دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

جزیره کیش، خلیج فارس، ایران ۱۹ الی ۲۱ اسفند ۱۳۸۶

انجمن کامپیوتر ایران
Computer Society of Iran

مقاله کامل

استفاده از روش مبتنی بر تعداد تناقضات برای حل مسائل ارضاء محدودیت توزیع‌شده

سمانه حسینی سمنانی
دانشجوی دکتری
گروه کامپیوتر، دانشگاه اصفهان، اصفهان، ایران
S_hosseiny@eng.ui.ac.ir

کامران زمانی‌فر
استادیار
گروه کامپیوتر، دانشگاه اصفهان، اصفهان، ایران
Zamanifar@eng.ui.ac.ir

چکیده: مسائل ارضاء محدودیت توزیع شده[۱] (DCSPs) گروه وسیعی از مسائل مطرح در دنیای واقعی را پوشش می‌دهند. این مسئله، DCSP را به زمینهٔ مهمی از تحقیقات تبدیل می‌کند. با در نظر گرفتن کلیهٔ تلاش‌هائی که اخیراً برای حل این دسته از مسائل انجام شده است می‌توان الگوریتم Asynchronous Partial Overlay (APO) را به عنوان یکی از موفق‌ترین این تلاش‌ها برشمرد. این الگوریتم برای حل یک مسئلهٔ ارضاء محدودیت ابتدا آن را به بخش‌های کوچک‌تری تقسیم می‌کند و سپس با انتخاب عامل‌هائی به عنوان عامل واسط سعی در حل متمرکز زیر مسئله‌های تولید شده دارد. این مقاله روش جدید و مؤثری را برای انتخاب این عامل‌های واسط معرفی می‌کند. به علاوه دو نسخهٔ گسترش یافته از الگوریتم APO، به نام‌های MaxCAPO و MaxCIAPO که از این استراتژی استفاده می‌کنند ارائه خواهند شد. ایدهٔ اصلی مطرح در این استراتژی این است که تعداد تناقضات (محدودیت‌های نقض شده) مرتبط با عامل‌های واسط تأثیر مستقیم بر کارآئی آن دارد. نتایج تجربی نشان می‌دهد که انتخاب عامل‌های واسط از میان عامل‌هائی که بیشترین تعداد تناقضات را دارند نه تنها منجر به کاهش قابل توجهی در پیچیدگی الگوریتم APO می‌شود، بلکه می‌تواند منجر به کاهش پیچیدگی الگوریتم‌های مشتق شده از APO، نظیر IAPO نیز بشود.

واژه های کلیدی: مسائل ارضاء محدودیت توزیع‌شده، خودمختاری، سیستم‌های چندعاملی، عملیات واسطه‌گری، هوش مصنوعی.

۱- مقدمه

تمام مسائلی که در آنها هدف یافتن مقادیر مناسب برای انتساب به متغیرهای توزیع‌شده است را می‌توان جزء مسائل ارضاء محدودیت توزیع‌شده به حساب آورد. در یک سیستم چند عاملی به هر عامل یک یا چند متغیر از میان متغیرهای توزیع‌شده منتسب می‌شود. وظیفهٔ این عامل خود مختار کنترل و مدیریت مقدار این متغیر می‌باشد. بسیاری از مسائل مطرح در دنیای واقعی و مسائل چند عاملی را می‌توان تحت این مدل در نظر گرفت، که از جمله می‌توان به مواردی نظیر، زمان‌بندی ملاقات توزیع‌شده[۲] [۱]، مسائل انتساب منابع توزیع شده[۳] [۲] و نگهداری واقعایعات چند عاملی[۴] [۳] اشاره کرد. با توجه به وسعت این دسته از مسائل، الگوریتم‌های متنوعی برای حل آنها از سال ۱۹۹۱ تاکنون ارائه شده است. برخی از این الگوریتم‌ها مانند الگوریتم عقب‌گرد نامتقارن[۵] (ABT) [۴] و الگوریتم Weak-Commitment نامتقارن[۶] (AWC) [۵]، کاملاً توزیع‌شده هستند در حالی که برخی دیگر از ترکیب روش‌های توزیع‌شده و متمرکز استفاده می‌کنند. یکی از موفق‌ترین الگوریتم‌های مطرح در دستهٔ دوم الگوریتم Asynchronous Partial Overlay (APO) است [۶] که توسط میلر[۷] و لزر[۸] ارائه شده است.

یکی از کلیدی‌ترین بخش‌های الگوریتم APO بخش انتخاب عامل واسط است، که در آن عامل‌هائی که در میان خود تناقضاتی را مشاهده می‌کنند، از میان خود عاملی را که بیشترین اولویت را دارد به عنوان عامل واسط انتخاب می‌کنند. وظیفهٔ عامل واسط حل متمرکز زیر مسئله است. در ادامه، الگوریتم APO، با قرار دادن زیر مسئله‌های حل شده در کنار هم کل مسئله را حل می‌کند. به دلیل حساسیت بخش انتخاب عامل واسط در الگوریتم APO، گزینش بهترین شیوه برای انتخاب این عامل می‌تواند تأثیر بسزائی بر کارآئی آن داشته باشد.

این مقاله یک استراتژی مبتنی بر تعداد تناقضات را معرفی می‌کند که با استفاده از آن عامل‌های مؤثرتر و نیرومندتری به عنوان عامل واسط انتخاب می‌شوند. ایدهٔ اصلی مطرح در این استراتژی این است که، عامل‌هائی که کامل‌ترین اطلاعات را نسبت به تناقضات موجود در زیر مسئله دارند می‌توانند به راه‌حل‌های بهتری برای حل زیر مسائل دست

پیدا کنند و این خود منجر به افزایش سرعت الگوریتم و کاهش تعداد پیغام‌های رد و بدل شده می‌شود.

