ارائه راهکاری جهت مقابله با مشکل انفجار فضای حالت در سیستم‌های تبدیل گراف با استفاده از الگوریتم‌های پرندگان و جستجوی گرانشی

مریم مرادی[۱]، دانشجوی کارشناسی ارشد، رزا یوسفیان[۲]، دانش‌آموخته کارشناسی ارشد، وحید رافع[۳]، استادیار

۱- دانشکده فنی و مهندسی - گروه مهندسی کامپیوتر - دانشگاه اراک - اراک - ایران - m.65moradi@gmail.com
۲- موسسه آموزش عالی غیرانتفاعی فیض الاسلام - اصفهان - ایران - rosa8a81@yahoo.com
۳- دانشکده فنی و مهندسی - گروه مهندسی کامپیوتر - دانشگاه اراک - اراک - ایران - v-rafe@araku.ac.ir

چکیده: وارسی مدل، یک روش خودکار و راهکاری مناسب به‌منظور درستی‌یابی سیستم‌های نرم‌افزاری مطمئن است. در این سیستم‌ها، نمی‌توان ریسک بروز خطا را حتی در فرآیند تست پذیرفت و لذا لازم است فرآیند درستی‌یابی، قبل از پیاده‌سازی و در سطح مدل انجام شود. سیستم‌های تبدیل گراف، از پرکاربردترین سیستم‌های مدل‌سازی رسمی و راهکاری مناسب به‌منظور مدل‌سازی و وارسی سیستم‌های پیچیده هستند. اما این سیستم‌ها در فرآیند وارسی مدل از مشکل انفجار فضای حالت رنج می‌برند که در صورت گسترده‌بودن ابعاد مسئله و لذا بزرگ‌شدن فضای حالت مدل، سیستم با کمبود حافظه مواجه می‌شود. لذا هدف از این پژوهش، پیشنهاد راهکاری جهت مقابله با این مشکل در فرآیند وارسی سیستم‌های تبدیل گراف است. راهکارهای ارائه‌شده، به‌جای تولید کل فضای حالت، آن را در جهت رسیدن به یک حالت خطا به‌طور مثال بن‌بست، هدایت می‌کنند. راهکار پیشنهادی بر مبنای الگوریتم پرندگان طراحی شده و برای جلوگیری از مشکل به دام افتادن در بهینه‌های محلی که مشکل اصلی این الگوریتم است، با الگوریتم جستجوی گرانشی که دارای قدرت خوبی در جستجوی محلی است، ترکیب شده است. در نهایت به‌منظور ارزیابی نتایج راهکارهای ارائه‌شده، این راهکارها در ابزار Groove- از ابزارهای مدل‌سازی تبدیل گراف-پیاده‌سازی شده‌اند.

واژه‌های کلیدی: وارسی مدل، سیستم تبدیل گراف، انفجار فضای حالت، الگوریتم پرندگان، الگوریتم جستجوی گرانشی.

An Approach to Cope with the State Space Explosion Problem in Graph Transformation Systems Using PSO and GSA Algorithms

M. Moradi[1], R. Yousefian[2], V. Rafe[3]

1, 3 - Faculty of Engineering, Department of computer engineering, Arak University, Arak, Iran
2-Feizol Eslam Institution, Khomeinishahr, Isfahan- Iran

Abstract: One of the best solutions to verify software systems (especially critical systems) is model checking in which all reachable states are generated from an initial state and the generated state space is searched to find errors or some desirable patterns. However, the problem for many real and complex systems is the state space explosion in which model checking cannot generate all the states. In this paper, a solution is presented to cope with this problem in systems modeled by graph transformation. Since meta-heuristic algorithms are proper solutions to search in very large problem spaces, they employed in this research to find errors (e.g. deadlocks) in systems which cannot be verified through existing model checking approaches because of state space explosion problem. To do so, a Particle Swarm optimization (PSO) algorithm is empployed to consider only a subset of states (called population) in each step of the algorithm. To avoid local optima problem, PSO is combined with Gravitational Search Algorithm (GSA). Our proposed approach is implemented in GROOVE –a toolset for designing and model checking graph transformation system-. The experiments show better results in terms of accuracy, speed and memory usage in comparison with the existing approaches.

