز می‌توان بیان کرد:

$$V^\pi = E[inf[Obj(x_k^t|k = 0, 1, 2, \ldots]]$$

$$= E[\lim_{k\to\infty} Obj(x_k^T)] \quad \text{چون } \pi \text{ یکنوا است}$$

$$= E[\sum_{k=0}^{\infty} R(x_k^T, x_{k+1}^T)] \quad \text{چون } \pi \text{ مناسب است}$$

که تمام میانگینها روی توزیع مسیر $P(T|x_0^T = x)$ گرفته می‌شود و تابع هزینه R توسط عبارت زیر تعریف می‌شود:

$$R(x, x') = \left\{ \begin{array}{ll} Obj(x) & if x \neq END, and x' = END. \\ 0 & otherwise. \end{array} \right\}$$

نوشتن V^π به این صورت، مشخص می‌کند که این دقیقاً همان «تابع مقدار سیاست» در زنجیره مارکوف (X, π, R) می‌باشد و به این معناست که V^π در معادلات Bellman برای پیش‌بینی (فرمول ۲ از فصل ۳) صدق می‌کند و لذا تمام الگوریتمهای یادگیری تقویتی قابل اِعمال هستند. بویژه با استفاده از روش یادگیری $TD(\lambda)$، STAGE می‌تواند یک ساختار خطی را، بدون این که مسیری را در حافظه ذخیره نماید، برای تخمین V^π یاد بگیرد.

STAGE بر مبنای تخمین‌زننده Fit، تابع ارزیابی V^π را از داده‌های آموزشی تولید می‌کند. مثالهایی از تخمین‌زننده Fit عبارتند از: شبکه‌های CMAC، شبکه چند لایه پرسپترون، توابع پایه شعاعی و چندجمله‌ای‌ها. Boyan در [۹] شرایطی را برای کارا بودن Fit بیان می‌کند و نهایتاً به این

نتیجه می‌رسد که برای بدست آوردن حداکثر کارایی، تخمین‌زننده Fit باید دارای ساختار خطی باشد. در فصل بعد اثر الگوریتم‌های جستجوی محلی مختلف و توابع تخمین‌زننده مختلف گوناگون را بر کارایی STAGE مورد بررسی قرار می‌دهیم.

۴—۴ بحث

در این فصل دیدیم که چگونه می‌توان فرایند بهینه‌سازی یک مسئله ترکیبیاتی را بصورت جستجو در یک فضای حالات در نظر گرفت. بعنوان یک راه حل، جستجوهای محلی را برای یافتن جواب معرفی کردیم ولی دیدیم که با توجه به قضیه فاجعه حد مرکزی، با اجرای چندبارهٔ یک الگوریتم جستجوی محلی بصورت کورکورانه، نمی‌توان انتظار داشت که به جوابهای خیلی خوبی برسیم. به همین دلیل، به جستجوهای مطلع [۲۰] روی آوردیم که از ابتکار، برای راهنمایی فرایند جستجو به نواحی امیدوارکننده استفاده می‌کنند. بر این اساس الگوریتم STAGE را مرور کردیم و دیدیم که STAGE از مشخصه‌های مختلف حالات (علاوه بر تابع هدف) استفاده می‌کند تا با یک تابع ارزیابی — که با مسیرهای بدست آمده از شبیه‌سازی مونت—کارلو آموزش داده شده است — خوب بودنِ حالات را به عنوان نقطه اولیه جستجو تخمین می‌زند. همچنین ساختار سطح هزینه مسائل ترکیبیاتی را بررسی کردیم و مشاهده کردیم که ساختار شهودیِ دره بزرگ را می‌توان برای چگونگی پراکندگی بهینه‌های محلی (در فضای مشخصه‌ها) در نظر گرفت؛ من نتیجه کار Boese برای مسائل TSP و تقسیم‌بندی گراف را در مورد مسئله bin-packing تحقیق نمودم. در انتها نیز کمی به مباحث نظری STAGE پرداختیم و دیدیم که بدست آوردن مسیر به بهینه سراسری در فضای حالات، هم‌ارزِ پیدا کردن سیاست بهینه در زنجیره مارکوف می‌باشد. در فصل بعد، اثر الگوریتمهای جستجوی محلی مختلف و ساختارهای تخمین‌زننده

[۲۰] informed

گوناگون را بر کارایی STAGE مورد بررسی قرار می‌دهیم.

فصل ۵

تحلیل STAGE و مطالعه ساختار سطح هزینه در مسائل بهینه‌سازی

در فصل پیش STAGE را دیدیم و آن را برای حل مسئله bin packing به کار بردیم. در الگوریتم STAGE پارامترهایی وجود دارد که می‌توان آنها را تغییر داد و سپس اثر آنها را بر عملکرد STAGE مشاهده نمود. از جمله پارامترهای قابل تغییر، الگوریتم جستجوی محلی و ساختار استفاده شده برای تابع ارزیابی (یاد گرفته شده) می‌باشد. در این فصل اثر استفاده از الگوریتم‌های جستجوی محلیِ تپه‌نوردی با بیشترین نزول [۱]، تپه‌نوردی با اولین نزول [۲] و تپه‌نوردیِ تصادفی [۳] را بر کارایی STAGE مورد بررسی قرار خواهیم داد و بر مبنای نتایج حاصل، مشخص می‌کنیم که کدام یک همراه با STAGE منجر به بدست آمدن جوابهای دقیق‌تری برای مسئله bin packing خواهد شد. سپس به بررسی استفاده از ساختارهای مختلف یادگیری در STAGE می‌پردازیم و اثر استفاده از شبکه‌های

[۱] best improvement hill climbing

[۲] first improvement hill climbing

[۳] stochastic hill climbing

CMAC، توابع چندجمله‌ای، توابع نمایی و توابع لگاریتمی را خواهیم دید [۲۱].

تا کنون ادعاهای متفاوتی در مورد چگونگی ساختار سطح هزینه مسائل بهینه‌سازی ارائه شده است. ما نمونه‌ای از این ادعا یعنی ساختار دره بزرگ را در فصل ۴ برای مسائل بهینه‌سازی ترکیبیاتی دیدیم. در این بخش ساختار ادعا شده برای بعضی مسائل دیگر دامنه‌ها را تحت عنوان چشم‌انداز شایستگی[۴] خواهیم دید و سپس به تکمیل مباحث مطرح شده در فصل ۴ دررابطه با ساختار سطح هزینه مسائل ترکیبیاتی می‌پردازیم. پس از آن ساختار سطح هزینه مسئله تنظیم صفحه دَرهم[۵] (BBS) را مدنظر قرار خواهیم داد.

بویان مسئله BBS را به عنوان مسئله‌ای که STAGE در آن ضعیف عمل می‌کند، مطرح ساخته است[۹]؛ نتایج بدست آمده از اجرای STAGE روی این مسئله نشان دهندهٔ ضعف عملکرد آن در مقایسه با الگوریتم‌های تپه‌نوردی و SA می‌باشد [۹]. اگر بتوانیم مشابه تحلیلی که برای سطح هزینه مسئله bin packing داشتیم، برای مطالعه سطح هزینه مسئله BBS نیز داشته باشیم آنگاه نتایج، تحلیل ما را از کارایی STAGE تکمیل خواهند کرد. به این نحو که ما قبلاً ادعا کردیم علت عملکرد خوب STAGE در حل مسائل بهینه‌سازی ترکیبیاتی این است که ساختار دره بزرگ بر بهینه‌های محلی این مسائل حاکم است و STAGE از این اطلاع در بدست آوردن یک تابع ارزیابی مناسب استفاده می‌کند، ما این موضوع را برای مسئله bin packing در فصل ۴ تحقیق نمودیم. اگر تحقیقات در مورد ساختار سطح هزینه مسئله BBS مؤید این موضوع باشد که ساختار دره بزرگ برای بهینه‌های محلی آن درست نمی‌باشد آنگاه ما دلیلی دیگر برای درستی ادعای خود در مورد STAGE ارائه داده‌ایم.

---

[۴] این اصطلاح معمولاً در مبحث الگوریتم‌های تکاملی بکار می‌رود و معادل با همان اصطلاح ساختار سطح هزینه می‌باشد.

[5] boogle board setup

۵—۱ اثر الگوریتم‌های مختلف جستجوی محلی

در فصل ۲ الگوریتم‌های مختلف جستجو را دیدیم و تپه‌نوردی یکی از الگوریتم‌هایی بود که مرور کردیم. الگوریتم‌های جستجوی محلی که در این بخش خواهیم دید، گونه‌های مختلف این الگوریتمِ پایه می‌باشند که عبارتند از: تپه‌نوردی با بیشترین نزول، تپه‌نوردی با اولین نزول و تپه‌نوردیِ تصادفی. در تپه‌نوردی با اولین نزول، همسایه‌های یک حالت بطور سیستماتیک تولید و بررسی می‌شوند و اولین وضعیتی که بهبودی در تابع هدف ایجاد کند، پذیرفته خواهد شد. در تپه‌نوردی با بیشترین نزول، تمام همسایه‌های یک حالت بررسی می‌شوند و سپس حالتی انتخاب می‌شود که بیشترین تنزّل را در تابع هدف ایجاد نماید. ولی معمولاً چون بررسی همهٔ همسایه‌های یک حالت کاری زمان‌گیر است، از الگوریتم تپه‌نوردیِ تصادفی استفاده می‌شود. در این الگوریتم، تعداد محدودی از همسایه‌های یک حالت بطور تصادفی انتخاب می‌شوند [6] و سپس حالتی انتخاب می‌شود که بهبودی در تابع هدف ایجاد نماید. علامت اختصاری و نام الگوریتم‌ها در جدول ۵—۱ آورده شده است. در تمامی این الگوریتم‌ها، اگر به وضعیتی برسیم که مقدار تابع هدف در آن از مقدار تابع هدف در همهٔ همسایه‌ها کمتر باشد آنگاه این وضعیت، نتیجهٔ جستجوی محلی و بهینهٔ محلی نامیده می‌شود.

Algorithm	Description
Fihc	First improvement hill climbing
Sdhc	Steepest descent hill climbing
Sthc	Stochastic hill climbing
	Patience = 250

جدول ۵—۱: خلاصه‌ای از الگوریتم‌های جستجوی محلی مختلف و اختصار نام آنها.

اِعمال الگوریتم جستجوی محلی در دو نقطه از STAGE امکانپذیر است: در بهینه‌سازی تابع ارزیابی و در بهینه‌سازی تابع هدف اصلی. در دو بخش بعد الگوریتم‌های مختلف جستجوی محلی را

[6] که تعداد این همسایه‌های تصادفی با پارامتر Patience مشخص می‌شود.

به ترتیب در بهینه‌سازی تابع ارزیابی و بهینه‌سازی تابع هدف اصلی بکار می‌بریم و اثر آنها را در کارایی STAGE با هم مقایسه می‌کنیم.

۵—۱—۱ جستجوی محلی در بهینه‌سازی تابع ارزیابی

در این بخش، نتایج حاصل از اجرای الگوریتم STAGE همراه با هر کدام از الگوریتم‌های جستجوی محلی جدول ۵—۱ برای بهینه‌سازی تایع ارزیابی را ارائه کرده و سپس الگوریتم‌ها را مقایسه می‌کنیم. مقایسهٔ الگوریتم‌ها بر مبنای دقت جوابهای بدست آمده در یک «زمان محدود» می‌باشد. در STAGE می‌توان حداکثرِ تعداد حالاتِ مشاهده شده را با پارامتر TotalEvals کنترل نمود؛ در اجرای الگوریتم جستجوی محلی روی تابع هدف و اجرای الگوریتم جستجوی محلی روی تابع ارزیابی، بعضی حالاتِ مشاهده شده رد و بعضی قبول می‌شوند، TotalEvals شامل مجموع این حالات است.

در آزمایشات این بخش، الگوریتم‌ها با پارامتر TotalEvals یکسان روی نمونه مسائل اجرا می‌شوند و سپس دقت نتایج بدست آمده مقایسه خواهند شد؛ به اینصورت زمان اجرا برای همهٔ الگوریتمها تقریباً یکسان خواهد بود و چون معیار، تعداد حالاتِ مشاهده شده است، وابستگی نتایج به سرعت کامپیوتر استفاده شده نیز از بین خواهد رفت.

STAGE یک الگوریتم تصادفی است (چون از Sthc برای بهینه‌سازی تابع هدف استفاده می‌کند)، بنابراین برای تحلیل نتایجِ آن، باید به دفعات اجرا شود. فرض کنید می‌خواهیم اثر استفاده از Fihc و Sthc در STAGE را با هم مقایسه کنیم، عادلانه نیست که با یک بار اجرای هر دو الگوریتم روی یک نمونه مسئله، نتایج را مقایسه کنیم چون نتایج به دنبالهٔ اعداد تصادفیِ تولید شده وابسته هستند. روش ما این است که هر دو الگوریتم را ۳۰ بار روی نمونه مسئله اجرا می‌کنیم و سپس بهترین

جواب، بدترین جواب، میانگین و فاصله اطمینان (معمولاً ۹۰ درصد) برای میانگین دو مجموعه جواب را با هم مقایسه می‌کنیم.

فرض کنید $A = \{a_1, a_2, \ldots, a_n\}$ جوابهای بدست آمده از n بار اجرای الگوریتم باشند، چون جوابهای بدست آمده از هم مستقل هستند لذا ما با ۳۰ نمونه تصادفیِ مستقل از هم سر و کار داریم، بنابراین از توزیع نرمال برای بدست آوردن فاصله اطمینان استفاده می‌کنیم. میانگین و واریانس به ترتیب عبارتند از: $\mu = \frac{\sum_{i=1}^{n} a_i}{a_n}$ و $S = \frac{\sum_{i=1}^{n} (a_i-\mu)^2}{n-1}$، حال عبارت $\mu \pm Z_{\frac{\alpha}{2}}.\frac{S}{\sqrt{n}}$ فاصلهٔ اطمینان $1-\alpha$ درصدی را مشخص می‌کند.

در آزمایشات این بخش، الگوریتم‌های جستجوی محلی مختلف در بهینه‌سازی تابع ارزیابی بکار رفته است و از Sthc با پارامتر Patience=250 برای بهینه‌سازی تابع هدف استفاده شده است. نتایج اجرای STAGE روی نمونه مسئله‌های U500 که شامل ۵۰۰ سطل با ظرفیت ۱۵۰ هستند در جدول ۵—۲ آورده شده است[۷] که شامل میانگین با فاصله اطمینان ۹۰ درصد، بهترین و بدترین جواب بدست آمده از ۳۰ اجرای الگوریتم‌ها روی هر نمونه مسئله است. . همچنین جدول ۵—۳ نتایج اجرای STAGE روی نمونه مسائل U250 را نشان می‌دهد که شامل ۲۵۰ سطل با ظرفیت ۱۵۰ می‌باشد. مقدار TotalEvals در همهٔ این آزمایشات ۵۰۰۰۰۰ می‌باشد.