در ادامهٔ این مقاله پس از معرفی رسمی مسئلهٔ ارضاء محدودیت، مروری بر الگوریتم‌های مطرح در این زمینه خواهیم داشت. در بخش ۳ علاوه بر معرفی دقیق الگوریتم APO، به ارائهٔ روش انتخاب عامل واسط مبتنی بر تعداد تناقضات خواهیم پرداخت و بر اساس آن الگوریتم MaxCAPO معرفی خواهد شد. بخش ۴ نیز علاوه بر معرفی الگوریتم IAPO، که نسخه‌ای گسترش یافته از الگوریتم APO است، الگوریتم MaxCIAPO را نیز که از روش جدید انتخاب عامل واسط استفاده می‌کند ارائه می‌کند. در نهایت در بخش ۵، شرائط و نتایج آزمایشات انجام شده برای مقایسهٔ الگوریتم‌های MaxCAPO و MaxCIAPO با الگوریتم APO ارائه شده است و در ادامه برخی راهکارهای ممکن برای پیشبرد اهداف این پژوهش معرفی شده است.

۲- پیشینه

در این بخش پس از معرفی رسمی مسائل ارضاء محدودیت توزیع شده، به بررسی مهم‌ترین الگوریتم‌های ارائه شده برای حل این دسته از مسائل می‌پردازیم.

۲-۱ مسائل ارضاء محدودیت توزیع شده

مسئلهٔ ارضاء محدودیت توزیع شده حالت توزیع شدهٔ مسئلهٔ ارضاء محدودیت استاندارد است. این محیط توزیع شده شامل تعدادی عامل هوشمند است که هر یک، یک یا چند متغیر را حمل و کنترل می‌کنند. مسئله ارضاء محدودیت توزیع شده اولین بار توسط سیکارو[۹]، یوکو[۱۰] و همکارانشان به صورت رسمی مطرح شد [۴، ۷]. مسئلهٔ ارضاء محدودیت که پایه و اساس مسئله ارضاء محدودیت توزیع شده است به صورت زیر تعریف می‌شود:

- مجموعه‌ای از متغیر ها: $V=\{x_1,...,x_n\}$
- مجموعه‌ای از دامنه‌های متناهی برای متغیرها: $D=\{D_1,...,D_n\}$
- مجموعه‌ای از محدودیت‌ها: $R=\{R_1,...,R_m\}$ که در آن هر $R_i(d_{i1},...,d_{ij})$ روی ضرب کارتزین $D_{i1} \times \ldots \times D_{ij}$ تعریف شده‌است. اگر مقدار منسب به این متغیرها محدودیت‌های موجود را ارضاء کند R_iها مقدار true را برمی‌گرداند و در غیر این صورت مقدار False برگردانده می‌شود.

هدف نهایی حل مسئلهٔ ارضاء محدودیت توزیع‌شده یافتن مقادیری برای انتساب به متغیرها است که کل محدودیت‌های موجود در R را ارضاء کنند. هر عامل سعی می‌کند نه تنها با ارضاء محدودیت‌های محلی خود بلکه با برقراری ارتباط با سایر عامل‌ها به منظور حل محدودیت‌های خارجی به این هدف نزدیک و نزدیک‌تر شود.

۲-۲ الگوریتم‌های مطرح برای حل DCSP

از آنجا که مسئلهٔ ارضاء محدودیت (CSP)، پایه و اساس مسئلهٔ ارضاء محدودیت توزیع‌شده (DCSP) است، بسیاری از الگوریتم‌های مطرح شده برای حل CSP نیز پایهٔ الگوریتم‌های مطرح برای حل DCSP هستند. مثلاً الگوریتم عقب‌گرد نامتقارن (ABT) نسخهٔ توزیع‌شدهٔ الگوریتم عقب‌گرد (BT) است و یا الگوریتم Weak-Commitment نامتقارن (AWC) نسخهٔ توزیع‌شدهٔ الگوریتم Weak-Commitment متمرکز است. جدول ۱ خلاصه‌ای از مهم‌ترین الگوریتم‌های مطرح برای حل DCSP را نشان می‌دهد. از میان این الگوریتم‌ها الگوریتم APO جدیدترین و موفق‌ترین آنها به حساب می‌آید.

در الگوریتم ABT هر عامل یک مقدار تصادفی از مقادیر موجود در دامنه‌اش را به متغیرش منتسب می‌کند. از آن‌جا که در این الگوریتم هر عامل یک شماره اولویت دارد، اگر مقدار منتسب به متغیر یک عامل با مقدار عامل‌های با اولویت بالاترش سازگار نباشد عامل کم اولویت‌تر باید مقدارش را تغییر دهد. به زبان دیگر، در چنین شرایطی عامل کم - اولویت‌تر تمام مقادیر ممکن برای انتساب به متغیرش را بررسی می‌کند و در صورت عدم وجود مقدار سازگاری، عقب‌گرد می‌کند.

الگوریتم موفق دیگر، AWC است که بسیاری از خصوصیات ABT را به ارث می‌برند اما از یک روش ابتکاری جدید به نام min-conflict سود می‌برد. با استفاده از این روش احتمال اتخاذ تصمیمات بد و انتخاب راه‌حل‌های نامناسب کاهش می‌یابد. کارآیی این دو الگوریتم با استفاده از انواع روش‌های ابتکاری و هوشمند ارائه شده به سرعت بهبود یافته‌است. ایدهٔ اصلی الگوریتم APO، انتخاب عامل‌های خاص به عنوان عامل واسط برای حل متمرکز زیر مسئله‌های محلی است. این عامل-های واسط سعی می‌کنند با راه‌اندازی یک جلسه واسطه‌گری اطلاعات ضروری و مورد نیاز خود را نسبت به زیر مسئله جمع‌آوری کنند.