Keywords: Model checking, deadlock, particle swarm optimization, PSO, gravitational search algorithm, graph transformation system, state space explosion

تاریخ ارسال مقاله: ۹۳/۰۱/۱۸
تاریخ اصلاح مقاله: ۹۳/۰۴/۲۸
تاریخ پذیرش مقاله: ۹۳/۰۶/۳۰
نام نویسنده مسئول: وحید رافع
نشانی نویسنده مسئول: ایران - اراک - سردشت - پردیس دانشگاه اراک - دانشکده فنی و مهندسی - گروه کامپیوتر

۱- مقدمه

وابستگی زندگی بشر به کامپیوتر،روزبه‌روز و به سرعت در حال گسترش است. با پیشرفت علم بشر در همه زمینه‌ها، نیاز به سرعت و دقت به خصوص در حوزه‌هایی که پیچیدگی بالایی دارند و انجام کارها و محاسباتی که در آن‌ها به سرعت و دقتی فراتر از توانایی‌های معمول بشر نیاز دارد، اهمیت یافته و به‌تبع آن، صحت سیستم‌های کامپیوتری، تبدیل به یکی از دغدغه‌های کارشناسان حوزه نرم‌افزارگردیده است. اساساً تولید سیستم‌ها با بروز خطاهایی همراه است. برخی از این خطاها تأثیرات بحرانی روی زندگی بشر ندارند، اما گاهی اوقات سیستم موردبحث، یک سیستم ایمنی-بحرانی است. در این سیستم‌ها حتی برای مرحله تست نیز نمی‌توان ریسک بروز خطا را حتی برای یک‌بار پذیرفت. وجود یک خطای کوچک برنامه‌نویسی در این سیستم‌ها می‌تواند منجر به وارد آمدن خسارات مالی و جانی جبران‌ناپذیر شود [۱]. در این‌گونه موارد، از فرآیند وارسی مدل برای درستی‌یابی مسئله مفروض استفاده می شود که پیش‌نیاز آن، توصیف رسمی سیستم است. وارسی‌کننده مدل از طریق بررسی تمام حالت‌ها و انتقال‌ها مشخص می‌کند که مسئله مدل‌شده در این سیستم آیا خصوصیت موردنظر را برآورده می‌کند یا خیر؟

یک روش بسیار پرکاربرد برای مدل‌سازی، استفاده از سیستم تبدیل گراف است که برای توصیف و مدل‌سازی از پایه ریاضیاتی مبتنی بر گراف استفاده می‌کند. Groove [۲]،یکی از ابزارهای رایج برای درستی‌یابی مسائل مدل شده در سیستم تبدیل گراف است که پس از مدل‌کردن و از طریق تولید کل فضای حالت مدل و سپس بررسی تمام حالت‌ها، عملیات وارسی را انجام می‌دهد. اما این روند از مشکل انفجار فضای حالت رنج می‌برد. فضای حالت، مجموعه حالت‌های محتملی است که با شروع از حالت اولیه می‌توان به آن‌ها رسید. تعداد این حالت‌ها بسته به ابعاد مسئله می‌تواند بسیار زیاد و روند رشد آن نمایی باشد. فضای حالت گاهی آن‌قدر بزرگ است که حافظه سیستم قادر به تولید، نگه‌داری و بررسی همه حالت‌ها نیست لذا این فرآیند به دلیل کمبود حافظه، بدون نتیجه و ناموفق خواهد بود. لذا راه‌حل‌های متعددی برای مقابله با این مشکل ارائه شده است.