همانطور که در جدول‌های ۵—۳ و ۵—۲ می‌توان دید، الگوریتم Fihc از بقیه الگوریتم‌ها بهتر عمل کرده است، علت این امر این است که این الگوریتم تا آنجا که می‌تواند فضای جستجو را مورد کاوش قرار می‌دهد ضمن این که با توجه به مشاهدات محدودِ خود در هر زمان، سعی می‌کند از اطلاعات بدست آمده برای انتخاب مسیر صحیح استفاده کند، به عبارت بهتر سعی می‌کند مصالحه [۸]

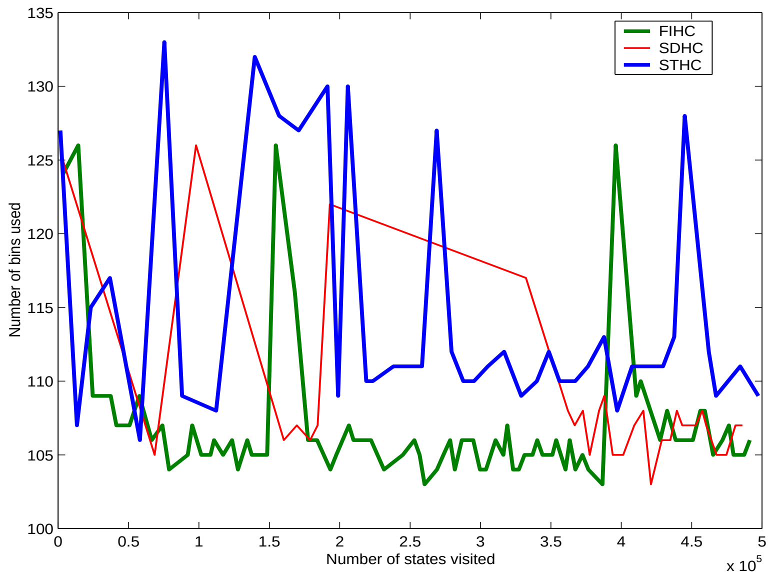
[۷] تمام نمونه مسائل این تز از صفحه اینترنتی http://www.ms.ic.ac.uk/info.html برداشته شده‌اند.
[۸] trade off

Instance	Algorithm	Mean ±	Tolerance	Best	Worst
U500_00	Fihc	208.80 ±	0.472	207	211
	Sdhc	214.50 ±	3.021	209	244
	Sthc	212.80 ±	0.859	209	216
U500_01	Fihc	211.80 ±	0.505	210	215
	Sdhc	219.64 ±	5.402	211	260
	Sthc	215.50 ±	0.735	211	218
U500_02	Fihc	212.00 ±	0.439	211	215
	Sdhc	223.05 ±	6.475	212	260
	Sthc	216.10 ±	0.900	210	219
U500_03	Fihc	214.89 ±	0.439	212	216
	Sdhc	225.50 ±	7.371	215	273
	Sthc	218.35 ±	0.518	215	220
U500_04	Fihc	216.05 ±	0.670	213	221
	Sdhc	231.00 ±	8.251	216	271
	Sthc	220.10 ±	0.637	218	224

جدول ۵–۲: نتایج حاصل حل نمونه مسئله‌های مختلف توسط STAGE با الگوریتم‌های جستجوی محلی مختلف. هر الگوریتم برای هر نمونه از مسئله ۳۰ بار اجرا شده است.

Instance	Algorithm	Mean ±	Tolerance	Best	Worst
U250_00	Fihc	102.80 ±	0.312	101	104
	Sdhc	105.55 ±	1.529	103	121
	Sthc	104.80 ±	0.312	103	106
U250_01	Fihc	103.44 ±	0.279	103	105
	Sdhc	105.05 ±	0.571	103	110
	Sthc	105.15 ±	0.275	104	106
U250_02	Fihc	105.55 ±	0.246	105	107
	Sdhc	107.09 ±	0.476	105	109
	Sthc	107.44 ±	0.337	106	109
U250_03	Fihc	103.59 ±	0.205	103	104
	Sdhc	105.19 ±	0.250	104	106
	Sthc	105.30 ±	0.328	104	107
U250_04	Fihc	105.09 ±	0.263	104	106
	Sdhc	106.30 ±	0.481	105	110
	Sthc	106.44 ±	0.246	105	107

جدول ۵–۳: نتایج حاصل حل نمونه مسئله‌های مختلف توسط STAGE با الگوریتم‌های جستجوی محلی مختلف. هر الگوریتم برای هر نمونه از مسئله ۳۰ بار اجرا شده است.



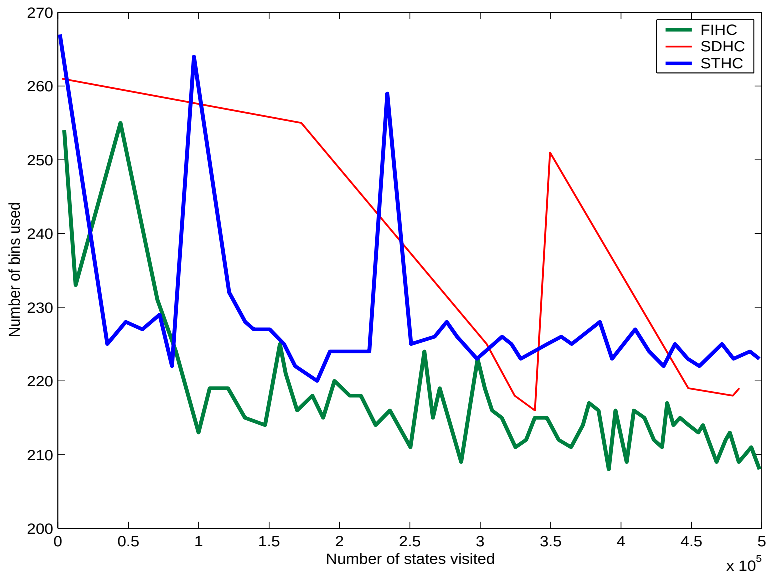
شکل ۵–۱: اثر استفاده از الگوریتم‌های جستجوی محلی مختلف در STAGE برای حل نمونه مسئله U250_00.

معقولی بین کاوش [۹] و تقیّد [۱۰] برقرار نماید. بعد از Fihc، الگوریتم Sthc قرار دارد، این الگوریتم (نسبت به Fihc) قسمت بیشتری از فضای جستجو را می‌کاود اما از اطلاعات بدست آمده (از مشاهدهٔ حالات)، کمتر استفاده می‌کند. در انتها نیز الگوریتم Bihc قرار دارد، زیرا این الگوریتم به شدّت خود رابه یک شاخهٔ جستجو مقیّد می‌کند و زمان زیادی از وقت خود را صرف بررسی حالاتِ یک شاخه می‌نماید؛ به عبارتِ بهتر، کاوشِ فضای جستجو را به خوبی انجام نمی‌دهد. در شکل‌های ۵–۱ و ۵–۲ چگونگی کاوش فضای جستجو توسط الگوریتم‌های جستجوی محلی مختلف همراه با STAGE نشان داده شده است.

---

[۹] exploration

[۱۰] exploitation



شکل ۵–۲: اثر استفاده از الگوریتم‌های جستجوی محلی مختلف در STAGE برای حل نمونه مسئله U500_00.

۵–۱–۲ جستجوی محلی در بهینه‌سازی تابع هدف

در این بخش، الگوریتم‌های مختلف جستجوی محلی را در بهینه‌سازی تابع هدف بررسی می‌کنیم و آنها را با هم مقایسه می‌کنیم. مقایسه الگوریتم‌ها بر اساس نتایج بدست آمده از اجرای متعدد آنها (۳۰ بار) می‌باشد. برای مقایسه الگوریتم‌ها دو حالت در نظر گرفته‌ایم: اجرای آنها در زمان محدود و اجرای آنها در تعداد گام محدود. برای کنترل این که زمان اجرا برای همه یکسان باشد، مشابه با بخش قبل، تعداد حالات دیده شده (شامل حالات پذیرفته شده و رد شده) را برای همه مساوی 2,000,000 در نظر گرفته‌ایم. در اجرا با تعداد گام یکسان، فقط تعداد حالات پذیرفته شده را در نظر گرفته‌ایم و در واقع تعداد گام‌هایی را که هر الگوریتم برای رسیدن به جواب بر می‌دارد یکسان (مساوی ۴۰۰) فرض کرده‌ایم. به این طریق می‌توان دید بهتری نسبت به عملکرد الگوریتم‌ها داشت، بطور مثال در Sdhc همهٔ حالات همسایه بررسی می‌شوند ولی تنها یک گام برداشته می‌شود (یعنی فقط یک حالت

Algorithm	Description
Fihc	First improvement hill climbing
Sdhc	Steepest descent hill climbing
Sthc1	Stochastic hill climbing, Patience = 350
Sthc2	Stochastic hill climbing, Patience = 700

جدول ۵—۴: خلاصه‌ای از الگوریتم‌های جستجوی محلی مختلف و اختصار نام آنها.

پذیرفته می‌شود) و اگر معیار فقط تعداد حالات دیده شده باشد مقایسهٔ آن با Sthc (که به مراتب تعداد همسایه‌های بسیار کمتری را برای برداشتن گام بررسی می‌کند) عادلانه نخواهد بود.

الگوریتم‌های مختلف جستجوی محلی که در بهینه‌سازی تابع هدف از آنها استفاده شده است در جدول ۵—۴ آورده شده است، لازم به ذکر است که در همهٔ آنها از Sthc با پارامتر Patience=250 برای بهینه‌سازی تابع ارزیابی استفاده شده است. نتایج حاصل از ۳۰ اجرای الگوریتم‌ها روی نمونه مسائل U250 که شامل ۲۵۰ سطل با ظرفیت ۱۵۰ می‌باشد در جدول‌های ۵—۵ و ۵—٦ آورده شده است که شامل میانگین با فاصله اطمینان ۹۵ درصد، بهترین و بدترین جواب می‌باشند. دقت کنید که در جدول ۵—۵ معیار خاتمهٔ الگوریتم‌ها تعداد حالات دیده شده می‌باشد (TotalEvals=2,000,000) و در جدول ۵—٦ معیار خاتمه الگوریتم‌ها تعداد گام‌های برداشته شده (400) می‌باشد.

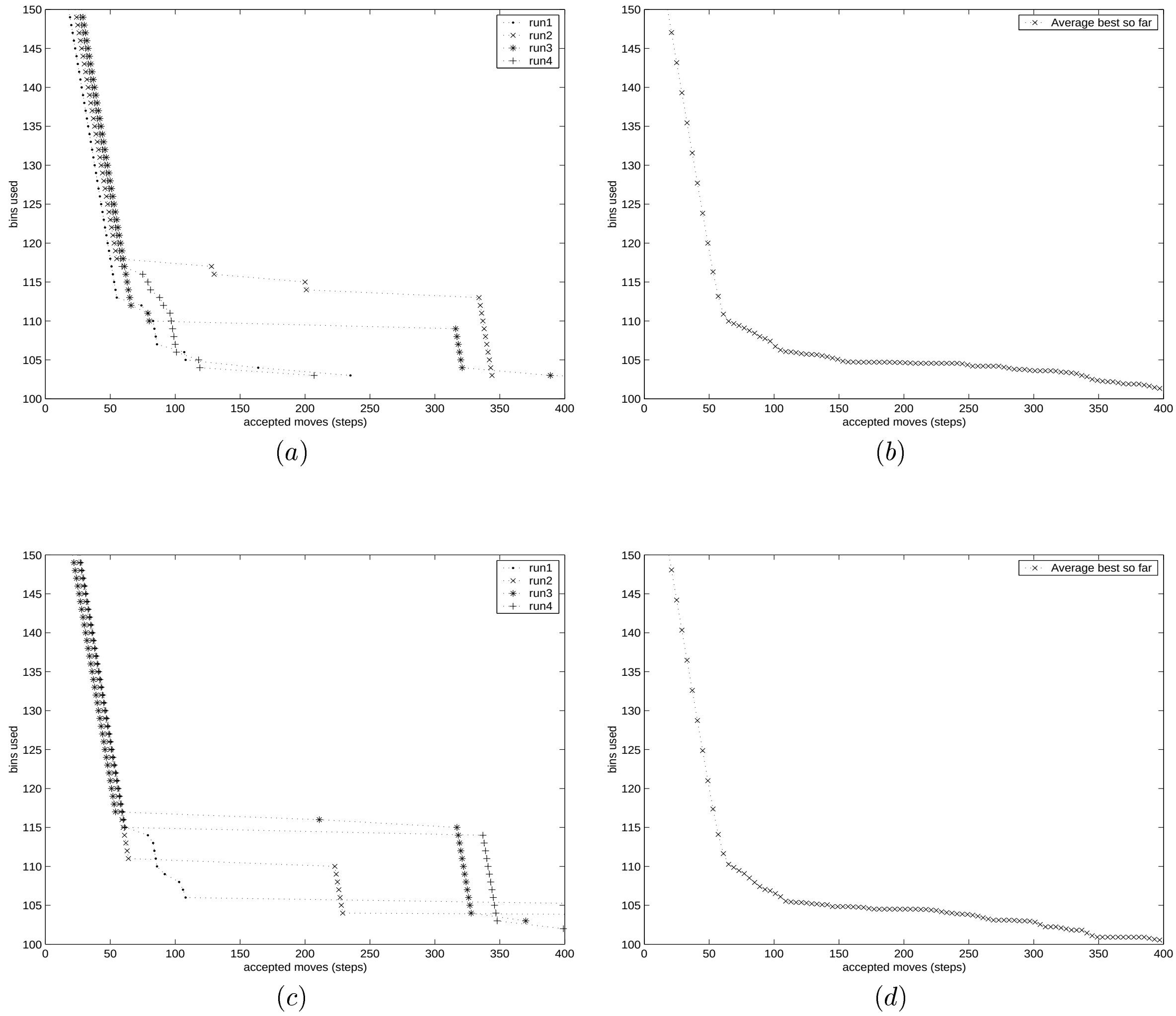
شکل‌های ۵—۳ و ۵—۴ عملکرد الگوریتم‌ها در حل نمونه مسئله U250_00 با محدودیت روی حالات دیده شده را رسم می‌کند. در شکل ۵—۳، نمودار (a) چهار اجرای الگوریتم Sdhc را روی این نمونه مسئله نشان می‌دهد و نمودار (b) عملکرد ۳۰ اجرای الگوریتم Sdhc را نشان می‌دهد، به این طریق که هر نقطه میانگین ۳۰ اجرای الگوریتم است؛ نمودارهای (c) و (d) همین نمودارها را برای الگوریتم Fihc نشان می‌دهند. همانطور که مشاهده می‌کنید عملکرد این دو الگوریتم بسیار شبیه به هم است (به شکل ۵—۵ که همین نمودارها را برای نمونه مسئله U250_09 رسم می‌کند، دقت کنید).

Instance	Algorithm	Mean ±	Tolerance	Best	Worst
U250_00	Fihc	109.27 ±	1.837	102	117
	Sdhc	124.40 ±	1.765	117	138
	Sthc	105.77 ±	0.328	104	107
U250_01	Fihc	111.00 ±	2.081	103	121
	Sdhc	124.90 ±	1.597	118	135
	Sthc	106.07 ±	0.439	104	108
U250_02	Fihc	110.40 ±	1.757	106	121
	Sdhc	124.50 ±	1.540	116	135
	Sthc	108.33 ±	0.388	107	111
U250_03	Fihc	109.17 ±	1.686	104	118
	Sdhc	125.60 ±	1.509	118	134
	Sthc	106.47 ±	0.523	104	109
U250_04	Fihc	110.60 ±	1.912	105	119
	Sdhc	124.77 ±	1.666	117	135
	Sthc	107.37 ±	0.365	106	109
U250_05	Fihc	109.67 ±	1.654	105	119
	Sdhc	125.33 ±	1.665	110	132
	Sthc	108.07 ±	0.439	106	110
U250_06	Fihc	110.13 ±	1.666	105	119
	Sdhc	125.90 ±	1.895	113	136
	Sthc	108.43 ±	0.403	106	110
U250_07	Fihc	111.30 ±	1.774	106	122
	Sdhc	126.13 ±	1.754	113	135
	Sthc	109.77 ±	0.380	108	112
U250_08	Fihc	112.93 ±	1.765	108	124
	Sdhc	123.00 ±	2.359	112	134
	Sthc	112.10 ±	0.463	110	114
U250_09	Fihc	109.50 ±	1.900	104	120
	Sdhc	126.40 ±	1.405	120	135
	Sthc	107.27 ±	0.459	105	110