گرچه الگوریتم APO نسبت به الگوریتم‌های پیشین خود برتری‌های قابل توجهی دارد، روش‌های هوشمند متعددی نیز اخیراً ارائه شده‌اند که سعی در برطرف کردن نقاط ضعف این الگوریتم‌ها دارند. برای مثال، میلر فعالیت‌هایی در زمینه گسترش APO برای اجرا در محیط‌های پویا [۸]، برای حل مسائل بهینه‌سازی [۹] و برای تأمین امنیت داده‌ها و حفظ خصوصی بودن آن‌ها [۱۰] انجام داده‌است. همچنین بنیش[۱۱] و سیده[۱۲] با تغییر استراتژی انتخاب عامل واسط، کارآئی APO را بهبود دادند. آن‌ها اثبات کردند که انتخاب عامل‌های با کمترین تعداد همسایه منجر به حل سریعتر مسئله خواهد شد [۱۱]. به علاوه یک استراتژی دوگانه غیر متمرکز متفاوت را نیز که بر مبنای ABT کار می‌کرد، ارائه کردند [۱۲]. همانطور که مشاهده می‌کنید تحقیقات متنوعی در این زمینه انجام شده‌است اما هیچ یک از آن‌ها تأثیر تعداد تناقضات را در پروسیجر انتخاب عامل واسط مورد بررسی قرار نداده‌اند.

جدول (۱): الگوریتم‌های مطرح برای حل DCSPs

	Single	multiple	partial
عقب گرد	Asynchronous BT	Secure DisCSP	Asynchronous IR
پیشرفت تکراری	Distributed Brakeout		iterative DB APO
ترکیبی	Asynchronous WC	variable ordering AWC agent-oriented AWC	

۳- استراتژی انتخاب عامل واسط مبتنی بر تعداد تناقضات

از آنجا که APO، پایه و اساس الگوریتم‌های ارائه شده در این مقاله است، در اینجا ابتدا مروری کلی بر این الگوریتم خواهیم داشت. این بخش با ارائهٔ الگوریتم MaxCAPO، ادامه خواهد یافت.

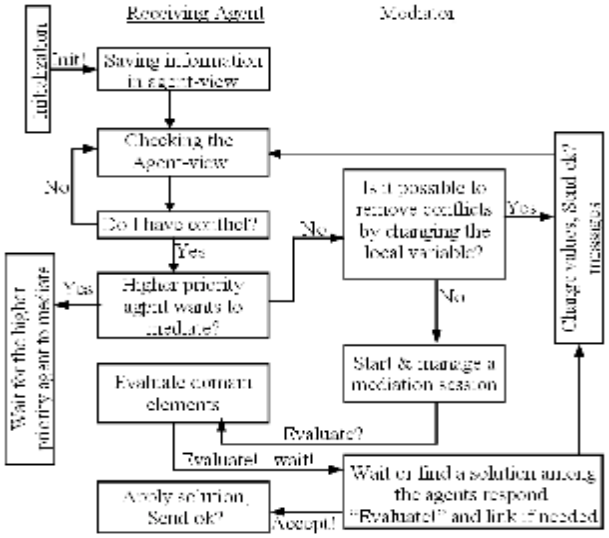
۳-۱ مروری برالگوریتمِ APO

شکل ۱، مراحل الگوریتم APO را به صورت خلاصه نشان می‌دهد. جدیدترین و کامل‌ترین نسخهٔ این الگوریتم در [۱۳] آمده است. APO با ارسال پیغام "init!"، که شامل اطلاعات اولیه راجع به فرستنده است، توسط هر عامل به عامل‌های همسایه‌اش آغاز می‌شود. عامل گیرنده اطلاعات دریافتی از پیغام "init!" را داخل لیستی به نام agent-view که نشان‌دهندهٔ دیدگاه هر عامل نسبت به سایر عامل‌ها است نگهداری می‌کند. سپس هر عامل با توجه به اطلاعات agent-view به بررسی وجود تناقض میان خود و سایر عامل‌ها می‌پردازد. در صورت وجود تناقض و عدم وجود عامل با اولویت بالاتر برای مدیریت جلسهٔ واسطه‌گری، عامل مذکور خود، نقش عامل واسط را خواهد پذیرفت.

در الگوریتم APO، اولویت عامل‌ها در مرحلهٔ اول بر اساس تعداد همسایگان آنها و در مرحلهٔ بعد بر اساس ترتیب الفبائی نام متغیرهای آنها تعیین می‌شود. پس از انتخاب عامل واسط، این عامل ابتدا سعی می‌کند با تغییر مقدار متغیر خود زیر مسئله را بدون راه‌اندازی جلسهٔ واسطه‌گری حل کند اما اگر هیچ یک از مقادیر موجود در دامنهٔ این متغیر کل محدودیت‌های موجود را ارضاء نکنند، عامل واسط اقدام به راه‌اندازی یک جلسهٔ واسطه‌گری می‌کند. این جلسه با ارسال یک پیغام "Evaluate?" توسط عامل واسط به عامل‌های مرتبط با آن آغاز می‌شود. هدف از ارسال این پیغام بدست آوردن اطلاعات کامل‌تری راجع به مقادیر جاری متغیرهای سایر عامل‌ها و مقادیر قابل پذیرش توسط آنها است.