تاکنون از روش‌های کلاسیکی همچون درستی‌یابی ترکیبی[۱] [۴،۳]، کاهش بر مبنای ترتیب جزئی[۲] [۳، ۵ و ۶] و کاهش تقارنی[۳] [۹-۷] به‌منظورکاهش اندازه‌ی فضای حالت مدل، استفاده شده است. اگرچه این روش‌ها توانسته‌اند مشکل انفجار فضای حالت را تا حدی بهبود بخشند ولی این مشکل همچنان به‌عنوان یک مشکل بنیادی در سیستم‌های وارسی مدل باقی مانده است و این سیستم‌ها همچنان از مشکل کمبود حافظه و سرعت پایین رنج می‌برند. در سال‌های اخیر، روش‌های هوش مصنوعی نیز به‌منظور رسیدن به راه‌حل یا افزایش سرعت همگرایی به‌طرف راه‌حل‌ها در سیستم های وارسی مدل، مورد توجه قرار گرفته‌اند [۱۵-۱۰].

روش پیشنهادی ارائه‌شده، با به‌کارگیری روش‌های هوشمند، ازجمله الگوریتم پرندگان(PSO)[۴] [۱۶]، در مسائلی با فضای حالت بسیار بزرگ و پیچیده، سعی بر کشف خطا دارد. قابل‌ذکر است، الگوریتم ارائه شده، متمرکز بر کشف بن‌بست در مسائل مدل‌شده در سیستم تبدیل گراف است. اما ازآنجاکه الگوریتم پرندگان از مشکل به دام افتادن در بهینه های محلی رنج می‌برد، با ترکیب آن با الگوریتم جستجوی گرانشی(GSA)[۵] [۱۷] سعی شده است دقت راهکار پیشنهادی تا حد ممکن، مورد قبول باشد.

در نهایت با تجهیز ابزار Groove به راهکار پیشنهادی، تلاش شده است گامی مؤثر در جهت بهبود فرآیند وارسی مدل قبل از تولید سیستم برداشته شود. با بررسی نتایج، مشاهده می‌شود که الگوریتم‌های ارائه شده در مقایسه با راهکارهای موجود، توانسته‌اند نتایج قابل قبولی را به دست آورند.

در ادامه مقاله، در بخش ۲ به کارهای مرتبط که در این زمینه انجام شده است اشاره می‌شود. در بخش ۳ بر رویکرد ارائه‌شده نگاهی مختصر می‌اندازیم. مفاهیم اولیه در راستای پژوهش انجام‌شده ازجمله؛ وارسی مدل، سیستم تبدیل گراف، الگوریتم پرندگان و الگوریتم جستجوی گرانشی در بخش ۴ به صورت اجمالی مطرح شده اند. در بخش ۵ نحوه ترکیب وارسی مدل با الگوریتم‌های پیشنهادی و روند عملکرد آن‌ها مورد بررسی قرار گرفته است. روش پیاده‌سازی و ارائه نتایج به‌دست‌آمده از راهکارهای پیشنهادی و ارزیابی آن‌ها در بخش ۶ نشان داده شده است. بخش ۷ نیز شامل نتیجه‌گیری و پیشنهادها برای کارهای آتی است.

۲- کارهای مرتبط

امروزه برای کشف خطاهایی همچون بن‌بست[۶] درسیستم‌های ایمنی–بحرانی، از وارسی مدل استفاده می‌شود. به‌طوری‌که پس از مدل‌شدن سیستم، کل فضای حالت آن تولید شده و تمام حالت ها بررسی می‌شوند. این روش با وجود مزایایی که دارد، مهم‌ترین مشکل آن انفجار فضای حالت است؛ به این صورت که با بزرگ شدن ابعاد مسائل، فضای حالت به صورت نمایی رشد می‌کند و با گسترش فضای حالت، فرآیند وارسی با مشکل مواجه می شود. لذا پژوهش‌های مختلفی در راستای مقابله با این مشکل ارائه شده است.