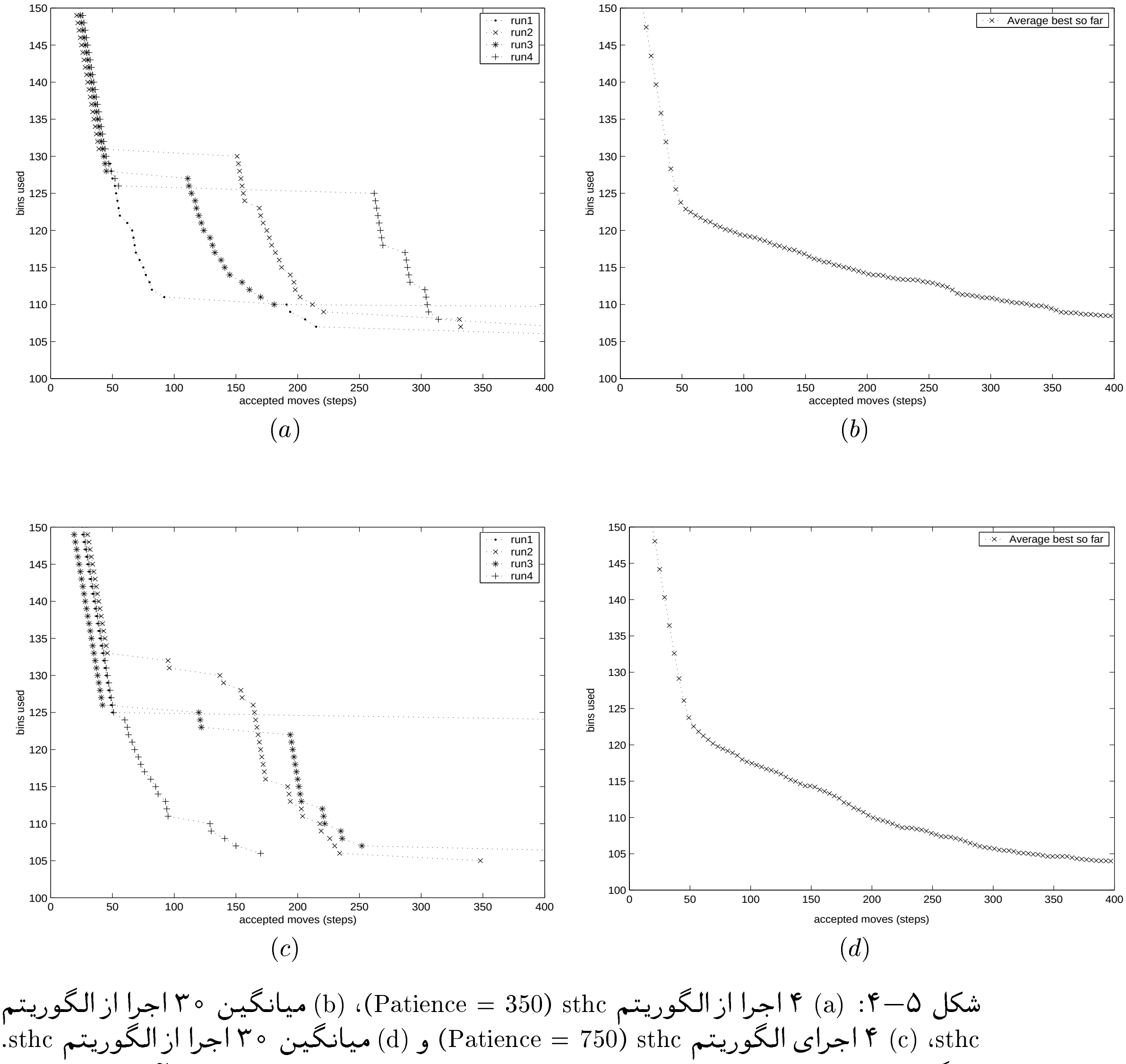
جدول ۵–۵: نتایج حاصل حل نمونه مسئله‌های مختلف توسط STAGE با الگوریتم‌های جستجوی محلی مختلف. هر الگوریتم برای هر نمونه از مسئله ۳۰ بار اجرا شده است و TotalEvals = 2,000,000 می‌باشد.

Instance	Algorithm	Mean ± Tolerance	Best	Worst
U250_00	Fihc	104.83 ± 1.546	102	116
	Sdhc	105.17 ± 1.599	102	115
	Sthc	113.03 ± 2.945	105	131
U250_01	Fihc	104.87 ± 1.193	103	114
	Sdhc	104.30 ± 1.029	102	114
	Sthc	115.60 ± 2.989	106	134
U250_02	Fihc	108.80 ± 1.914	105	122
	Sdhc	106.37 ± 0.844	104	116
	Sthc	114.53 ± 2.412	108	135
U250_03	Fihc	105.00 ± 1.342	102	116
	Sdhc	104.37 ± 0.892	103	113
	Sthc	110.60 ± 2.026	106	128
U250_04	Fihc	105.93 ± 0.836	104	116
	Sdhc	106.77 ± 1.527	104	120
	Sthc	115.97 ± 3.077	107	135
U250_05	Fihc	107.37 ± 1.443	104	119
	Sdhc	107.23 ± 1.643	104	119
	Sthc	113.23 ± 2.626	106	134
U250_06	Fihc	106.93 ± 1.353	104	117
	Sdhc	107.43 ± 1.696	104	119
	Sthc	116.40 ± 3.554	108	135
U250_07	Fihc	108.67 ± 1.191	106	119
	Sdhc	108.53 ± 1.129	106	120
	Sthc	118.20 ± 3.211	110	136
U250_08	Fihc	109.77 ± 0.903	108	121
	Sdhc	110.27 ± 0.836	108	121
	Sthc	118.33 ± 2.496	112	137
U250_09	Fihc	106.80 ± 1.654	103	118
	Sdhc	107.13 ± 1.819	104	120
	Sthc	112.03 ± 2.322	106	132

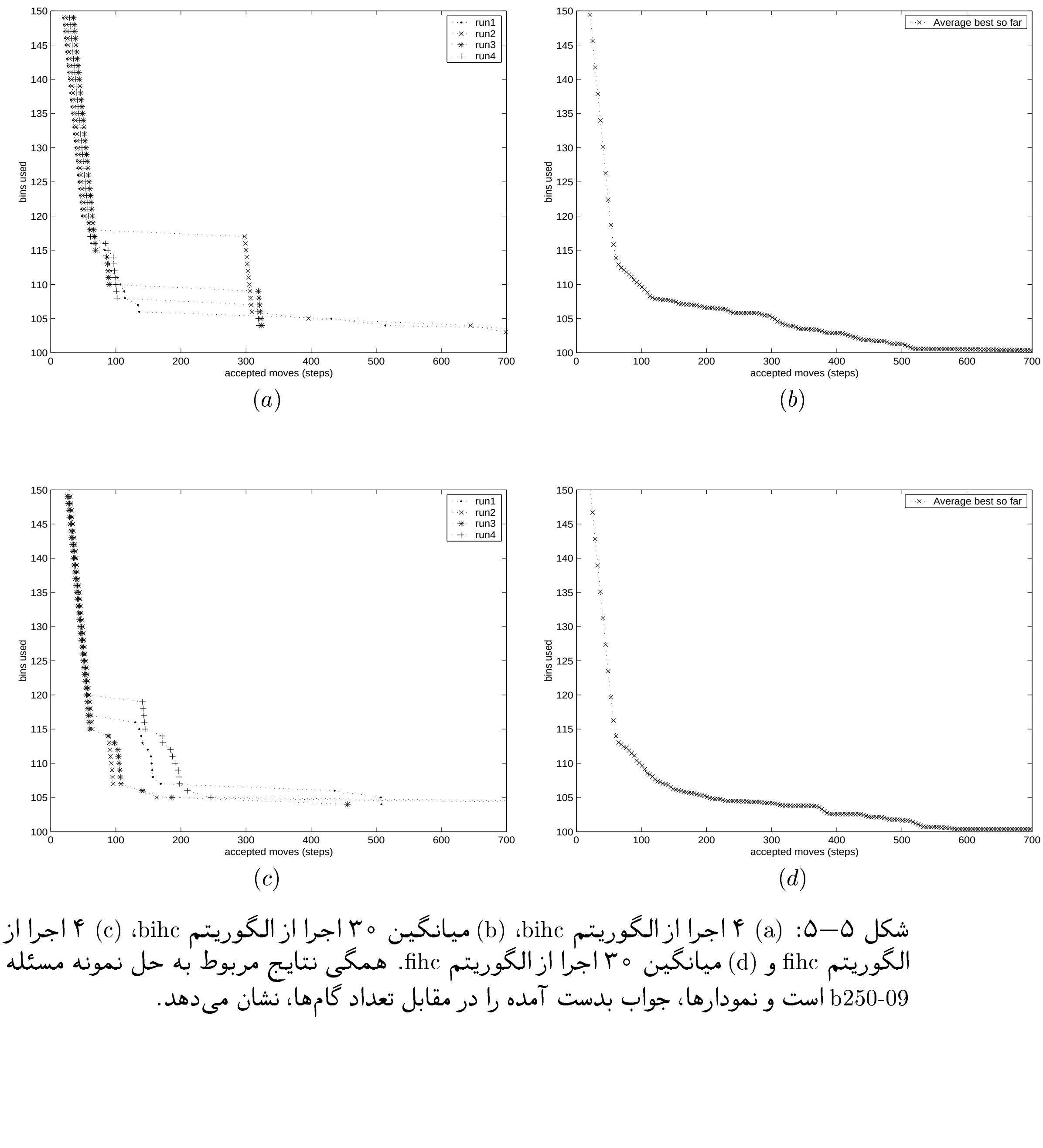
جدول ۵–۶: نتایج حاصل حل نمونه مسئله‌های مختلف توسط STAGE با الگوریتم‌های جستجوی محلی مختلف. هر الگوریتم برای هر نمونه از مسئله ۳۰ بار اجرا شده است و حداکثر گام‌ها (حرکت‌های پذیرفته شده)، 400 می‌باشد.



شکل ۵–۳: (a) ۴ اجرا از الگوریتم bihc، (b) میانگین ۳۰ اجرا از الگوریتم bihc، (c) ۴ اجرا از الگوریتم fihc و (d) میانگین ۳۰ اجرا از الگوریتم fihc. همگی نتایج مربوط به حل نمونه مسئله b250-00 است و نمودارها، جواب بدست آمده را در مقابل تعداد گام‌ها، نشان می‌دهد.



شکل ۵–۴: (a) ۴ اجرا از الگوریتم sthc (Patience = 350)، (b) میانگین ۳۰ اجرا از الگوریتم sthc، (c) ۴ اجرای الگوریتم sthc (Patience = 750) و (d) میانگین ۳۰ اجرا از الگوریتم sthc. همگی نتایج مربوط به حل نمونه مسئله b250-00 است و نمودارها، جواب بدست آمده را در مقابل تعداد گام‌ها، نشان می‌دهد.



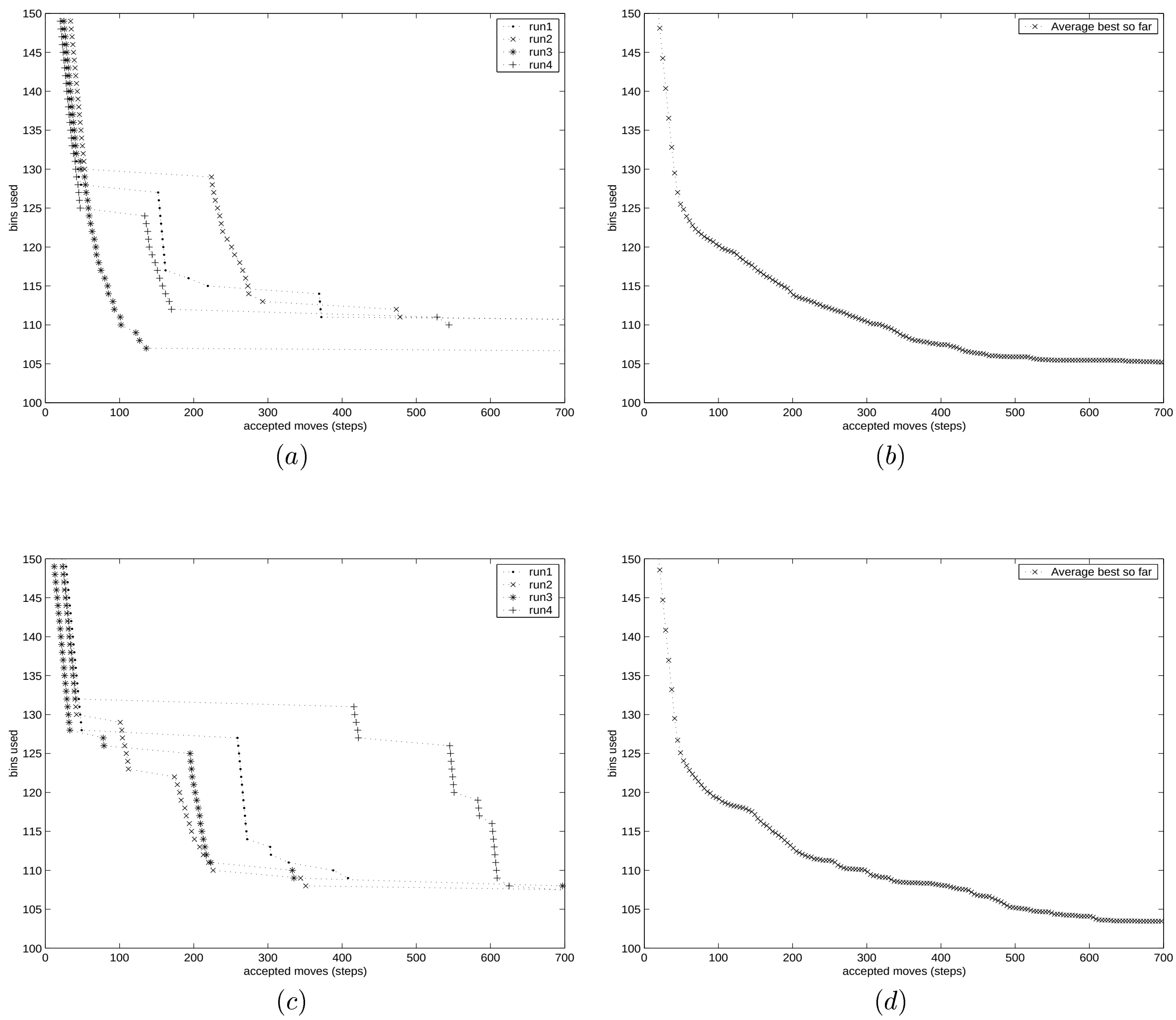
شکل ۵–۵: (a) ۴ اجرا از الگوریتم bihc، (b) میانگین ۳۰ اجرا از الگوریتم bihc، (c) ۴ اجرا از الگوریتم fihc و (d) میانگین ۳۰ اجرا از الگوریتم fihc. همگی نتایج مربوط به حل نمونه مسئله b250-09 است و نمودارها، جواب بدست آمده را در مقابل تعداد گام‌ها، نشان می‌دهد.

شکل ۵—۴، نمودارهای عملکرد دو الگوریتم Sthc1 و Sthc2 را برای نمونه مسئله U250_00 نشان می‌دهند، نمانطور که دیده می‌شود الگوریتم Sthc2 بطور میانگین بهتر از Sthc1 عمل کرده است؛ بعبارتی پارامتر Patience بالاتر، منجر به بدست آمدن جوابهای دقیق‌تری می‌شود (به شکل ۵—٦ که همین نمودارها را برای نمونه مسئله U250_09 رسم می‌کند، دقت کنید).