اگر عامل گیرندهٔ پیغام "Evaluate?" در حال شرکت در جلسهٔ دیگر و یا منتظر دریافت پیغامی مشابه از عامل با اولویت بالاتر باشد، با پیغام "Wait!" پاسخ خواهد داد و در غیر این‌صورت اطلاعات درخواست شده را توسط ارسال پیغام "Evaluate!" در اختیار عامل واسط قرار می‌دهد. پس از دریافت تمامی پاسخ‌ها، عامل واسط با انجام یک جستجوی شاخه و قید [۱۴] سعی در یافتن راه‌حل مناسبی برای زیر مسئله دارد. در صورت یافتن چنین راه‌حلی، عامل واسط با ارسال پیغام "Accept!" برای عامل‌های شرکت کننده در جلسه و پیغام "Ok!" برای سایر عامل‌های مرتبط که در جلسه حضور نداشته‌اند آنها را از مقادیر پیشنهادی جدید برای متغیرها آگاه می‌کند و در غیر این‌صورت مسئله را غیر قابل حل اعلام خواهد کرد.

گاهی راه‌حل یافت شده توسط عامل واسط برای عامل‌های خارج از جلسهٔ واسطه‌گری ایجاد محدودیت‌های جدیدی می‌کند. در چنین شرائطی عامل واسط نام این عامل‌ها را به لیست همسایگان خود اضافه می‌کند و یک اتصال[۱۳] مجازی بین خود و آنها بوجود می‌آورد.



شکل (۱): الگوریتم APO

بنابراین عامل واسط این‌طور تصور خواهد کرد که به طریقی به عامل مربوطه متصل شده است. این کار باعث می‌شود که در ادامهٔ فرآیند حل مسئله، عامل واسط اشتباه انجام شده را تکرار نکند و از این به بعد هنگام بررسی راه‌حل‌های موجود برای حل مسئله، عاملی را که قبلاً با اتخاذ تصمیمی موجب بوجود آمدن تناقض برای آن شده است، در نظر بگیرد تا اشتباه گذشته را دوباره تکرار نکند. به این مرحله از الگوریتم APO مرحلهٔ برقراری اتصال[۱۴] می‌گویند.

۳-۲ استفاده از استراتژی انتخاب عامل واسط مبتنی بر تعداد تناقضات در الگوریتم MaxCAPO

همان‌طور که در بخش قبل اشاره شد، الگوریتم APO، عامل با بالاترین اولویت را به عنوان عامل واسط انتخاب می‌کند. برای این کار ابتدا به هر عامل یک عدد اولویت[۱۵] منتسب می‌شود که بر اساس تعداد همسایگان

آن عامل مشخص می‌شود. استفاده از این روش منجر به انتخاب عامل-هائی که بیشترین اطلاعات را نسبت به زیر مسئله دارند به عنوان عامل واسط می‌شود. بنابراین اولویت‌دهی به عامل‌ها از آنجا که تضمین کنندهٔ اتخاذ مناسب‌ترین تصمیمات برای زیر مسئله است از نقاط کلیدی و مهم الگوریتم APO به شمار می‌رود.

با وجود تمام مزایای این روش انتخاب عامل واسط، نقاط ضعفی نیز در مورد آن مطرح است، برای مثال بر اساس این روش انتخاب عامل واسط، بارها اتفاق می‌افتد که چندین عامل به دلیل داشتن تعداد همسایگان مشابه اولویت یکسانی داشته باشند. در چنین شرائطی الگوریتم APO، برای رهائی از مشکل بوجود آمده اقدام به انتخاب عامل واسط بر اساس ترتیب الفبائی نام متغیرهای منتسب به عامل‌ها می‌کند. به عنوان مثال اگر دوگره به نام‌های ND5 و ND2 تعداد همسایگان مشابهی داشته باشند، الگوریتم APO، ND5 را به عنوان عامل واسط انتخاب می‌کند، زیرا بر اساس ترتیب الفبائی حروف ND5 از اولویت بالاتری برخوردار است.

به نظر می‌رسد این روش اولویت دهی به عامل‌ها، یک روش تصادفی برای رهائی از گرهٔ بوجود آمده در چنین شرائطی است. ما معتقدیم با جایگزینی این روش تصادفی با روش هوشمند مبتنی بر تعداد تناقضات می‌توان عامل‌های مناسب‌تری را به عنوان عامل واسط انتخاب کرد و در نهایت منجر به حل کارآمدتر مسئلهٔ ارضاء محدودیت توزیع‌شده شد.

الگوریتم MaxCAPO، نسخه‌ای گسترش یافته از الگوریتم APO است که در تمام موارد بجز مرحلهٔ انتخاب عامل واسط از الگوریتم APO تبعیت می‌کند، اما برای انتخاب عامل واسط از روش مبتنی بر تعداد تناقضات استفاده می‌کند. در این الگوریتم در شرائطی که تعداد همسایگان چند عامل مساوی با یکدیگر باشند، عاملی که حداکثر تعداد تناقضات را دارد نقش عامل واسط را به عهده می‌گیرد. از آنجا که اطلاعات این عامل نسبت به تناقضات موجود در زیر مسئله اطلاعات کامل‌تر و جامع‌تری است، انتخاب این عامل واسط منجر به حل سریع‌تر زیر مسئله و ارسال و دریافت تعداد کمتری پیغام خواهد شد. بنابراین در الگوریتم MaxCAPO، اولویت عامل‌ها در مرحلهٔ اول بر اساس تعداد همسایگان آنها و در مرحلهٔ بعد بر اساس تعداد تناقضات هر عامل مشخص می‌شود.