ازجمله راه‌حل‌های پیشنهاد شده با استفاده از روش‌های کلاسیک جهت نیل به این هدف، می‌توان به راه‌حل‌های مبتنی بر کاهش اندازه فضای حالت مدل [۸، ۱۸ و ۱۹] و راه‌حل های مبتنی بر اندوختن حافظه [۲۳-۲۰] اشاره کرد. این پژوهش‌ها، برای کاهش اندازه فضای حالت از روش‌هایی چون درستی‌یابی ترکیبی [۳، ۴]، کاهش برمبنای ترتیب جزئی [۳، ۵ و ۶] و کاهش تقارنی[۹-۷] استفاده می‌کنند. برای اندوختن حافظه نیز از الگوریتم هایی جهت کاهش حافظه لازم برای ذخیره سازی حالت‌ها استفاده می شود. این روش‌ها با اینکه مشکل انفجار فضای حالت را تا حدی بهبود بخشیده‌اند، اما مشکل همچنان

باقی است و فرآیند وارسی مدل در سیستم‌ها همچنان از مشکل مواجهه با کمبود حافظه و سرعت پایین رنج می‌برد.

روش‌های اکتشافی[۷]، دسته دیگری از روش‌هایی هستند که جهت مقابله با مشکل انفجار فضای حالت در وارسی مدل، به کارگرفته شده‌اند. در وارسی مدل نمادین در [۲۴] و در وارسی مدل صریح در[۲۵]، از روش‌های اکتشافی استفاده شده است که در جستجوی آن‌ها، ابتدا حالت‌هایی ملاقات می‌شوند که بیش‌ترین احتمال را برای رسیدن به خطا دارند. یک جستجوی اول بهترین، در وارسی مدل بر پایه نمودار تصمیم‌گیری دودویی، توسط ابزار Murφ در[۲۶] مورداستفاده قرارگرفته است. در [۲۷] نویسندگان،یک چارچوب با استفاده ازجستجوی اکتشافی به‌منظور آنالیز خصوصیات ساختاری سیستم‌های مدل‌شده با سیستم تبدیل گراف ترتیب داده‌اند. آن‌ها معتقدندکه جستجوی اکتشافی به کاهش آنالیزها و نیز رسیدن به راه حل‌های کوتاه‌تر – مسیرهای کوتاه‌تر در سیستم تبدیل گراف کمک می‌کند. در این پیاده‌سازی، از الگوریتم A*استفاده شده است. این چارچوب در HSF-SPIN که یک وارسی‌کننده مدل تخمینی سازگار با وارسی‌کننده مدل SPIN است، پیاده‌سازی شده است [۲۸].

روش‌های اکتشافی که تاکنون به آن‌ها اشاره شد، همگی روش های جستجوی کامل[۸] هستند و در تمامی آن‌ها در بدترین حالت، باید کل فضای حالت کاوش شود. بنابراین اگرچه این روش‌ها، به یافتن خطا به نحو سریع‌تر کمک می‌کنند، اما همچنان از مشکل انفجار فضای حالت رنج می‌برند. در روش پیشنهادشده در این پژوهش، راه‌حلی ارائه شده است که در آن به‌جای کل فضای حالت، هر بار تنها بخش‌هایی از آن برای جستجو در نظر گرفته می‌شوند.

ازجمله دیگر روش‌هایی که در چارچوب هوش مصنوعی در این راستا به کار گرفته شده است، می‌توان از الگوریتم کلونی مورچه‌ها[۹] [۱۰، ۱۱ و ۱۵] چارچوبی مبتنی بر یادگیری تقویتی [۱۲] و الگوریتم ژنتیک [۱۵-۱۳ و ۲۹] نام برد.