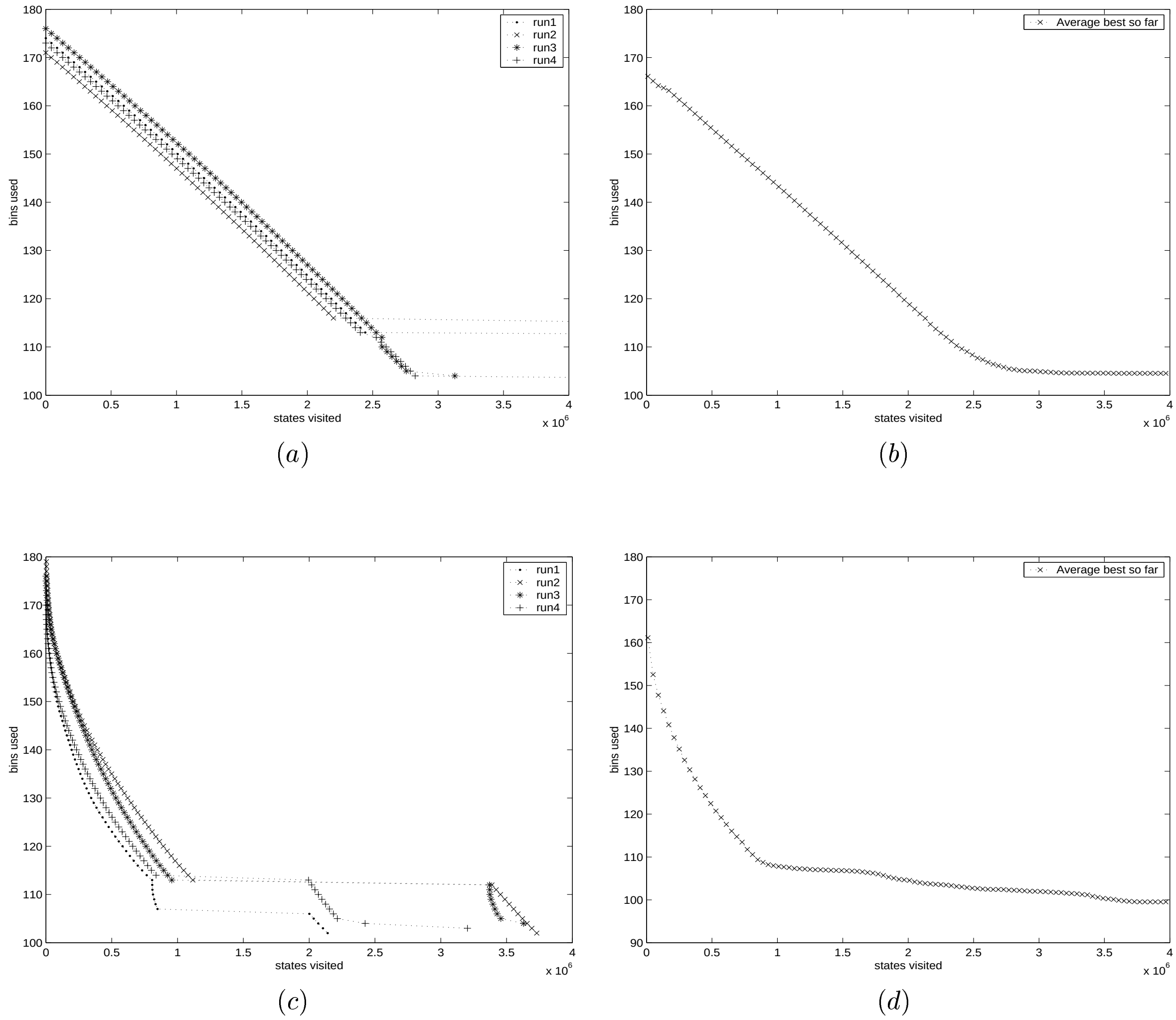
شکل‌های ۵—۷ و ۵—۸ عملکرد الگوریتم‌ها در حل نمونه مسئله U250_00 با محدودیت روی گام‌های برداشته شده را رسم می‌کند. در شکل ۵—۷، نمودار (a) چهار اجرای الگوریتم Sdhc را روی این نمونه مسئله نشان می‌دهد و نمودار (b) عملکرد ۳۰ اجرای الگوریتم Sdhc را نشان می‌دهد، به این طریق که هر نقطه میانگین ۳۰ اجرای الگوریتم است؛ نمودارهای (c) و (d) همین نمودارها را برای الگوریتم Fihc نشان می‌دهند. همانطور که مشاهده می‌کنید عملکرد این دو الگوریتم بسیار شبیه به هم است.

شکل ۵—۸، نمودارهای عملکرد دو الگوریتم Sthc1 و Sthc2 را برای نمونه مسئله U250_00 نشان می‌دهند، نمانطور که دیده می‌شود الگوریتم Sthc2 بطور میانگین بهتر از Sthc1 عمل کرده است؛ بعبارتی پارامتر Patience بالاتر، منجر به بدست آمدن جوابهای دقیق‌تری می‌شود (به شکل ۵—۹ که همین نمودارها را با وضوح بیشتر رسم می‌کند، دقت کنید).

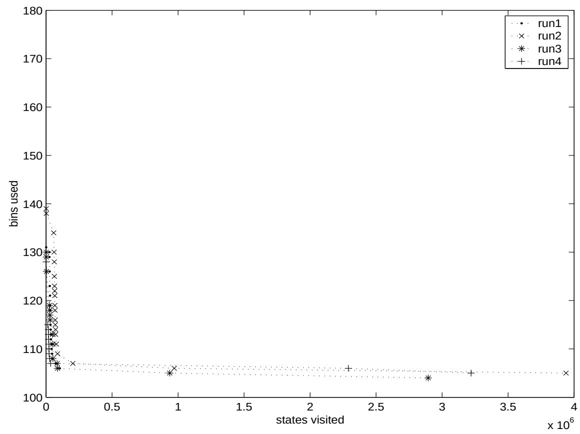
شکل ۵—۱۰ مقایسه الگوریتم‌ها را بطور کامل نشان می‌دهد. در نمودار (a) هر چهار الگوریتم بر اساس حداکثر تعداد گام‌های مجاز، مقایسه شده‌اند، Bihc و Fihc عملکردی شبیه به هم دارند و هر دو بهتر از Sthc عمل می‌کنند. همچنین Sthc2 بهتر از Sthc1 است و این نشان می‌دهد اگر تصمیم‌گیری در مورد حالت بعدی با «دقت» بیشتری انجام گیرد (یعنی در انتخاب حالت بعدی، تعداد همسایه‌های بررسی شده زیادتر باشد)، با برداشتن تعداد کمتری گام به جواب خواهیم رسید (این



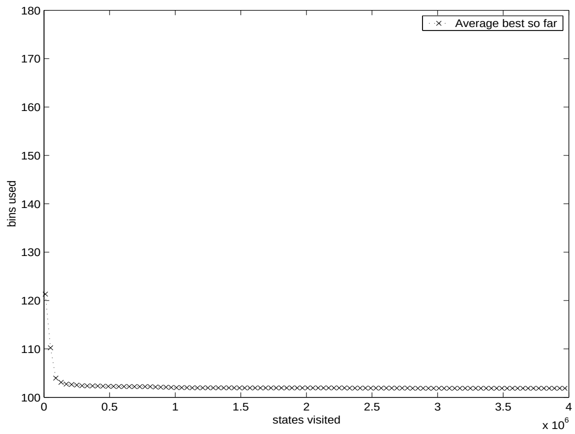
شکل ۵–۶: (a) ۴ اجرا از الگوریتم sthc (Patience = 350)، (b) میانگین ۳۰ اجرا از الگوریتم sthc، (c) ۴ اجرای الگوریتم sthc (Patience = 750) و (d) میانگین ۳۰ اجرا از الگوریتم sthc. همگی نتایج مربوط به حل نمونه مسئله b250-09 است و نمودارها، جواب بدست آمده را در مقابل تعداد گام‌ها، نشان می‌دهد.



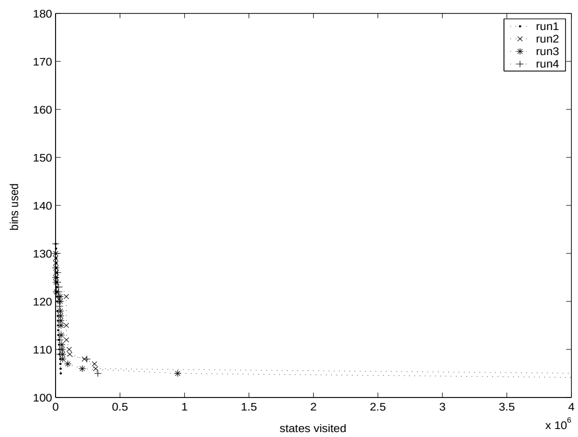
شکل ۵–۷: (a) ۴ اجرا از الگوریتم bihc، (b) میانگین ۳۰ اجرا از الگوریتم bihc، (c) ۴ اجرا از الگوریتم fihc و (d) میانگین ۳۰ اجرا از الگوریتم fihc. همگی نتایج مربوط به حل نمونه مسئله b250-09 است و نمودارها، جواب بدست آمده را در مقابل تعداد حالات بررسی شده، نشان می‌دهد.



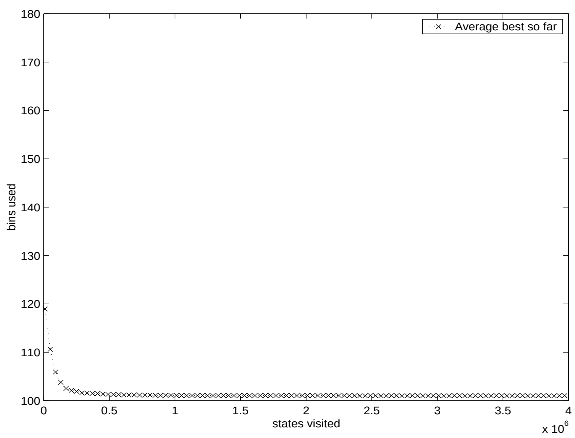
(a)



(b)

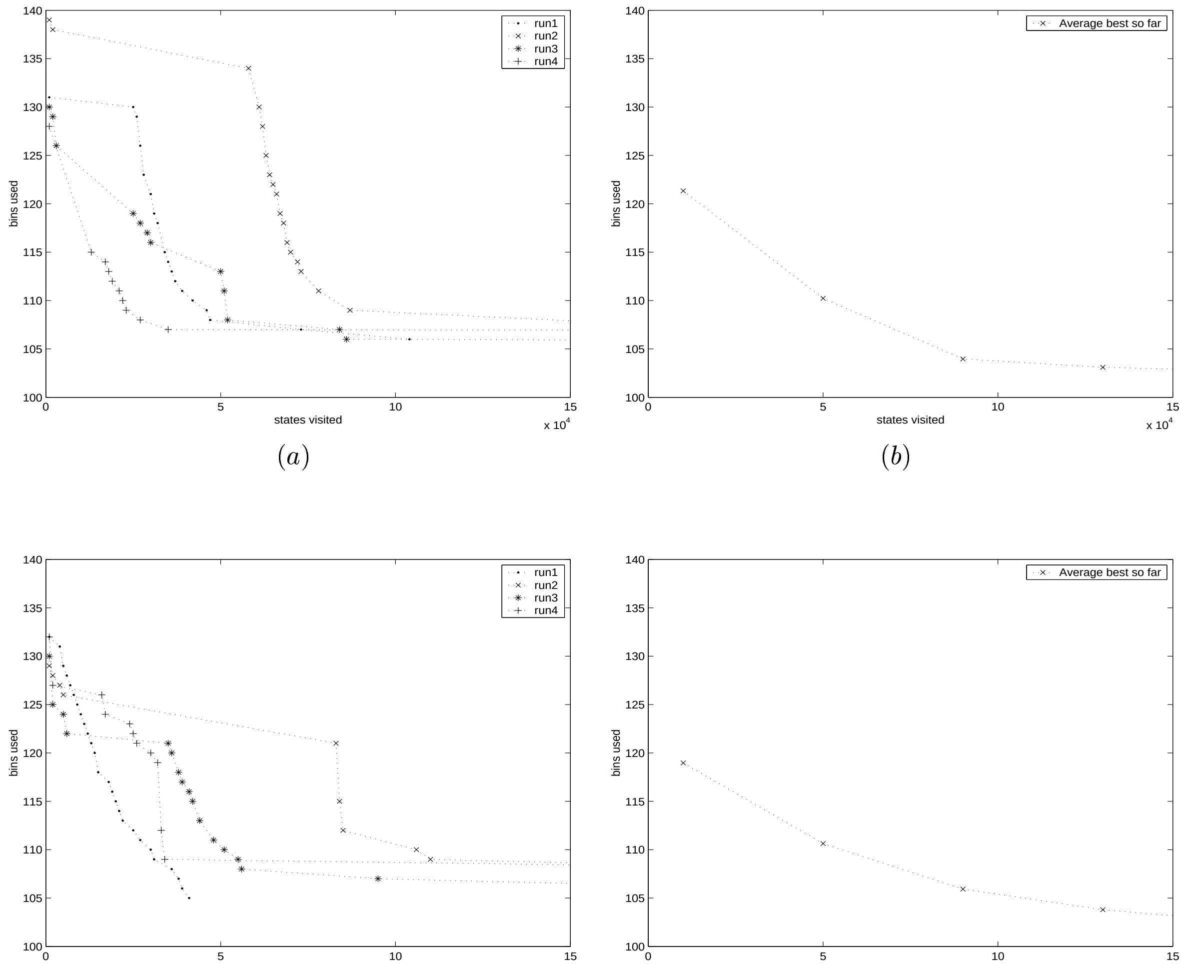


(c)

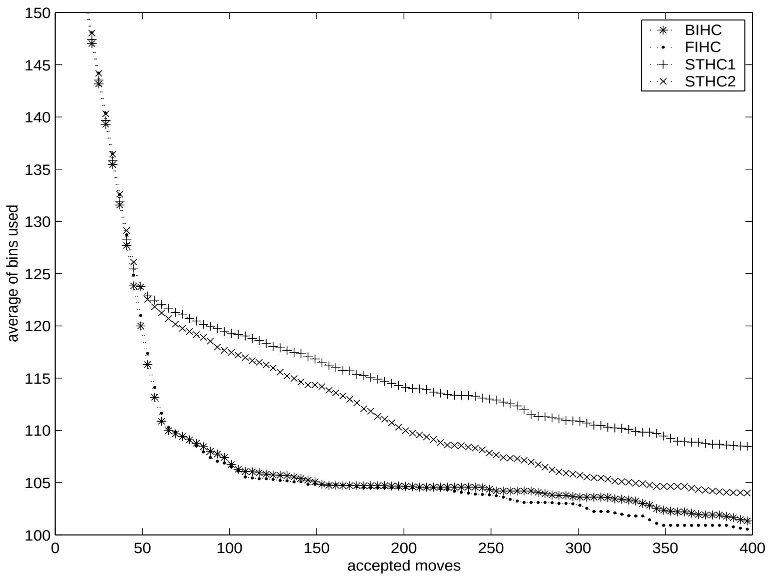


(d)

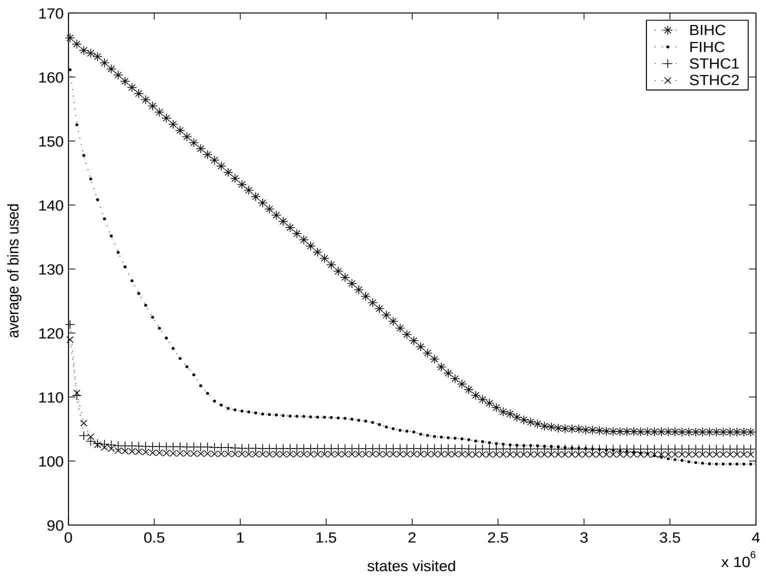
شکل ۵—۸: (a) ۴ اجرا از الگوریتم sthc (Patience = 350)، (b) میانگین ۳۰ اجرا از الگوریتم sthc، (c) ۴ اجرای الگوریتم sthc (Patience = 750) و (d) میانگین ۳۰ اجرا از الگوریتم sthc. همگی نتایج مربوط به حل نمونه مسئله b250-00 است و نمودارها، جواب بدست آمده را در مقابل تعداد حالات بررسی شده، نشان می‌دهد.



شکل ۵–۹: (a) ۴ اجرا از الگوریتم sthc (Patience = 350)، (b) میانگین ۳۰ اجرا از الگوریتم sthc، (c) ۴ اجرای الگوریتم sthc (Patience = 750) و (d) میانگین ۳۰ اجرا از الگوریتم sthc. همگی نتایج مربوط به حل نمونه مسئله b250-00 است و نمودارها، جواب بدست آمده را در مقابل تعداد حالات بررسی شده، نشان می‌دهد.