البته در الگوریتم MaxCAPO نیز ممکن است با حالتی مواجه شویم که اولاً تعداد همسایگان چند عامل یکسان باشد و ثانیاً تعداد تناقضات موجود بین برخی از این عامل‌ها با عامل‌های همسایه‌شان یکسان باشد. اگرچه چنین حالتی بسیار به ندرت اتفاق می‌افتد، اما در صورت بروز چنین حالتی در MaxCAPO نیز بر اساس ترتیب الفبائی نام عامل‌ها تصمیم مناسب در جهت انتخاب عامل واسط اتخاذ خواهد شد. پس به این نتیجه می‌رسیم که در الگوریتم MaxCAPO، عامل واسط ابتدا بر اساس تعداد همسایگان عامل‌ها، در مرحلهٔ بعد بر اساس تعداد تناقضات عامل‌ها و در مرحلهٔ آخر بر اساس ترتیب الفبائی نام عامل‌ها انتخاب می‌شود.

۴- الگوریتم MaxCIAPO (Max-Conflict Inverse APO)

در این بخش علاوه بر معرفی الگوریتم IAPO، نسخهٔ گسترش یافته‌ای از این الگوریتم که از استراتژی مبتنی بر تناقض استفاده می‌کند به نام MaxCIAPO نیز ارائه می‌شود.

الگوریتم IAPO، نسخه‌ای گسترش یافته از الگوریتم APO است، که در آن عامل‌های واسط طوری انتخاب می‌شوند که منجر به تولید جلسات واسطه‌گری کوچک‌تری نسبت به APO شوند. APO با انتخاب عامل‌هائی که حداکثر تعداد تناقضات را دارند به عنوان عامل واسط به سمت تشکیل جلسات واسطه‌گری با بزرگترین سایز متمایل است، در حالی که بر اساس تحلیل‌های ریاضی گزارش شده در [۱۱]، با انتخاب جلسات واسطه‌گری کوچک‌تر می‌توان کارائی الگوریتم را بالاتر برد. میکائل بنیش[۱۶] و همکارانش معتقدند که دو منبع مهم پیچیدگی که مستقیماً روی پیچیدگی APO مؤثرند عبارتند از، پیچیدگی عملیات واسطه‌گری[۱۷] و پیچیدگی عملیات کنار هم قرار دادن زیر مسئله‌های حل شده[۱۸]. پیچیدگی عملیات واسطه‌گری به پیچیدگی عملیات جستجوی شاخه و قید مورد استفاده برای حل متمرکز زیر مسئله‌ها مربوط است.

با انتخاب جلسات واسطه‌گری بزرگتر، پیچیدگی عملیات جستجو به صورت چشمگیری افزایش می‌یابد، بنابراین با انتخاب جلسات واسطه-گری کوچک‌تر، پیچیدگی عملیات واسطه‌گری کاهش می‌یابد. از طرف دیگر، با این کار تعدا جلسات واسطه‌گری و در نتیجه پیچیدگی کنارهم قرار دادن آنها بیشتر می‌شود. بنابراین مهم‌ترین نکته در اینجا یافتن حالت تعادلی میان این دو نوع پیچیدگی است. بنیش و همکارانش ثابت کردند که با توجه به این دو نوع پیچیدگی، انتخاب عامل‌های واسط کوچک‌تر کارآئی را بهبود می‌بخشد.IAPO نسخه‌ای گسترش‌یافته از APO است که از این تئوری پیروی می‌کند. برای این کار اولویت عامل‌ها را دقیقاً معکوس اولویت عامل‌ها در APO تعریف می‌کند، به این معنا که اولویت عامل‌ها در IAPO نسبت معکوس با تعداد همسایگان هر عامل دارد. سایر قسمت‌های الگوریتم IAPO مشابه الگوریتم APO است.

هر دو الگوریتم APO و IAPO اولویت عامل‌ها را بر اساس تعداد محدودیت‌ها و بدون در نظر گرفتن تعداد محدودیت‌های نقض شده تعیین می‌کنند. در بخش قبل الگوریتم MaxCAPO معرفی شد. این الگوریتم روش انتخاب عامل واسط را در شرائطی که عامل‌ها تعداد همسایگان مساوی داشتند تغییر داد. در این بخش نیز مورد دیگری از استفاده از روش انتخاب عامل واسط مبتنی بر تعداد تناقضات مطرح شده است. همان‌طور که مشاهده می‌شود این استراتژی یک روش کلی

است که می‌تواند در مورد الگوریتم‌های مختلفی در زمینهٔ DCSP مورد استفاده قرار گیرد. گرچه ممکن است روی الگوریتم‌های مختلف به یک اندازه تأثیر مثبت نداشته باشد.

در این بخش می‌خواهیم استراتژی مشابه آنچه در مورد APO استفاده شده و منجر به تولید الگوریتم MaxCAPO شد را در مورد IAPO نیز اعمال کنیم. گرچه IAPO، استراتژی انتخاب عامل واسط را تغییر می‌دهد، در این استراتژی جدید نیز بارها اتفاق می‌افتد که چندین عامل اولویت یکسانی داشته باشند. دقیقاً مشابه روشی که در MaxCPAO بکار برده شد، در MaxCIAPO نیز در شرائطی که چندین عامل تعداد همسایگان یکسان و در نتیجه اولویت یکسانی دارند، عاملی که بیشترین تعداد تناقضات را دارد به عنوان عامل واسط انتخاب خواهد شد. تمام مزایائی که در بخش قبل در مورد الگوریتم MaxCAPO اشاره شد در اینجا نیز در مورد الگوریتم MaxCIAPO نیز صادق است. شکل ۲ چهار استراتژی انتخاب عامل واسط را که شامل استراتژی‌های مورد استفاده در APO و IAPO به علاوهٔ دو استراتژی معرفی شده در این مقاله است را نشان می‌دهد.