راهکارهای مبتنی بر الگوریتم کلونی مورچه‌ها، در هنگام پیمایش مسیرها برای یافتن حالت خطا، با الهام گرفتن از رفتار مورچه‌ها، مسیرهای کوتاه‌تر را به مسیرهای طولانی‌تر ترجیح می‌دهند؛ که این نکته به دلیل نیاز به حافظه کم‌تر برای ذخیره حالت‌ها، باعث جلوگیری از انفجار فضای حالت می‌شود. لذا در پژوهش‌های صورت‌گرفته برمبنای این الگوریتم، برنامه‌های وارسی مدل توانسته‌اند بهینه‌ترین یا نزدیک به بهینه‌ترین پاسخ را برای مسائل بیابند. [۱۰، ۱۱ و ۱۵] با این‌حال تا کنون پژوهشی در این زمینه در حوزه سیستم‌های تبدیل گراف صورت نگرفته است.

هدف از ارائه یک چارچوب مبتنی بر یادگیری تقویتی، بهینه‌سازی مصرف حافظه به‌کمک افزایش احتمال انتخاب مسیرهایی است که در آن‌ها خصوصیات مورد نظر، نقض می‌شوند. این چارچوب، تنها برای وارسی خصوصیات ارائه‌شده توسط منطق زمانی خطی[۱۰] ارائه شده است [۱۲].

به کمک الگوریتم ژنتیک[۱۱] نیز کارهایی در مقابله با مشکل انفجار فضای حالت سیستم‌های وارسی مدل، انجام شده است [۱۳، ۱۴ و ۲۹]. در [۱۳،۱۴] تمرکز بر روی مقابله با مشکل انفجار فضای حالت در فرآیند وارسی مدل در مسائل مدل‌شده برمبنای مدل‌سازی مبتنی بر متن است. در [۱۳] یک چارچوب برای کاهش فضای حالت در سیستم‌های واکنشی[۱۲] ارائه شده است. در این روش با بهره‌گیری از الگوریتم ژنتیک، به‌جای جستجوی همه فضای حالت، جستجو را به سمت حالت‌های خطا؛ به‌طور مثال بن‌بست، هدایت می‌کند. الگوریتم ارائه‌شده، در verisoft که ابزاری جهت کاوش فضای حالت مبتنی بر مدل سازی متنی است، پیاده سازی شده است. در [۱۴] نیز یک سیستم وارسی مدل مبتنی بر مدل‌سازی متنی، جهت کشف بن‌بست، بر روی JPF (Java Pathfinder)، که یک سیستم درستی‌یابی برنامه‌های اجرایی جاوا است، طراحی شده است. همان‌طور که اشاره شد این پژوهش‌ها بر روی مدل‌های متنی و نه سیستم‌های تبدیل گراف تمرکز کرده‌اند.

تمامی راهکارهای ارائه شده، با هدف مقابله با مشکل انفجار فضای حالت در سیستم‌های وارسی مدل انجام شده‌اند اما هیچیک از آن‌ها بر روی سیستم‌های تبدیل گراف متمرکز نیستند. این در حالی است که سیستم تبدیل گراف، یک زبان رسمی گرافیکی است که در فازهای مختلف توسعه نرم‌افزار مانند فرامدل‌سازی[۱۳][۳۰]، ارائه سبک معماری[۱۴][۳۱]، پالایش[۱۵][۳۲]، بازسازی[۱۶][۳۳]، تبدیل مدل[۳۴]، تحلیل کارایی[۱۷][۳۵] و ... مورد استفاده قرار می‌گیرد.

در [۳۶]، یک تکنیک از نوع وارسی مدل محدودشده[۱۸](BMC)، برای سیستم‌های تبدیل گراف، ارائه شده است. در این تکنیک، نمونه گراف BMC، به یک نمونه SMT[۱۹] تبدیل می‌شود که گراف اولیه، قوانین تبدیل و خصوصیات مورد وارسی، توسط منطق مرتبه اول توصیف می‌شوند. در این تکنیک، خطایی که در پی کشف آن هستیم، به صورت یک گراف ممنوعه[۲۰] در نظرگرفته شده و بررسی می شود که آیا آن الگوی ممنوعه، دسترس‌پذیر است یا نه. محققان این پژوهش، ابزار مورد نظر خود را به‌عنوان افزونه‌ای بر روی ابزار Groove پیاده‌سازی کرده‌اند. اگرچه ماهیت این پژوهش، به کلی با پژوهش حاضر متفاوت است و با استفاده از حل‌کننده‌های SAT به بررسی و آنالیز سیستم می‌پردازد و بر پایه جبر رابطه‌ای کار می‌کند و همچنین محققان آن، آزمایش‌های خود را تنها بر روی یک مطالعه موردی تست کرده‌اند و به جزئیات پیاده‌سازی نظیر کامپیوتری که آزمایش‌ها را بر روی آن انجام داده‌اند، اشاره‌ای نکرده‌اند، با این حال، نتایج این پژوهش را با نتایج پژوهش حاضر در بخش ۶، مقایسه کرده‌ایم.