(a)



(b)

شکل ۵−۱۰: (a) میانگین ۳۰ اجرا از الگوریتم‌های جستجوی محلی مختلف، نمودارِ جواب بدست آمده در مقابل گام‌ها رسم شده است، (b) میانگین ۳۰ اجرا از الگوریتم‌های جستجوی محلی مختلف، نمودارِ جواب بدست آمده در مقابل تعداد حالات بررسی شده، رسم شده است.

موضوع در رابطه با Bihc و الگوریتم‌های دیگر نیز درست است). بطور مهم‌تر، نتایج این نکته را تأکید می‌کند که برداشتن گام‌های عمیق، در نهایت به یافتن سریعتر جواب کمک می‌کند. نکتهٔ جالب دیگر این نمودار این است که جستجوی فضای جواب بطور سیستماتیک بهتر از جستجوی تصادفی (Sthc) می‌باشد، یعنی ساختار سطح هزینه به گونه‌ایست که جستجوی منظم آن منجر به نتیجه خوب می‌شود. حال به نمودار (b) توجه کنید که عملکرد الگوریتم‌ها را با توجه به حالت‌های دیده شده رسم می‌کند، ملاحظه می‌کنید که Sthc بسیار بهتر از Fihc و Bihc عمل کرده است. مقایسه Sthc و Fihc نشان می‌دهد که برداشتن گام‌های مؤثر توسط Fihc مستلزم بررسی تعداد حالت‌های زیادی است و این موضوع در مورد Bihc بطور حادتر مطرح می‌شود چون برداشتن یک گام مشروط به بررسی تعداد زیادی از حالات همسایه است. از طرفی نتایجِ خوب Sthc نشان دهندهٔ این مطلب است که تابع ارزیابی نقش مهمی در فرایند بهینه‌سازی داشته است، چون احتمال این که با Sthc در بهینهٔ محلی گیر کنیم زیاد است که این باعث می‌شود تعداد آموزشها (برای تابع ارزیابی) در Sthc بیشتر از دو الگوریتم دیگر باشد، در نتیجه تابع ارزیابی نقش بیشتری در کنترل نقاط شروع جستجو ایفا می‌کند.

۵–۲ ساختارهای مختلف یادگیری

در این بخش به بررسی استفاده از ساختارهای یادگیری متفاوت در STAGE می‌پردازیم. در فصل ۴ گفتیم که STAGE سعی می‌کند رابطه‌ای را در نگاشت بین وضعیتهای اولیه جستجو و نتیجه بدست آمده (توسط یک الگوریتم جستجوی محلی) کشف کند و آن را «یاد بگیرد»، سپس از این رابطهٔ یاد گرفته شده، استفاده می‌کند تا نتیجهٔ نهایی جستجو از یک حالت را پیش‌بینی کند. یادگیری، با برازش منحنی روی زوج مرتبهای (x,y) انجام می‌شود که x بردار مشخصه‌های حالت [۱۱] و y جواب نهایی

[۱۱] state's features

(یا هزینه) بدست آمده می‌باشد.

شهود ما در مورد ساختار سطح هزینه مسائل ترکیبیاتی، اطلاعاتی در مورد انتخاب یک ساختار تخمین‌زنندهٔ خوب در اختیارمان می‌گذارد. با توجه به شکل دره بزرگ، ما تابع زیر را برای یادگیری انتخاب کرده‌ایم:

$$g(x) = ke^{\frac{||x-m||^{۲}}{۲\sigma^{۲}}} \tag{۱}$$

این تابع شبیه توابع گاوسی است با این تفاوت که در توان، علامت منفی ظاهر نشده است، با تصور شکل این تابع می‌توان شباهت رفتار آن را با دره بزرگ دید. در این تابع k و σ اعدادی حقیقی هستند و m برداری حقیقی است که این پارامترها باید توسط الگوریتم یادگیری طوری تعیین شوند که عبارت زیر را حداقل نمایند:

$$\sum_i (y_i - g(x_i))^{۲} \tag{۲}$$

و در حقیقت ما یک مسئلهٔ حداقل‌سازی خطا را مطرح کرده‌ایم. برای بدست آوردن پارامترها، به روش ساده‌تری نیز می‌توان عمل نمود: برازش یک چندجمله‌ای درجه دوم از زوج مرتب‌های $(x_i, \log(y_i))$. این مسئله کاملاً شناخته شده است و می‌توان با روش جبری مطرح شده در فصل ۳ و انجام یک کار محاسباتی ساده، جواب دقیق آن را بدست آورد. این تابع یاد گرفته شده را $\bar{g}$ می‌نامیم، حال اگر بخواهیم مقدار متناظر با بردار x را توسط $\bar{g}$ محاسبه کنیم باید $e^{\bar{g}(x)}$ را به عنوان نتیجه برگردانیم.

ساختار بعدی که برای یادگیری پیشنهاد کرده‌ایم تابع لگاریتمی زیر می‌باشد:

$$l(x) = \log(\frac{||x-m||^{۲}}{۲\sigma^{۲}} + k) \tag{۳}$$

که k و σ اعدادی حقیقی هستند و m برداری حقیقی است و بصورتی بدست می‌آیند که عبارت زیر را

حداقل نمایند:

$$\sum_i (y_i - l(x_i))^2 \quad (۴)$$

مشابه با حالت قبل، این مسئله را تبدیل به برازش یک چندجمله‌ای درجه دوم از (x_i, e^{y_i}) می‌کنیم. تابع یاد گرفته شده را $\bar{l}$ می‌نامیم، حال اگر بردار مشخصهٔ x داده شد، مقدار متناظر با آن از رابطهٔ $\log(\bar{l}(x))$ بدست می‌آید.

بویان مشخصاتی را برای یک ساختار تخمین‌زنندهٔ خوب (که می‌خواهد همراه با STAGE بکار رود) مطرح ساخته است [۹]: باید در مقابل خطا پایدار باشد [۱۲]، افزایشی باشد یعنی الگوریتم آموزش به آن طوری باشد که برای آموزشِ نقاط جدید، نیازی به آموزش مجدد نقاط قدیمی نباشد و مقادیر خوبی برای حالات دیده نشده تولید نماید. با این شرایط او شبکه CMAC را پیشنهاد نمود که ما قبلاً آن را در فصل ۳ توضیح دادیم.

۵–۲–۱ آزمایشات روی bin packing

در این بخش به ارائه نتایج آزمایشات انجام شده روی مسئله bin packing می‌پردازیم. چهار ساختار تابع برای یادگیری پیشنهاد کرده‌ایم که عبارتند از: شبکه CMAC[۱۳]، تابع لگاریتمی، تابع نمایی و چندجمله‌ای درجه دوم. به همان دلایلی که در بخش قبل مطرح شد، STAGE با هر کدام از ساختارهای یادگیری به دفعات متعدد (۳۰ بار) روی یک نمونه مسئله اجرا شده است و مقایسه بر اساس بهترین جواب، بدترین جواب، میانگین و فاصله اطمینان ۹۰ درصدی برای میانگین می‌باشد.

---

[۱۲] noise tolerant

[۱۳] در بکارگیری شبکه CMAC در STAGE، از پیاده‌سازی دانشگاه New Hampshire برای CMAC استفاده شده است، این پیاده‌سازی در این صفحه اینترنتی قابل دسترسی است: http://ece.unh.edu/robots/cmac.htm

Instance	Algorithm	Mean ± Tolerance	Best	Worst
U500_00	Exp	213.44 ± 3.61	208	251
	Log	235.94 ± 6.12	212	251
	Cmac	218.94 ± 0.64	215	222
	Poly	210.39 ± 0.83	206	214
U500_01	Exp	214.69 ± 0.93	210	218
	Log	244.80 ± 4.37	217	255
	Cmac	222.60 ± 0.50	220	225
	Poly	214.10 ± 0.94	210	220
U500_02	Exp	217.50 ± 3.73	210	255
	Log	244.30 ± 4.43	218	255
	Cmac	223.14 ± 0.56	220	225
	Poly	216.39 ± 2.11	211	236
U500_03	Exp	217.14 ± 0.87	212	221
	Log	244.60 ± 5.33	220	258
	Cmac	225.55 ± 0.71	222	229
	Poly	217.85 ± 1.15	213	225
U500_04	Exp	219.39 ± 0.84	215	222
	Log	244.69 ± 4.80	227	260
	Cmac	227.30 ± 0.51	224	229
	Poly	218.94 ± 0.89	216	223

جدول ۵—۷: نتایج حاصل حل نمونه مسئله‌های مختلف توسط STAGE با ساختارهای یادگیری متفاوت. هر الگوریتم برای هر نمونه از مسئله ۳۰ بار اجرا شده است.

در جداول ۵—۷ و ۵—۸ نتایج بدست آمده از اجرای STAGE همراه با ساختارهای یادگیری ذکر شده، روی نمونه مسئله‌های U500 آورده شده است. دقت کنید که $\text{TotalEvals} = 150000$ می‌باشد، در جدول ۵—۹ نتایج برای این نمونه مسائل در حالتی که $\text{TotalEvals} = 50000$ باشد، ارائه شده است.

همانطور که در جدول‌های ۵—۷ و ۵—۸ دیده می‌شود، چندجمله‌ای درجه دوم بهتر از توابع دیگر عمل کرده است و جوابهای بهتری بدست آورده است. اما تابع نمایی نیز تا حد زیادی شبیه به آن عمل کرده است و در حالتی که تعداد حالات مجاز کم است (جدول ۵—۹)، این تابع بهتر از بقیه عمل کرده است. ما این پدیده را به نحوهٔ توزیع بهینه‌های محلی در فضای جواب نسبت می‌دهیم[۱۴]؛ بهینه‌های محلی در فضا به شکل یک تابع نمایی پخش شده‌اند.

[۱۴] قضیه فاجعه حد مرکزی را در فصل ۴ به خاطر بیاورید.

Instance	Algorithm	Mean ±	Tolerance	Best	Worst
U500_05	Exp	219.69 ±	1.159	214	226
	Log	245.14 ±	5.910	223	261
	Cmac	227.25 ±	0.781	225	232
	Poly	218.75 ±	1.012	215	223
U500_06	Exp	221.20 ±	0.852	216	226
	Log	245.35 ±	5.160	223	258
	Cmac	229.69 ±	0.547	227	233
	Poly	220.44 ±	0.582	217	223
U500_07	Exp	219.44 ±	3.834	214	259
	Log	248.69 ±	4.840	220	260
	Cmac	226.00 ±	0.772	223	231
	Poly	217.50 ±	0.839	213	221
U500_08	Exp	209.94 ±	0.950	206	216
	Log	238.10 ±	4.570	211	249
	Cmac	217.05 ±	0.347	216	219
	Poly	209.14 ±	0.965	203	213
U500_09	Exp	214.25 ±	0.726	210	217
	Log	236.80 ±	5.770	218	254
	Cmac	222.25 ±	0.361	221	224
	Poly	215.50 ±	3.443	210	251

جدول ۵–۸: نتایج حاصل حل نمونه مسئله‌های مختلف توسط STAGE با ساختارهای یادگیری متفاوت. هر الگوریتم برای هر نمونه از مسئله ۳۰ بار اجرا شده است.

Instance	Algorithm	Mean ±	Tolerance	Best	Worst
U500_00	Exp	215.14 ±	1.429	210	223
	Poly	221.94 ±	4.688	210	254
U500_01	Exp	219.50 ±	3.028	212	245
	Poly	222.80 ±	2.882	215	244
U500_02	Exp	226.50 ±	4.168	217	256
	Poly	222.00 ±	3.269	215	251
U500_03	Exp	225.94 ±	3.443	218	254
	Poly	224.14 ±	2.493	215	243
U500_04	Exp	226.60 ±	2.824	218	249
	Poly	226.30 ±	1.648	220	233
U500_05	Exp	227.14 ±	3.650	217	261
	Poly	226.80 ±	3.694	220	262
U500_06	Exp	227.55 ±	3.974	221	268
	Poly	229.05 ±	3.806	221	265
U500_07	Exp	225.94 ±	4.839	215	261
	Poly	225.64 ±	3.379	218	255
U500_08	Exp	213.80 ±	1.242	207	218
	Poly	215.64±	2.797	209	241
U500_09	Exp	220.00 ±	1.325	215	226
	Poly	223.85 ±	4.220	214	257

جدول ۵–۹: نتایج حاصل حل نمونه مسئله‌های مختلف توسط STAGE با ساختارهای یادگیری نمایی و چندجمله‌ای. هر الگوریتم برای هر نمونه از مسئله ۳۰ بار اجرا شده است.

۵—۳ چشم‌انداز شایستگی

در این بخش می‌خواهیم از دید مبحث محاسبات تکاملی به ساختار سطح هزینه مسائل نگاه کنیم. در واقع چشم‌انداز شایستگی[15] نامی دیگر برای مفهوم ساختار سطح هزینه است و یکی از مفاهیم پایه در تئوری تکامل می‌باشد. همراه با این مفهوم، بطور ضمنی مجموعه‌ای از genotypeها وجود دارد که با ترتیب خاصی (در یک فضای دنباله [16]) مرتب شده‌اند و هر genotype مجاور دیگری است اگر تنها در مقدار یک ژن با هم تفاوت داشته باشند (به عبارتی تنها با یک اعمال عملگر ژنتیکی جهش، بتوان از یکی به دیگری رسید). این فضا و مقادیر عددی شایستگی مربوط به هر genotype، چشم‌انداز شایستگی را تشکیل می‌دهد. به عنوان مثال می‌توان پدیده‌هایی فیزیکی مانند شایستگی هر مولکول در انجام یک عمل خاص یا تکثیر RNA در یک محیط زجاجی [17] را با این مفهوم توصیف کرد.

از مدتها پیش محققان می‌دانسته‌اند که دینامیک تکامل روی یک چشم‌انداز شایستگی به ساختار آماری آن بستگی دارد، یعنی برای پیدا کردن نقطه بهینه نمی‌توان فرض استاندارد داروین در مورد جهش تصادفی را بصورت کورکورانه بکار برد چون بطور خیلی ساده ممکن است مقادیر شایستگی بصورت تصادفی توزیع نشده باشند. لذا نیاز به توصیف چشم‌انداز شایستگی برای دامنه‌های مختلف با ابزار بیان مختلف احساس شد. بطور مثال همانطور که ما ساختار دره بزرگ را برای مسائل بهینه‌سازی ترکیبیاتی دیدیم، فراکتالها برای توصیف چشم‌انداز شایستگی دیگر مسائل مورد استفاده قرار گرفتند [۳۴][18].

---

[15] fitness landscape

[16] sequence space

[17] vitro

[18] در این مرجع، مدل N-K و مدل P-spin نیز برای بیان چشم‌انداز شایستگی مورد بررسی قرار گرفته‌اند.

۵–۳–۱ همبستگی فاصله محدود

همبستگی فاصله محدود [۱۹] (FDC) معیاری است که Jones و Forrest برای نشان دادن درجهٔ مشکلی مسائل بهینه‌سازی (یا جستجو) پیشنهاد کردند [۲۴] که این معیار از ایدهٔ تناظر میان الگوریتم‌های ژنتیک و جستجوهای ابتکاری فضای حالت نتیجه شده است [۲۳]. با در نظر گرفتن افراد جمعیت به عنوان رئوس گراف جستجو و ایجاد یال بین هر دو رأسِ با فاصله هامیلتونی یک (یعنی با انجام تنها یک جهش بتوان از یکی به دیگری رسید) یک ساختار همسایگی بدست می آید که با مقدار شایستگی هر نقطه، چشم‌انداز شایستگی را می‌سازد (همانطور که در بخش قبل گفته شد). همانطور که در جستجوهای ابتکاری فرآیند جستجو توسط یک تابع ابتکار [۲۰] راهنمایی می‌شود، در الگوریتم‌های تکاملی نیز فرایند پیدا کردن جواب توسط تابع شایستگی هدایت می‌شود [۲۴]. در جستجوی ابتکاری فضای حالت، اصل مهمی که در انتخاب تابع ابتکار در نظر گرفته می‌شود این است که مقدار این تابع باید به نوعی با فاصله تا نقطه بهینه، همبستگی [۲۱] داشته باشد. به عنوان مثال در الگوریتم جستجوی ابتکاری A^*، از تابع ابتکار به عنوان تخمینی از فاصله تا نقطه بهینه استفاده می‌شود. Jones و Forrest توجه کردند که اگر ارتباط میان الگوریتم ژنتیک و جستجوی ابتکاری درست است آنگاه میزانی که تابع شایستگی معرف این مفهوم (یعنی همبستگی میان مقدار تابع و فاصله تا نقطه بهینه که در جستجوی ابتکاری دیده شد) است می‌تواند نشان دهندهٔ درجه سختی چشم‌اندازی باشد که باید جستجو شود. آنها معیار FDC را بر اساس همبستگی مقدار تابع شایستگی با فاصله تا نقطه بهینه تعریف کردند و از آن برای رده‌بندی سختیِ مسائلی که می‌توان با الگوریتم ژنتیک حل کرد، استفاده نمودند. در ادامه بطور مختصر روش کار آنها را توضیح خواهم داد.