APO	A_i = | neighborhood (i) | If several agents have the same P_i then Mediator = the agent that have the highest lexicographical order

Max Conflict APO (MaxCAPO)	A_i = | neighborhood (i) | C_i = | conflicts number (i) |
	If several agents have the same P_i then
	Mediator = the agent with the most C_i
	If several agents have the same C_i then
	Mediator = the agent that have the
	highest lexicographical order

Inverse APO (IAPO)	$A_i = \frac{1}{\| \text{neighborhood (i)} \|}$ If several agents have the same P_i then Mediator = the agent that have the highest lexicographical order

Max Conflict Inverse APO (MaxCIAPO)	$P_i = \frac{1}{\| \text{neighborhood (i)} \|}$ C_i = conflicts number (i)
	If several agents have the same P_i then
	Mediator = the agent with the most C_i
	If several agents have the same C_i then
	Mediator = the agent that have the
	highest lexicographical order

شکل (۲): چهار استراتژی انتخاب عامل واسط

۵- ارزیابی تجربی

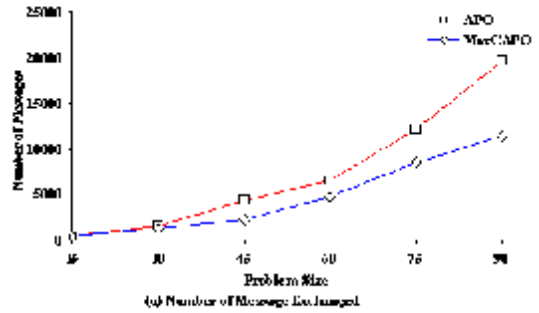
۵-۱ محیط و تنظیمات آزمایش

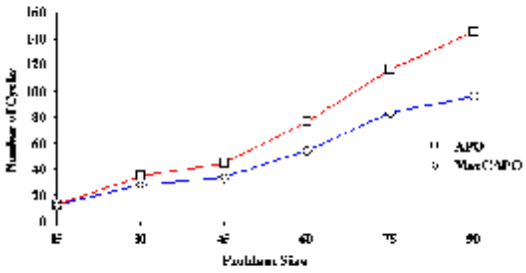
این بخش با ارائهٔ نتایجی که حاصل از شبیه‌سازی الگوریتم‌های APO، MaxCAPO و MaxCIAPO در شبیه‌ساز Farm [۱۵] است، به مقایسهٔ الگوریتم‌های MaxCAPO و MaxCIAPO با APO و IAPO می‌پردازیم.

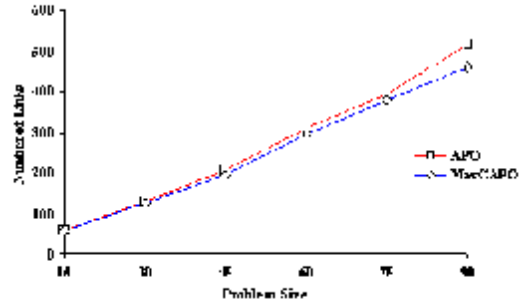
در آزمایش‌های انجام شده در این بخش سعی کردیم پارامترهای انتخابی را طوری انتخاب کنیم که با آزمایشات انجام شدهٔ قبلی کاملاً متناسب باشد و لذا به راحتی بتوان نتایج حاصل از این تحقیق را با نتایج قبلی مورد مقایسه قرار داد.

محیط شبیه‌سازی انتخاب شده شبیه سازی به نام Farm است [۱۶] که به زبان Java نوشته شده است و امکانات نسبتاً کاملی برای شبیه-سازی یک محیط چند عاملی را فراهم می‌کند.

در این آزمایش برای ایجاد سازگاری با نتایج گزارش شده از APO، تعداد گره‌های گراف‌های تولید شده در مسئلهٔ رنگ‌آمیزی گراف برای مقایسهٔ الگوریتم‌های APO و MaxCAPO به ترتیب ۱۵ ، ۳۰ ، ۴۵ ، ۶۰ ، ۷۵ و ۹۰ عدد با چگالی ۲/۳ و برای مقایسهٔ الگوریتم‌های IAPO و MaxCIAPO به ترتیب ۱۵، ۳۰، ۴۵، ۶۰ عدد با چگالی ۲/۷ در نظر گرفته شد. به این ترتیب تعداد یال‌های تولید شده در آزمایش اول ضریب ۲/۳ از تعداد گره‌های موجود در همان گراف خواهد بود و در آزمایش دوم به تعداد ضریب ۲/۷ آن در نظر





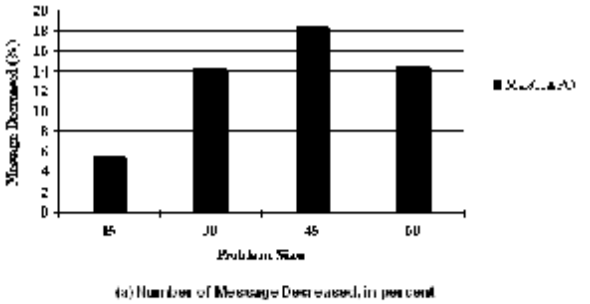


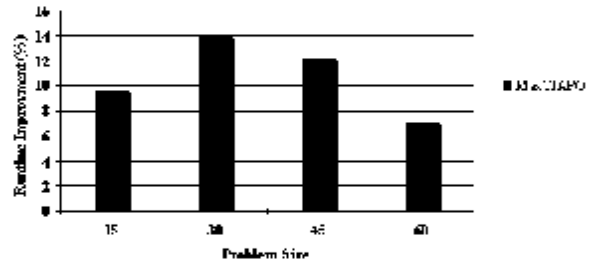
شکل (۳): مقایسهٔ تعداد پیغام‌های رد و بدل شده، زمان اجرا و تعداد اتصالات تولید شده برای حل نمونه‌های قابل حل مسئلهٔ D3GC چگالی متوسط با تعداد متنوعی از متغیرها در دو الگوریتم APO و MaxCAPO