در [۱] نیز با بهره‌گیری از الگوریتم ژنتیک، نیل به هدف موردنظر، با تأکید بر یافتن بن‌بست، بر روی سیستم‌های تبدیل گراف متمرکز شده است. ایده کلی در این روش این است که به‌جای تولید کل فضای حالت در فرآیند وارسی مدل، با استفاده از روشی مبتنی بر الگوریتم

ژنتیـــک، هر بار تنها بخشی از فضای حالت تولید شود. این پژوهش موفق شده است مصرف حافظه را بهبود بخشد.

۳- رویکرد پیشنهادی

پس از مطالعه کارهای مشـــابه می‌توان گفت باوجود کاربردهای فراوان سیستم تبدیل گراف [۳۵-۳۰]، به‌جز [۱] و [۴۳] در پژوهش دیگری بر روی مقابله با مشکل انفجار فضای حالت در وارسی مدل با تمرکز بر سیستم‌های تبدیل گراف پژوهشی صورت نگرفته است. مقایسه نتایج راهکار پیشـــنهادی با نتایج این دو پژوهش در بخش ۶ موجود اســـت. همچنین ازآنجاکه هدف از این مقاله، مقابله با مشـــکل انفجار فضـــای حالت جهت وارســـی مدل اســـت، روش‌هایی که بر مبنای حذف راه‌حل‌های احتمالی کار می‌کنند، نمی‌توانند روش‌های مؤثری باشـــند. لـــذا یافتن راهکاری جهت هدایت الگوریتم به سمت حالت‌های بهینه به‌جای حذف آن‌ها، منطقی‌تر به نظر می‌رســـد. همان‌طور که پیش‌تر ذکر شد، از این ایده در [۱] توسط الگوریتم ژنتیک استفاده شده است. باوجود این‌که الگوریتم ژنتیک در حل مسائل بهینه‌سازی، مخصوصاً در فضاهای بزرگ، کارکرد مناسبی دارد، بااین‌حال به‌کارگیری آن در هدف موردبحث این پژوهش، از مشـــکل هم‌گرایی دیرهنگام و در مواردی از مشکل نتایج ضعیف رنج می‌برد [۱]. لذا بهتر است الگوریتم دیگری را برای نیل به این هدف بکار بســـت. الگوریتم پرندگان، با به‌کارگیری مفاهیم ســـاده و پارامترهای اندک، یکی از الگوریتم‌های مؤثر در زمره هوش مصنوعی و محاسباتی به‌حساب می‌آید. می‌توان با بهره‌گیری از این الگوریتم و با ایده گرفتن از فرآیند وارســـی مدل، راهکار بهتری را جهت نیل به هدف این پژوهش ارائه داد.