[۱۹] finite distance correlation

[۲۰] heuristic function

[۲۱] correlation

ساده‌ترین راه برای اندازه‌گیری همبستگی میان تابع شایستگی و فاصله تا نقطه بهینه، این گونه است که ابتدا یک مسئله که بهینه سراسری آن از قبل مشخص است در نظر گرفته می‌شود، سپس با داشتن چند نمونه از زوج مرتبهای (فاصله،شایستگی) می‌توان ضریب همبستگی r را بدست آورد. با داشتن $F = \{f_۱, \ldots, f_n\}$ از مقدار شایستگی n نفر و $D = \{d_۱, \ldots, d_n\}$ از فاصله این n نفر تا نقطه بهینه، می‌توان ضریب همبستگی r را بصورت زیر محاسبه نمود:

$$r = \frac{\frac{۱}{n}\sum_{i=۱}^{n}(f_i - \bar{f})(d_i - \bar{d})}{S_F S_d} \qquad (۵)$$

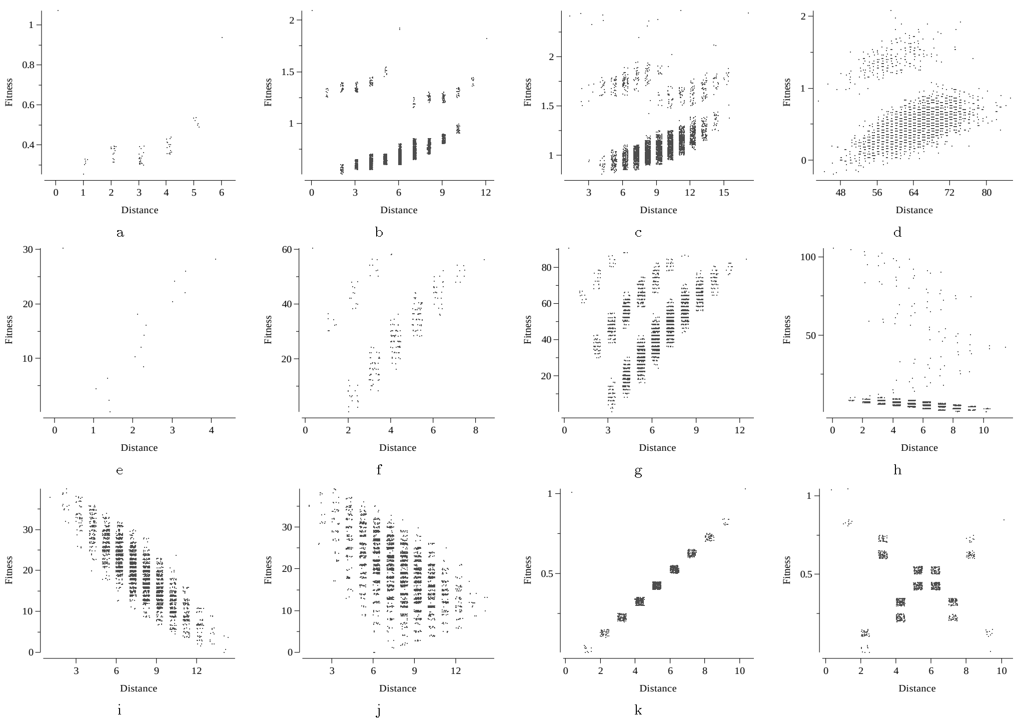
که S_F، S_D، $\bar{f}$ و $\bar{d}$ به ترتیب انحراف استاندارد شایستگی‌ها، انحراف استاندارد فاصله‌ها، میانگین شایستگی‌ها و میانگین فاصله‌ها هستند.

البته ضریب همبستگی تنها یک معیار از میان معیارهای ممکن برای نشان دادن ارتباط میان فاصله و شایستگی می‌باشد و می‌توان از معیارهای دیگر نیز استفاده نمود. در هر صورت رسم نمودار دو بعدی شایستگی—فاصله می‌تواند آموزنده باشد. در واقع نمودارهای پراکندگی رسم شده در فصل ۴ نیز بر مبنای همین ایده هستند، در آنجا ما از پراکندگی نقاط (حول نقطه بهینه سراسری) توانستیم به شهود دره بزرگ برسیم.

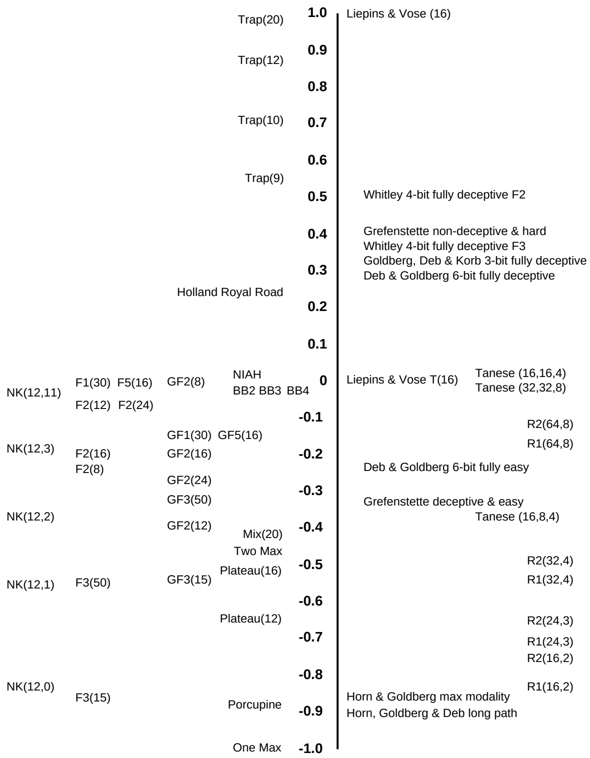
اگر مسئله ما یک مسئله بیشینه‌سازی باشد آنگاه باید امیدوار باشیم که با زیاد شدن شایستگی، مقدار فاصله تا بهینه سراسری کم شود و لذا مقدار ایده‌ال برای r، ۱− خواهد بود. برعکس، اگر مسئله ما یک مسئله کمینه‌سازی باشد آنگاه مقدار ایده‌ال برای r، ۱+ خواهد بود. در [۲۴] چندین نمونه مسئله بیشینه‌سازی در نظر گرفته شده‌اند و برای آنها مقدار r محاسبه شده است. می‌توانید این مسائل را بر حسب مقدار r آنها در شکل ۵—۱۲ ببینید. توضیح در مورد مسائل و اختصار نام آنها در جدول ۵—۱۰ آمده است. در شکل ۵—۱۱ نیز می‌توانید نمودارهای پراکندگی نقاط بر حسب فاصله تا نقطه

Abbreviation	Problem Description
BBk	Busy Beaver problem with k states.
Deb & Goldberg	6-bit fully deceptive and easy functions.
Fk(j)	De Jong's function k with j bits.
GFk(j)	As above, though Gray coded.
Goldberg, Korb & Deb	3-bit fully deceptive.
Grefenstette easy	The deceptive but easy function.
Grefenstette hard	The non-deceptive but hard function.
Holland royal road	Holland's 240-bit royal road function.
Horn, Goldberg & Deb	The long path problem with 40 bits.
Horn & Goldberg	A 33-bit maximally rugged function.
Liepins & Vose (k)	Deceptive problem with k bits.
Mix(n)	Ackley's mix function on n bits.
NIAH	Needle in a haystack.
NK(n; k)	Kauffman's NK landscape, N = n, K=k.
One Max	Ackley's single-peaked function.
Plateau(n)	Ackley's plateau function on n bits.
Porcupine(n)	Ackley's porcupine function on n bits.
R(n; b)	Mitchell et al. n-bit royal road, b-bit blocks.
Tanese (l; n; o)	l-bit Tanese function of n terms, order o.
Trap(n)	Ackley's trap function on n bits.
Two Max(n)	Ackley's two-peaked function on n bits.
Whitley Fk	4-bit fully deceptive function k.

جدول ۵–۱۰: توضیح اختصارها و نام توابعی که در شکل‌های ۵–۳ و ۵–۴ از آنها استفاده شده است، برای توضیح بیشتر در مورد نام منابع مسائل به [22] مراجعه کنید.



شکل ۵—۱۱: در این شکل نمونه‌هایی از نمودارهای پراکندگی شایستگی بر حسب فاصله را مشاهده می‌کنید. منبع توابع در جدول ۵—۷ آمده است. مقدار FDC برای مسئله‌هایی که از ۱۲ بیت بزرگتر هستند از محاسبه بر روی ۴۰۰۰ نقطه تصادفی بدست آمده است. توابع به این صورت هستند: (a-c) یک، دو و سه کپی از مسئله فریبندهٔ کامل Deb-Goldberg (۶ بیتی) هستند ($r = 0.3$). (d) مسئله راه پادشاهی (royal road) از Holland روی ۱۲۸ بیت ($b = 8$، $k = 4$ و $g = 0$) ($r = .27$). (e-g) یک، دو و سه کپی از مسئله فریبندهٔ F2 از Whitely که ۴ بیتی است ($r = 0.51$). (h) مسئله مسیر طولانی از Horn، Goldberg و Deb با ۱۱ بیت ($r = -0.12$). (i،j) مسئله بهینه‌سازی کد Gray و F3 (مربوط به De Jonge) که ۱۵ بیتی است ($r = -0.57$ و -8.86). (k) مسئله فریبنده کاملِ Liepins و Vose روی ۱۰ بیت ($r = 0.98$). (l) مسئله تغییریافته در قسمت k ($r = -0.02$).



شکل ۵–۱۲: خط عمودی مقادیر مختلف r را برای مسائل مختلف نشان می‌دهد. منبع مسائل و اختصار نام‌ها در جدول ۵–۷ آمده است.

بهینه را برای این مسائل ببینید[۲۲].

۵—۴ ساختار سطح هزینه مسئله مرتب‌سازی صفحه درهم

در این بخش ابتدا به تعریف مسئله مرتب‌سازی صفحه درهم (BBS) می‌پردازیم و سپس به بررسی ساختار سطح هزینه آن خواهیم پرداخت. بررسی نتایج به ما کمک می‌کند تا بدانیم STAGE روی چه مسائلی خوب عمل می‌کند.

۵—۴—۱ معرفی مسئله

در بازی BBS، ۲۵ مکعب وجود دارند که روی وجوه هر یک از آنها یک حرف انگلیسی نوشته شده است. این مکعبها در یک صفحه مربعی شکل ۵ × ۵ قرار دارند (شکل ۵—۱۳). هدف بازی پیدا کردن کلمات انگلیسی است که روی مسیرهای مختلف این مربع قرار دارند. یک مسیر قانونی می‌تواند شامل گام‌های افقی، عمودی و مورب باشد، اما یک مسیر نمی‌تواند بیشتر از یک بار یک مکعب را انتخاب نماید. کلمات طولانی‌تر باارزش‌تر از کلمات کوتاهتر هستند؛ امتیاز [۲۳] پیدا کردن یک کلمه ۴ حرفی ۱ واحد، یک کلمه ۵ حرفی ۲ واحد، یک کلمه ٦ حرفی ۳ واحد، یک کلمه ۷ حرفی ۵ واحد و امتیاز کلمات ۸ حرفی یا بلندتر، ۱۱ واحد می‌باشد.

[۲۲] این دو شکل از منبع [۲۴] اقتباس شده‌اند.
[۲۳] score

G	R	R	W	Y
H	W	X	V	P
Z	K	Y	T	W
D	J	G	D	D
Y	A	D	S	Y

R	S	T	C	S
D	E	I	A	E
G	N	L	R	P
E	A	T	E	S
M	S	S	I	D

شکل ۵—۱۳: در سمت چپ یک صفحه درهم را می‌بینید که بطور تصادفی تولید شده است (۸ کلمه، ۱۰ امتیاز و Obj = -0.01). در سمت راست هم یک صفحه درهم بهینه شده را می‌بینید (۲۰۳۴ کلمه، ۹۲۴۵ امتیاز و Obj = -9.245). صفحهٔ سمت راست شامل کلماتی مانند delicateness، depreciated و specialties می‌باشد.

با داشتن یک صفحه درهم x، می‌توان تمام کلمات انگلیسی موجود در آن را با انجام یک کار محاسباتی بدست آورد؛ با نشان دادن فرهنگ لغت [۲۴] با یک درخت پیشوندی، Score(x) (امتیاز صفحه x) می‌تواند در چند میلی ثانیه محاسبه شود. با این وجود پیدا کردن این که چه ترکیبی از حروف در صفحه بیشترین امتیاز را دارند کار بسیار مشکلی است. ما کار بیشینه‌سازی را با توجه به Obj(x) = - Score(x) / 1000 به کمینه‌سازی تبدیل نموده‌ایم. توجه کنید که بجای این که در مسئله تحمیل کنیم در محل هر مکعب تنها ٦ حرف (که روی وجوه آن نوشته شده‌اند) می‌توانند ظاهر شوند، فرض کرده‌ایم که در هر محل، هر یک از ٢٦ حرف انگلیسی قابل استفاده هستند و در نتیجه فضای جستجوی بسیار بزرگی با $٢٦^{٢٥}$ حالت ساخته‌ایم. پیدا کردن بهینه سراسری در این فضای بزرگ

[۲۴] نام فرهنگ لغت مورد استفاده OSPD3 می‌باشد که یک فرهنگ لغت استاندارد برای انجام بازیهایی است که با کلمات سر و کار دارند. این فرهنگ لغت در این صفحه اینترنتی در دسترس است: http://runslinux.net/projects.html#margana.

جستجو کاری مشکل [۲۵] است لذا رویکرد استفاده از جستجوی محلی عاقلانه به نظر می‌رسد.

۵–۴–۲ جستجوی محلی

برای بدست آوردن یک بهینه محلیِ تصادفی، می‌توان این گونه عمل کرد: ابتدا یک حالت اولیه تصادفی انتخاب می‌شود (با انتخاب ۲۵ حرف انگلیسی بصورت تصادفی با احتمال یکسان)، سپس هر یک از عملگرهای زیر با احتمال ۰.۵ اِعمال می‌شوند:

- یک خانه از مربع را بطور تصادفی انتخاب کنید و سپس یک حرف جدید بصورت تصادفی انتخاب کرده و در آن خانه بگذارید (حرف جدید بر اساس احتمالی متناسب با تکرارش در فرهنگ لغت انتخاب می‌شود).
- یک خانه از مربع را بطور تصادفی انتخاب کنید و سپس یکی از (حداکثر) ۸ همسایه‌های آن را نیز بطور تصادفی انتخاب کنید، اکنون حرف‌های قرار گرفته در دو خانه را با هم تعویض کنید.

به این صورت با یک الگوریتم جستجوی محلیِ تصادفی، از نقاط اولیه تصادفی به بهینه‌های محلی تصادفی خواهیم رسید.