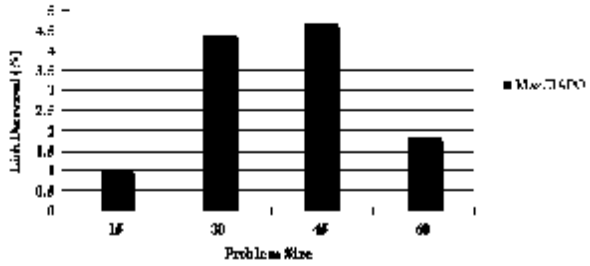
گرفته شده است. به طور کلی ۶۰۰ نمونه گراف کاملاً تصادفی در آزمایش اول و ۴۰۰ گراف تصادفی در آزمایش دوم تولید و توسط هر دو الگوریتم حل شدند.

۵-۲ نتایج آزمایش

شکل ۳ نتایج حاصل از اجرای الگوریتم‌های MaxCAPO و APO را که حاصل انجام آزمایشات مطابق با پارامترهای ذکر شده است، ارائه می‌کند. نکتهٔ قابل توجه مطرح در این قسمت، که دلیلی بر صحت و دقت آزمایشات انجام شده است این است که نتایج







شکل (۴): درصد بهبود الگوریتم MaxCIAPO در پیغام‌های رد و بدل شده، زمان اجرا و تعداد اتصالات تولید شده برای حل نمونه‌های قابل حل مسئلهٔ D3GC چگالی ۲/۷ با تعداد متنوعی از متغیرها در مقایسه با الگوریتم IAPO

حاصل از اجرای الگوریتم APO با آنچه قبلاً در مورد آن گزارش شده بود [۱۳] مطابقت دارد. این موضوع علاوه بر تأیید گزارشات ارائه شده در مورد APO، صحت شبیه‌سازی در این تحقیق را نیز مورد تأیید قرار خواهد داد و نشان می‌دهد بهبود حاصل شده در MaxCAPO تنها حاصل دخالت دادن تعداد تناقضات در استراتژی انتخاب عامل واسط بوده است و نه چیز دیگر. شکل ۴ نیز نشان‌دهندهٔ درصد بهبود انواع پارامترها در الگوریتم MaxCIAPO در مقایسه با IAPO است. همانطور که در شکل ۳ و ۴ دیده می‌شود، الگوریتم‌های MaxCAPO و MaxCIAPO به ترتیب بهبود قابل توجهی را نسبت به الگوریتم‌های APO و IAPO نشان می‌دهد و این بهبود تمام پارامترهای : "تعداد

پیغام‌ها" ، "زمان اجرا" و "تعداداتصالات تولید شده" را پوشش می‌دهد. از سوئی دیگر این بهبود در گراف‌های تولید شده در تمام اندازه‌ها دیده می‌شود. به این معنا که این بهبود تمام گراف‌های تولید شده با انواع اندازه‌ها را نیز پوشش می‌دهد. نکتهٔ مهم دیگری که در اینجا مطرح می‌شود این است که با افزایش اندازهٔ مسئله، تفاوت میان نتایج حاصل از الگوریتم‌های MaxCAPO و APO بیشتر می‌شود. به زبانی دیگر، هر چه مسئله بزرگتر و مشکل‌تر می‌شود، تأثیر در نظر گرفتن تعداد تناقضات موجود بیشتر و عمیق‌تر می‌شود.

۶- نتیجه‌گیری و راه‌کارهای پیشنهادی

در این مقاله، نسخه‌های جدیدی از الگوریتم APO، به نام‌های MaxCAPO و MaxCIAPO ارائه شد. با ارائهٔ این الگوریتم‌ها نقش تعداد تناقضات موجود در انتخاب عامل واسط مورد بررسی قرار گرفت. نتایج تجربی حاصل شده نشان داد که عامل‌های واسطی که بیشترین تعداد تناقضات را دارند می‌توانند مسئله را سریعتر و با رد و بدل کردن تعداد کمتری پیغام حل کنند. برای نشان دادن کامل بودن این ایده، صدها نمونه گراف از مسئلهٔ D3GC در اندازه‌های مختلف به صورت تصادفی تولید شده و مورد آزمایش قرار گرفت. سپس در هر مورد میانگین نتایج بدست آمده محاسبه شد. این نتایج نشان داد که در تمام موارد الگوریتم MaxCAPO بهتر از APO ، بهترین الگوریتم شناخته شده در این زمینه تاکنون، عمل می‌کند. همچنین الگوریتم MaxCIAPO بهتر از الگوریتم IAPO و همچنین APO عمل می‌کند.

از آنجا که DCSP محدودهٔ وسیعی از مسائل را پوشش می‌دهد و اینکه APO بهترین الگوریتم در این زمینه است، الگوریتم‌های ارائه شده در این پژوهش، با بهبود الگوریتم APO می‌توانند کمک شایانی در جهت حل دستهٔ وسیعی از مسائل مطرح در زمینهٔ هوش مصنوعی و مسائل چندعاملی انجام دهند.