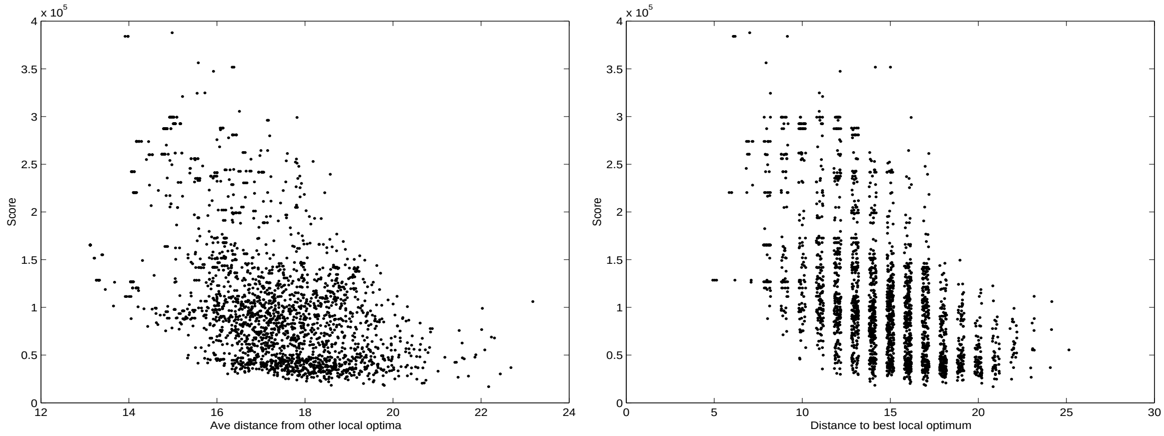
با بکارگیری این الگوریتم جستجوی محلی همراه با STAGE، بویان ملاحظه کرد که عملکرد STAGE در مقایسه با الگوریتم‌های تپه‌نوردی و SA، ضعیف می‌باشد. در کارهای بویان، این تنها دامنه‌ای بود که انتخاب هوشمندانهٔ نقطه آغازین جستجو به بالا بردن کیفیت جواب نهایی کمکی نکرد[۹]. بویان این مطلب را این گونه توضیح داد که STAGE هنگامی موفق است که بتواند رابطه‌ای

[۲۵] intractable

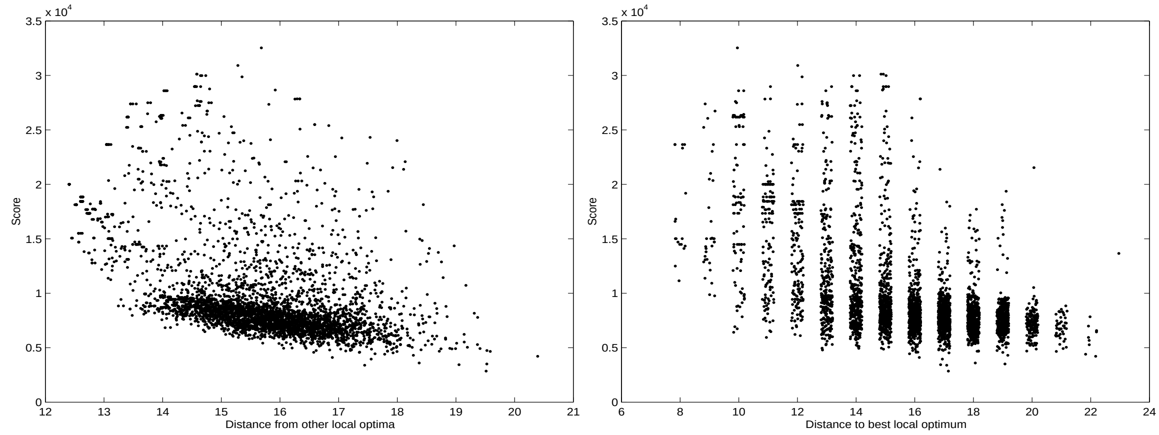
را در نگاشت بین حالات اولیه به جواب نهایی پیدا کند. در مورد مسئله BBS، این رابطه وجود ندارد و نتایج تپه‌نوردی از چند نقطه شروع کاملاً متفاوت رابطهٔ خاصی را نشان نمی‌دهد، لذا STAGE خوب عمل نمی‌کند. اما ما ادعا می‌کنیم که چون STAGE در مسائلی خوب عمل می‌کند که ساختار دره بزرگ بر بهینه‌های محلی آنها حاکم است و این ساختار بر بهینه‌های محلی مسئله BBS حاکم نیست، لذا عملکرد STAGE در حل این مسئله ضعیف است.

۵—۴—۳ ساختار سطح هزینه

برای بدست آوردن شهودی در مورد ساختار سطح هزینه مسئله BBS، من نتایج حاصل از اجرای الگوریتم جستجوی محلی بخش قبل را از تعداد زیادی نقطهٔ اولیه تصادفی بدست آوردم. فاصلهٔ بینِ دو حالت x_1 و x_2 را بر حسب عملگرهای گفته شده در بخش قبل این گونه تعریف می‌کنیم که عبارتست از حداقل عملگرهای لازم برای تبدیل x_1 به x_2 (یا برعکس)، به این صورت یک ساختار همسایگی بین نقاط تعریف می‌شود. حال نمودارهای پراکندگی نقاط در شکل ۵—۱۴ را در نظر بگیرید. در نمودار سمت راست امتیاز هر نقطه بر حسب فاصله آن تا بهترین نقطه بهینه محلی (در بین نقاط بهینه محلی، در حقیقت ما این نقطه را با تقریبی همان بهینه سراسری در نظر گرفته‌ایم) و در سمت چپ، نمودار امتیاز هر نقطه بر حسب میانگین فاصله‌هایش تا بقیه نقاط رسم شده است. توجه کنید که در رسم نمودار سمت راست، هر جا که دو نقطه روی هم منطبق شده‌اند مقدار کمی نویز به یکی از آنها اضافه شده است تا پراکندگی، بهتر مشاهده شود. شکل ۵—۱۵ همین نمودارها را برای حالتی رسم می‌کند که امتیاز همهٔ کلمات (در موقع امتیازدهی به یک وضعیت صفحه) برابر باشد، یعنی ما به کلمات یک، دو و ...حرفی امتیاز یکسانِ یک بدهیم.



شکل ۵–۱۴: پراکندگی ۲۵۰۱ نقطه بهینه محلی برای مسئله BBS، در حالتی که سیستم امتیازدهی به کلمات با طولهای مختلف، امتیازهای متفاوتی بدهد. شکل سمت چپ فاصله متوسط هر نقطه تا بقیه نقاط (۲۵۰۰ نقطه دیگر) را در مقابل امتیاز آن نقطه رسم می‌کند و شکل سمت راست فاصله هر نقطه تا بهترین جواب (یعنی جواب بهینه) را در مقابل امتیاز رسم می‌کند.



شکل ۵–۱۵: پراکندگی ۴۰۰۱ نقطه بهینه محلی برای مسئله BBS، در حالتی که سیستم امتیازدهی به کلمات با طولهای مختلف، امتیازِ یکسانِ یک بدهد. شکل سمت چپ فاصله متوسط هر نقطه تا بقیه نقاط (۴۰۰۰ نقطه دیگر) را در مقابل امتیاز آن نقطه رسم می‌کند و شکل سمت راست فاصله هر نقطه تا بهترین جواب (یعنی جواب بهینه) را در مقابل امتیاز رسم می‌کند.

اگر شکلهای این بخش را با شکلهای فصل ۴ که پراکندگی نقاط بهینه محلی را برای مسئله تقسیم‌بندی گراف، TSP و bin packing نشان می‌داد، مقایسه کنیم متوجه می‌شویم که در مسئله BBS نقاط بهینه محلی، بطور متقارنی حولِ بهینه سراسری قرار نگرفته‌اند و ساختار دره بزرگ نمی‌تواند برای این مسئله درست باشد، بنابراین عملکردِ ضعیف STAGE در آن توجیه می‌شود.

۵–۵ بحث

در این فصل، به بررسی تغییر الگوریتم جستجوی محلی و ساختار یادگیرنده در STAGE پرداختیم. نتایج نشان داد که در بهینه‌سازی تابع ارزیابی، الگوریتم جستجوی محلی Fihc از دیگر الگوریتم‌ها بهتر عمل می‌کند ضمن این که کارایی الگوریتم Sthc نیز بسیار به آن شبیه است. در بهینه‌سازی تابع هدف، با محدودیت روی تعداد گامهای برداشته شده، الگوریتم Bihc موفق‌تر از بقیه بوده است و این نشان دهندهٔ این مطلب است که برداشتن گامهای عمیق، در نهایت موجب بدست آمدن جواب بطور سریعتر و دقیق‌تر می‌شود. با محدودیت روی تعداد حالات دیده شده، الگوریتم Sthc بهتر از بقیه عمل می‌کند که این موضوع نقش مهم تابع ارزیابی را در فرایند جستجو مشخص می‌سازد. در تحلیل ساختارهای یادگیرنده، دیدیم که چندجمله‌ای درجه دوم، از دیگر ساختارهای یادگیرنده بهتر عمل می‌کند هر چند که تابع نمایی، هنگامی که تعداد حرکتهای مجاز کمتر باشد از آن بهتر عمل می‌کند. این نتیجه بسیار جالب توجه است و با اطلاعات قبلی ما در مورد نحوهٔ توزیع بهینه‌های محلی یک مسئله ترکیبیاتی (که در قالب قضیه فاجعه حد مرکزی بیان شده است)، همخوانی دارد.

در ادامهٔ فصل، به بررسی اصولی ساختار سطح هزینه برای مسائل بهینه‌سازی پرداختیم، موضوعی که تحت عنوان چشم‌انداز شایستگی در مقولهٔ الگوریتم‌های تکاملی مطرح شده است.

مدلهای مختلفی برای بیان ساختار سطح هزینه مسائل بهینه‌سازی بکار گرفته شده است و طیف این موضوع، علوم فیزیک و زیست‌شناسی را نیز در بر می‌گیرد. ما برای تکمیل تحلیل خود از STAGE، سطح هزینهٔ مسئله BBS را که STAGE در آن ضعیف عمل کرده است، مورد بررسی قرار دادیم و نتیجه بسیار جالب بود: ساختار دره بزرگ برای بهینه‌های محلی این مسئله درست نیست. این نتیجه، حدس ما را که STAGE در مسائلی خوب عمل می‌کند که ساختار دره بزرگ بر بهینه‌های محلی آنها حاکم باشد، تقویت می‌کند و ما را نهایتاً در انتخاب یک ساختار یادگیری مناسب، برای بدست آوردن نتایج دقیق‌تر، راهنمایی می‌سازد.

فصل ٦

نتیجه‌گیری و راه‌بردهای آتی

یادگیری تقویتی روشی برای طراحی سیستم‌های نرم‌افزاری مهیا می‌سازد که کارایی خود را بصورت خودکار با انجام تصمیم‌گیری بهبود بخشند، این روش بصورت زیر است:

- جمع‌آوری داده با مشاهدهٔ شبیه‌سازی کار.
- یادگیری آماری برای پیش‌بینی نتایج بلند مدت تصمیمات اتخاذ شده.
- استفاده از این پیش‌بینی در قالب تابع ارزیابی برای راهنمایی تصمیمات آتی.

در این رساله، ما الگوریتم‌های مختلف برای بهینه‌سازی شامل خانوادهٔ الگوریتم‌های تکاملی، فراالگوریتم بهینه‌سازی بر مبنای رفتار مورچگان، خانواده الگوریتم‌های بهینه‌سازی سراسری تکراری و محاسبات مولکولی را مرور کردیم. بویژه الگوریتم STAGE که بر مبنای یادگیری تقویتی برای بهینه‌سازی ترکیبیاتی پیشنهاد شده است را مورد توجه قرار دادیم و اثر ساختارهای یادگیری متفاوت و الگوریتم‌های جستجوی محلی مختلف را بر کارایی آن بررسی کردیم. می‌توان کارهای جدید انجام

شده در این پایان‌نامه را بصورت زیر خلاصه نمود:

- مروری بر روشهای جدید بهینه‌سازی در فصل ۲
- مروری بر ساختارهای مختلف یادگیری که ترکیب آنها با یادگیری تقویتی موفقیت‌آمیز بوده است (فصل ۳)
- تحلیل ساختار سطح هزینه مسئله bin-packing و تحقیق ساختار دره بزرگ برای نحوهٔ پراکندگی بهینه‌های محلی آن (فصل ۴)
- بررسی اثر الگوریتم‌های جستجوی محلی مختلف بر کارایی STAGE (فصل ۵)
- بررسی اثر ساختارهای یادگیری متفاوت بر کارایی STAGE (فصل ۵). همچنین در این فصل با پیشنهاد تابع نمایی به عنوان ساختار یادگیری، تأییدی بر قضیه فاجعه حد مرکزی بدست آوردیم.
- تحلیل ساختار سطح هزینه مسئله BBS (فصل ۵) به عنوان مسئله‌ای که STAGE در آن خوب عمل نمی‌کند و دیدیم که ساختار دره بزرگ برای آن درست نیست.

در ادامه این بخش، زمینه‌های تحقیقاتی آتی را که کار روی آنها امیدوارانه به نظر می‌رسد، ذکر می‌کنیم.

۶–۱ حل مسائل ILP توسط STAGE

هر مسئله ترکیبیاتی NP-Hard را می‌توان بصورت برنامه‌ریزی اعداد صحیح (ILP)[۱] نوشت و بطور کلی مسائل برنامه‌ریزی اعداد صحیح طیف گسترده‌ای را تشکیل می‌دهند، این مسائل در حالت کلی NP-Hard هستند و الگوریتمهایی بر اساس روشهای محکم ریاضی برای حل آنها وجود دارد (مانند cutting plane)[۲۸]. این مسائل را می‌توان به شکل کلی زیر نوشت:

$$\text{Obj: } f(x_1, x_2, \ldots, x_n)$$

subject to

$$g_1(x_1, x_2, \ldots, x_n) \geq 0$$

$$g_2(x_1, x_2, \ldots, x_n) \geq 0$$

$$\vdots$$

$$g_m(x_1, x_2, \ldots, x_n) \geq 0$$

$$\forall i, x_i \text{ is integer}$$

که تابع f همان تابع هدف بوده و خطی است، توابعِ محدودیت (یعنی g_iها) نیز همگی خطی هستند. می‌توان حل این مسئله را بصورت جستجو در یک فضای حالات در نظر گرفت. یک حالت عبارتست از یک مقداردهی معتبر به متغییرها، یعنی مقداری برای بردار X که $X = (x_1, x_2, \ldots, x_n)$ و در نامعادلات $g_1(X) \geq 0, \ldots, g_m(X) \geq 0$ صدق کند. برای تعریف ساختار همسایگی نیز می‌توان گفت همسایه‌های هر بردار X عبارتند از بردارهایی که با اضافه یا کم کردن یک واحد به هر یک از مؤلفه‌های بردار X بدست می‌آیند (هر حالت حداکثر $2 \times n$ حالتِ همسایه دارد)؛ بعبارتی شعاع

[۱] integer linear programming

همسایگی ۱ است و همسایه‌ها شامل فقط نقاط صحیح هستند.

به این صورت یک الگوریتم حریصانه از یک جواب اولیه شروع می‌شود و با در نظرگرفتن همسایه‌های هر حالت، جوابِ بهتر بعدی را در نظر می‌گیرد تا جاییکه در یک جواب بهینه محلی گیر کند. یک زمینه تحقیق جالب، بررسی فرایند جستجو در فضای حالات است (که در این حالت یک حجمِ محدب بسته یا باز است) و استفاده از الگوریتم STAGE برای راهنمایی فرایند جستجو به نقاط بهتر. در صورت استفاده از STAGE باید مشخصه‌های دیگر هر حالت (که مربوط به تابع هدف می‌باشند) را نیز برای ساختِ تابع ارزیابی در نظر گرفت. همچنین باید تدبیری اندیشیده شود تا بتوان یک جواب (حالت) تصادفیِ اولیه تولید نمود.

یک موضوع بسیار جالبتر این است که خود را به تابع هدف و توابع قید خطی، محدود نکنیم و توابع غیر خطی را هم لحاظکنیم؛ چگونگی عملکرد STAGE در این شرایط می‌تواند بسیار مهم باشد زیرا که دارای کاربردهای عملی در تحقیق در عملیات است، مثلاً در مسائل حمل و نقلی که دارای تابع هزینه غیر خطی می‌باشند.

۶—۲ حل مسأله رنگ‌آمیزی گراف توسط STAGE

یکی از مسائل مهم ترکیبیاتی، مسأله رنگ‌آمیزی گراف می‌باشد. مسأله به اینصورت است: گراف بدون جهت $G = (V, E)$ داده شده است، با حداقل تعداد رنگها گره‌های آن را طوری رنگ کنید که رأسهای همسایه دارای رنگهای متفاوت باشند؛ این مسأله NP-Complete می‌باشد [۱۹]. می‌توان الگوریتم STAGE را برای حل این مسأله بکار برد. برای این منظور، یک حالت (جواب) عبارتست از یک شیوهٔ رنگ‌آمیزی رئوس گراف بطوریکه قیود حفظ شوند. مشخصه‌های مناسب برای هر حالت می‌توانند

رنگهای لازم بدست آمده توسط یک الگوریتم حریصانه برای آن حالت، درجات بیشینه و کمینه بین رئوس گراف و واریانس تعداد رئوس موجود در هر رنگ باشند.

تنها مسألهٔ اصلی، اعمال یک الگوریتم حریصانه بصورت تکراری است. الگوریتم حریصانه را به اینصورت می‌توان طراحی کرد که یک جایگشت از رئوس را در نظر گرفت و سپس با شروع از اولین رأس، آنها را طوری با کمترین تعداد رنگ، رنگ‌آمیزی کنیم که قیود حفظ شوند. در گرافهای تصادفی که احتمال وجود یال بین هر دو رأس $\frac{1}{2}$ است، نشان داده شده است که بصورت میانگین این الگوریتم ۲ برابر تعداد رنگهای (کمینه) لازم را می‌دهد و در بدترین حالت، تعداد رنگهای برگشتی $O(n)$ خواهد بود [۱۴]. ابتکارهای [۲] مختلفی برای چیدن رئوس در یک دنباله [۳] داده شده است که یکی از معروفترین آنها متعلق به Culberson است [۱۴]: پس از رنگ‌آمیزی حریصانه رئوس یک گراف، رئوس همرنگ را کنار هم بگذارید بطوریکه رئوسی که دارای شماره رنگ بیشتری هستند در ابتدای دنباله قرار گیرند، سپس الگوریتم رنگ‌آمیزی حریصانه را روی این ترتیب از رئوس اجراکنید. به یک دلیل خیلی ساده می‌توان دید که امید داریم این ترتیب از رئوس نسبت به حالت قبل تعداد رنگهای کمتری مصرف کند: اگر فرض کنیم C_i مجموعه رئوسی باشد که دارای رنگ i هستند و C_j مجموعه رئوسی باشد که دارای رنگ j هستند، آنگاه $i > j$ به معنای این است که هر رأس در مجموعه C_i حداقل با یک رأس در مجموعه C_j در تداخل است (به عبارتی به آن یک یال دارد) ولی برعکس این موضوع درست نیست، یعنی ممکن است رئوسی در C_j باشند که با هیچ یک از رئوس موجود در C_i در تداخل نباشند.

با داشتن این الگوریتم حریصانه و انتخاب مشخصه‌های مناسب برای هر حالت، می‌توان

[۲] Heuristic

[۳] sequence

الگوریتم STAGE را برای حل این مسئله بکار برد و اثر استفاده از الگوریتمهای جستجوی محلی مختلف و استفاده از ساختارهای تخمین‌زننده مختلف را بر کارایی STAGE مورد مطالعه قرار داد و درستی نتایج بدست آمده در فصل۵ را تحقیق نمود.

٦—۳ استفاده از روشهای مدل کردن فازی در یادگیری تقویتی

ایده اصلی یادگیری تقویتی از مشاهده رفتار انسانها و دیگر حیوانات بدست آمده است: این موجودات بعد از انجام یک عمل با توجه به تشویق یا تنبیهی که از انجام عمل حاصل می‌شود، رفتاری را در خود تقویت یا تضعیف می‌نمایند [۴].

یادگیری تقویتی هنگامی پیش می‌آید که یک agant در یک محیط پویا می‌خواهد رفتاری را با آزمایش و خطا یاد بگیرد [۲۵] و کاری که در آن انجام می‌گیرد شباهت قابل توجهی به eponymous در روانشناسی دارد اما در جزئیات پیاده‌سازی و استفاده از کلمه تقویت، با آن تفاوت دارد.

با این دیدِ روانشناسانه به مسئله که از مشاهدهٔ رفتار یادگرفته شده توسط انسان و حیوانات بدست می‌آید، این انگیزه مطرح می‌شود که جزئیات مربوط به پیاده‌سازی نیز شبیه مکانیزم موجودات زنده باشد؛ در حقیقت خوبست که فرایند یادگیری رفتار نیز تا حد امکان به فرایند یادگیری موجودات زنده شباهت داشته باشد. روشهای یادگیری فازی سعی می‌کنند چنین کاری را انجام دهند؛ بویژه روش یادگیری فعال بر اساس فرایند یادگیری در مغز انسان شکل گرفته است. استفاده از روشهای یادگیری فازی در مبحث یادگیری تقویتی امیدوارانه به نظر می‌رسد و می‌تواند یک زمینه تحقیقاتی بسیار مهم تلقی شود.

---

[۴] این ایده برخاسته از نظریه کارکردگرایی در روانشناسی یادگیری است.

۶—۴ مطالب پایانی

از زمانی که ارتباط عمیق جستجوی ابتکاری در هوش مصنوعی و فرایند تصمیم‌گیری مارکوف شناخته شده است، شاخهٔ یادگیری تقویتی پیشرفت زیادی کرده است. امروزه الگوریتم‌های یادگیری تصمیم‌گیری از داده‌های بدست آمده از شبیه‌سازی، کاملاً شناخته شده هستند؛ هم برای مسائلی که فضای حالت آنها کوچک است و می‌توان تابع مقدار را دقیقاً نگه داشت و هم برای مسائلی که فضای حالت آنها بزرگ است (مانند بهینه‌سازی ترکیبیاتی) و باید تابع مقدار را بصورت فشرده در قالب یک تخمین‌زننده تابع نگهداری نمود[5]. با معرفی الگوریتم STAGE و با توجه به تحلیل آن، می‌توان انتظار داشت که جواب مسائل تصمیم‌گیری بسیار بزرگ در صنایع و علوم، بصورتی کارا بدست آیند که این خود باعث پیشرفت جامعه خواهد شد.

[5] در این زمینه، الگوریتم STAGE پیشرو محسوب می‌شود.

پیوست A

واژه‌نامه‌ی فارسی به انگلیسی

ابتکار، ابتکاری	heuristic
افزایشی	incremental
امکانپذیر	feasible
انتخاب	selection
بازخور	feedback
باناظر، باسرپرستی	supervised
برنامه‌ریزی پویا	dynamic programming
برنامه‌ریزی خطی	linear programming
بستر جذب	basin of attraction
بهینه	optimum
بهینه‌سازی	optimization
بی‌درنگ	realtime

پاداش .... reward

پایدار .... stationary

تابع هدف .... objective function

تابع مقدار .... value function

تپه‌نوردی .... hill climbing

ترکیب مجدد .... recombination

تصادفی .... random, stochastic

تصمیم‌گیری .... decision making

تقریب .... approximation

تقید .... exploitation

تکاملی .... evolutionary

تکراری .... iterative

جدول جستجو .... lookup table

جستجو .... search

جمعیت مورچه .... ant colony

جهش .... mutation

چشم‌انداز .... landscape

حالت .... state

حالت نهایی .... goal state

حریصانه .... greedy

داده‌ساختار .... data structure

دره بزرگ .... big valley

روال .... procedure

سراسری .... global

سیاست .... policy

شایستگی .... fitness

شعاعی .... radial

ضریب تنزیل .... discount factor

غنی .... rich

غیر همزمان .... asynchron

فاجعه .... catastrophe

فرامکعب .... hypercube

فضای حالت .... state space

قطعی .... deterministic

کار .... task

کاوش، کاویدن .... exploration

کنش .... action

محاسبات مولکولی .... molecular computing

محلی .... local

مدل گذر .... transition model

مسیر .... trajectory

مشخصه، مؤلفه .... feature

مصالحه .... trade off

مصنوعی .... artificial

مطلع .... informed

مقدار مرزی .... thredshold

نزول .... descend

همبستگی .... correlation

یادگیری تقویتی .... reinforcement learning

پیوست B

واژه‌نامه‌ی انگلیسی به فارسی

action .... کنش
ant colony .... جمعیت مورچه
approximation .... تقریب
artificial .... مصنوعی
asynchron .... غیر همزمان
basin of attraction .... بستر جذب
big valley .... دره بزرگ
catastrophe .... فاجعه
correlation .... همبستگی
data structure .... داده‌ساختار
decision making .... تصمیم‌گیری
descend .... نزول

قطعی .... deterministic
ضریب تنزیل .... discount factor
برنامه‌ریزی پویا .... dynamic programming
تکاملی .... evolutionary
تقید .... exploitation
کاوش، کاویدن .... exploration
امکانپذیر .... feasible
مشخصه، مؤلفه .... feature
بازخور .... feedback
شایستگی .... fitness
سراسری .... global
حالت نهایی .... goal state
حریصانه .... greedy
ابتکار، ابتکاری .... heuristic
تپه‌نوردی .... hill climbing
فرامکعب .... hypercube
افزایشی .... incremental
مطلع .... informed
تکراری .... iterative
چشم‌انداز .... landscape

برنامه‌ریزی خطی .... linear programming

محلی .... local

جدول جستجو .... lookup table

محاسبات مولکولی .... molecular computing

جهش .... mutation

تابع هدف .... objective function

بهینه‌سازی .... optimization

بهینه .... optimum

سیاست .... policy

روال .... procedure

شعاعی .... radial

تصادفی .... random

بی‌درنگ .... realtime

ترکیب مجدد .... recombination

یادگیری تقویتی .... reinforcement learning

پاداش .... reward

غنی .... rich

جستجو .... search

انتخاب .... selection

حالت .... state

فضای حالت .... state space

پایدار .... stationary

تصادفی .... stochastic

باناظر، باسرپرستی .... supervised

کار .... task

مقدار مرزی .... thredshold

مصالحه .... trade off

مسیر .... trajectory

مدل گذر .... transition model

تابع مقدار .... value function

کتاب‌نامه

[1] L. Adelman, "Molecular computation of solution to combinatorial problems", Science, 244, 1994.

[2] H. Ahrabian, E. Elahi, and A. Nowzari "A new DNA implementation of finite state machines", Ready to submit, 2002.

[3] Saeed Bagheri Shouraki, and Honda.

[4] Richard K. Belew, Ted Carson, Russell Impagliazzo, and Christos Papadimitriou, "Co-evolving Heuristics and the Problems They Solve", Draft version, EECS Department, University of California, Berekely, 1999.

[5] R. Bellman, "Dynamic Programming", Princeton University Press, 1957.

[6] D. Bertsekas, and J. Tisitsiklis, "Neuro-Dynamic Programming", Athena Scientific, Belmont, MA, 1996.

[7] K. D. Boese, “Models for Iterative Global Optimization”, Doctoral dissertion, Computer Science Department, University of California, Los Angeles, 1996.

[8] K. D. Boese, A. B. Khang, and S. Muddu, “A new adaptive multi-start technique for combinatorial global optimizations”, Operation Research Letters, 16, 1994.

[9] J. A. Boyan, “Learning Evaluation Function for Global Optimization”, Doctoral dissertion, Computer Science Department, Carnige Mellon University, 1998.

[10] J. A. Boyan, and A. W. Moore, “Generalization in reinforcement learning: Safely approximatin the value function”, Advances in Neural Information Processinf Systems 7, MIT Press, 1995.

[11] Chen Khong Tham, “Modular on-line function approximation for scaling up reinforcement learning”, Doctoral dissertion, Cambridge University, 1994.

[12] E. G. Coffman, M. R. Garey, and D. S. Johnson, “Approximation algorithms for bin-packing: a survey”, Approximation Algorithms for NP-Hard Problems, PWS publishing, 1996.

[13] R. Crites, and A. Barto, “Improving elevator performance using reinforcement learning”, Advances in Neural Information Systems 8, 1996.

[14] Joseph C. Culberson, “Iterated Greedy Graph Coloring and The Difficulty Landscape”, Technical Report 92-07, Computer Science Department, University of Alberta, 1992.

[15] Gretchen Dickie, “http://www.cs.montana.edu/ dickie/DNAComputing.html”, Accessed 2002.

[16] M. Dorigo, “Optmization, Learning, and Natural Algorithms” Doctoral dissertion, Dipartimento di Electronica e Informazione, Politecnico di Milano, Italy, 1992.

[17] M. Dorigo, and G. Di Carlo, “Ant Colony Optimization: A new meta-heuristic”, Evolutionary Computation, 1999.

[18] M. Dorigo, V. Maniezzo, and A. Colorni, “Positive feedback as a search strategy”, Technical Report, 91-016, Dipartmento di Electronica e Informazione, Politecnico di Milano, Italy, 1991.

[19] M. R. Garey, D. S. Johnson, “Computer and Intractability: A Guide to the Theory of NP-Comleteness”, W. H. Freeman, 1979.

[20] D. E. Goldberg, “Genetic Algorithms in Search, Optimization, and Machine Learning”, Addison-Wesley, Reading, Mass., 1989.

[21] Gholamreza Haffari, and Saeed Bagheri Shouraki,“Analysis of STAGE based on Solving Bin Packing Problem”, Accepted to The International Conference on Machine Learning and Applications (ICMLA’02), Las Vegas, 2002.

[22] J. Holland, “Adaption in natural and artificial systems”, University of Michigan Press, Ann Arbor, 1975.

[23] T. C. Jones,“Evolutionary Algorithms, Fitness Landscapes and Search”, Doctoral Dissertion, University of New Mexico, Albuquerque, 1995.

[24] Terry Jones, and Stephanie Forrest, “Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms”,Proc. of the Sixth Int. Conf. on Genetic Algorithms, San Francisco, CA, 1995.

[25] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore ,“Reinforcement Learning: A Survey”, Available at: http://researchindex.com, 1996, Accessed 2002.

[26] S. Kirkpatrick, C. D. Gelatt, and M. Vecchi, “Optimization based on Simulated Annealing”, Science, 220(4598):671-680, 1983.

[27] Mark William Shannon Land, “Evolutionary Algorithms with Local Search for Combinatorial Optimization”, Doctoral dissertion, Computer Science Department, University of California, San Diego, 1998.

[28] C. H. Papdimitriou, and K. Steiglitz, " Combinatorial Optimization : Algorithms and Complexity", Prentice-Hall, New York, 1982.

[29] M. L. Puterman, "Markov Decision Processes-Discrete Stochastic Dynamic Programming", John Wily & Sons Inc., New York 1994.

[30] R. S.Sutton, "Learning to predict by the methods of temporal differences", Machine Learning, 3, 1988.

[31] G. Tesauro, "TD-Gammon, a self teaching backgammon program, achieves master level play", Neural Computation, 6(2), 1994.

[32] J. N. Tsitsiklis, and B. Van Roy, "An analysis of temporal-difference learning with function approximation", Technical Report LIDS-P-2322, MIT, 1996.

[33] C. Watkins, "Learning From Delayed Rewards", Doctoral dissertion, Cambridge University, 1989.

[34] Edward D. Weinberger, and Peter F. Stadler,"Why Some Fitness Landscapes are Fractal", Available at http://researchindex.com, 1992, Accessed 2002.

[35] W. Zhang, "Reinforcement Learning for Job-Shop Scheduling" Doctoral dissertion, Oregon State University, 1996.

Combinatorial Optmization based on Evaluation Functions

Abstract

Recent researches have shown the success of using Reinforcement Learning in solving combinatorial optimization problems. The main idea of these methods is to learn (near) optimal evaluation functions to improve local searches and find (near) optimal solutions. STAGE algorithm, introduced by Boyan & Moore, is one of the most important algorithms in this area. In this dissertion, we focus on Bin-Packing problem, an important NP-Complete problem. We analyze cost surface structure of this problem and investigate "big valley" structure for the set of its local minima. The result gives reasons for STAGE's success in solving this problem. Then based on experimental results of Bin-Packing problem, we analyze the effectiveness of using different local search algorithms and different learning structures in STAGE. Additionally, the results confirm some of the theorems in Combinatorial Optimization.

Keywords: *Reinforcement Learning, Combinatorial Optimization, Evaluation Functions, Cost Surface Structure*

Combinatorial Optmization based on Evaluation Functions

by

Gholamreza Haffari

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science
in
Artificial Intelligence

Under supervision of
Dr. Saeed Bagheri Shouraki

July 2002

Computer Engineering Department

Sharif University of Technology

Tehran