استفاده از واحدهای اندازه‌گیری دیگر، برای محاسبهٔ پیچیدگی محاسباتی APO مانند بررسی‌های محدودیت غیر همروند که اخیراً توسط میزلز[۱۹] و همکارانش [۱۷] ارئه شده است، به عنوان یکی از کارهای مناسب در تحقیقات آتی در ادامهٔ این تحقیق قابل انجام است.

استفاده از استراتژی انتخاب عامل‌های واسط با حداکثر تعداد تناقضات در سایر نمونه‌های مسائل ارضاء محدودیت توزیع‌شده مانند مسائل ارضاء محدودیت با محدودیت‌های دودوئی تصادفی[۲۰] و سایر دامنه‌ها مثل sensorDCSP [۱۸] ، می‌تواند در جهت گسترش استراتژی ارائه شده سودمند باشد. همچنین تغییر سایر بخش‌های الگوریتم APO به جز بخش انتخاب عامل واسط که در این تحقیق مورد بررسی قرار گرفت نیز می‌تواند به عنوان زمینه‌ای دیگر برای تحقیقات آتی در نظر گرفته شود. برای مثال می‌توان الگوریتم جستجوی مورد استفاده در الگوریتم APO را با سایر الگوریتم‌های جستجوی کارآمدتر تعویض کرد.

و در پایان، ارائهٔ اثباتی ریاضی بر آنچه در این مقاله ارائه شد می‌تواند گامی سودمند در جهت تأیید تئوری آن به شمار آید.

۷- مراجع

[1] Modi, A P. J., Veloso, M., Smith, S., Cmradar, J. Oh., *A personal assistant agent for calendar management*. Agent Oriented Information Systems (AOIS), 2004.

[2] Conry, S. E., Kuwabara, K., Lesser, V. R., Meyer R. A., *Multistage negotiation for distributed constraint satisfaction.*, International Journal of IEEE Transactions on Systems, Man and Cybernetics 21(6), pp.1462–1477, 1991.

[3] Huhns, M. N., Bridgeland, D. M., *Multi-agent truth maintenance.*, International Journal of IEEE Transactions on Systems, Man and Cybernetics 21(6), pp. 1437–1445, 1991.

[4] Yokoo, M., Durfee, E. H., *Distributed constraint Satisfaction for formalizing distributed problem solving.*, 12th IEEE International Conference on Distributed Computing Systems, 1, pp. 614–621, 1992.

[5] Yokoo, M., *Asynchronous weak-commitment search for solving distributed constraint satisfaction problems.*, The First International Conference on Principles and Practice of Constraint Programming, pp.88–102, 1995.

[6] Mailler, R., Lesser, V., *Using cooperative mediation to solve distributed constraint satisfaction problems.*, Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), New York, pp. 446–453, 2004.

[7] Sycara, K., Roth, S. F., Sadeh N., Fox, M. S., *Distributed constrained heuristic search.*, International Journal of IEEE Transactions on Systems, Man and Cybernetics, 21(6), pp. 1446–1461, 1991.

[8] Mailler, R., *Comparing two approaches to dynamic, distributed constraint satisfaction.*, Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005),pp. 1049–1056, 2005.

[9] Mailler, R., Lesser, V., *Solving distributed constraint optimization problems using cooperative mediation.*, Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), pp.438–445, 2004.

[10] Mailler, R., *Solving distributed CSPs using dynamic partial centralization without explicit constraint passing.*, Second Workshop on the Challenges in the Coordination of Large Scale Multi-Agent Systems (LSMAS 2005), 2005.

[11] Benish, M., Sadeh, N., *Effect of mediator selection strategy for distributed constraint satisfaction*, Workshop on Distributed Constraint Reasoning (DCR)., 2005.

[12] Benish, M., Sadeh, N., *Examining distributed constraint satisfaction problem (DCSP) coordination tradeoffs.* International Conference on Automated Agents and Multi-Agent Systems (AAMAS), 2006.

[13] Mailler, R., Lesser, V., *Asynchronous partial overlay: a new algorithm for solving distributed constraint satisfaction problems.,* Journal of Artificial Intelligence Research (JAIR) 25, pp. 529–576, 2006.

[14] Freuder¸ E. C., Wallace, R. J., *Partial constraint satisfaction,* Artificial Intelligence,58 (1–3),pp. 21–70, 1992.

[15] Horling, B., Mailler, R., Lesser, V., *Farm: a scalable environment for multi-agent development and evaluation.,* In Second International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003), 2003.

[16] Horling, B., Mailler, R., Lesser, V., *The Farm Distributed Simulation Environment.*, Computer Science Technical Report 2004-12, Number 2004-12, 2004.

[17] Meisels, A., Kaplansky, E., Razgon, I., Zivan, R., *Comparing performance of distributed constraints processing algorithms.,* DCR Workshop at AAMAS'02,2002.

[18] Fernandez, C., Bejar, R., Krishnamachari, B., Gomes, C., Selman, B., Distributed Sensor Networks: A Multiagent Perspective, chap. Communication and Computation in Distributed CSP Algorithms, pp.299–317. Kluwer Academic Publishers, 2003.

زیرنویس‌ها

[1] Distributed Constraint Sati faction Problem

[2] Distributed Meeting Scheduling

[3] Distributed Resource Allocator

[4] Multi – agent Truth Maintainance

[5] Asynchronous Backtracking

[6] Asynchronous Weak-Commitment

[7] Mailler

[8] Lesser

[9] Sycaro

[10] Yokoo

[11] Benish

[12] Sadeh

[13] link

[14] Linking Step

[15] Priority Number

[16] Michael Benish

[17] Mediation Process

[18] Overlay Process

[19] Meisels

[20] Random binary DCSP