لازم به ذکر اســـت، الگوریتم پرندگان نیز دچار ضـــعف اســـت و می‌تواند جســـتجو را در دام بهینه‌های محلی گرفتار کند. بنابراین پیشـــنهاد این پژوهش اســـتفاده از الگوریتم پرندگان در کنار الگوریتم دیگری است که ضعف الگوریتم پرندگان را بهبود بخشد. ازآنجاکه ثابت شده است الگوریتم جستجوی گرانشی، در اجتناب از به دام افتادن در بهینه‌های محلی، بهتر از الگوریتم پرندگان عمل می‌کند [۵۲-۵۱]، در این پژوهش از ترکیبی از الگوریتم پرندگان و الگوریتم جســـتجوی گرانشـی اســتفاده شـده و نتایج آن با پژوهش‌های قبلی مقایسـه شـده اسـت. برای مدل‌سـازی و عملیات مربوط به کشـف خطا (بن‌بسـت) در مدل، از ابزار Groove [۳] که یکی از ابزارهای پرکاربرد سیستم تبدیل گراف است، استفاده شده است.

۴- مفاهیم اولیه

۴-۱- وارسی مدل

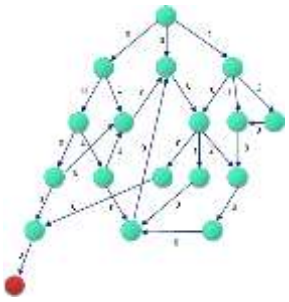
وارسی مدل، فرآیندی است که به‌طور سیستماتیک و خودکار بررسی می‌کند که آیا مدل توصیف‌شده از سیستم، خصوصیات مورد انتظار را برآورده می‌کند یا خیر. در فرآیند وارسی مدل، پس از توصیف سیستم توسط یک زبان رسمی و توصیف خصوصیات مهم آن، فضای حالت، در قالب یک گراف شـــامل حالت‌های مختلف ســـیســـتم به‌عنوان رئوس و انتقال‌های بین آن‌ها به‌عنوان یال‌ها، ایجاد شـــده و وارســـی‌کننده مدل، تمامی حالت‌ها و ســـناریوهای ممکن ســـیســـتم را کاوش می‌کند. به‌این‌ترتیب می‌توان نشان داد که مدل ارائه‌شده از سیستم، خصوصیات موردنظر را به‌درستی برآورده می‌کند یا خیر.

فرآیند وارسی مدل، قبـــل از پیاده‌ســـازی و در سطح مدل انجام می‌شود و شامل دو قاعده اصـــلی است:

۱) توصیف سیستم توسط یک روش رسمی.

۲) توصیف خصوصیات مهم سیستم توسط روش رسمی.

وارســـی‌کننده مدل از طریق بررســـی تمام حالت‌ها و انتقال‌های مدل‌شده توسط روش رسمی، مشخص می‌کند که سیستم، خصوصیت موردنظر را پشـــتیبانی می‌کند یا منجر به کشـــف خطا به‌طور مثال؛ بن‌بست می‌شود [۲]. در بحث وارسی مدل، بن‌بست، با توجه به توصیف سیستم، به وضعیتی اطلاق می شود که فضای حالت سیستم، دارای حالتی باشـــد که هیچ انتقال خروجی ندارد. به‌عبارت‌دیگر، پس‌ازآن حالت که نماینده یک وضعیت از سیستم است، هیچ سناریوی دیگری برای سیستم، وجود ندارد. به‌طور مثال در درخت فضای حالت شکل ۱، حالت قرمزرنگ، دچار بن‌بست است چراکه هیچ گذر خروجی برای آن وجود ندارد.



شکل ۱: به وضعیت نود قرمزرنگ کـه دارای هیچ گـذر خـروجی نیست، بن-بست گفته می‌شود

۴-۲- سیستم تبدیل گراف

تبدیـل گـراف، یک زبـان رسمی تصویری جهت مدل‌سازی سیستم‌های نرم‌افزاری اسـت که برای توصـیف خصـوصـیات رفتاری و سـاختاری سـیسـتم‌ها از یال‌ها و رئوس اسـتفاده می‌کند و به دلیل بهره‌گیری از ویـژگی گرافیکی بـودن، بـرای تـوصیف گویای سیستم‌های نرم‌افزاری بسیار مناسب است [۳۹]. یک سیستم تبدیل گراف، به صورت یک ۳تایی تعریف می‌شـود: GTS=(TG,HG,R) که TG، گراف نوع[۲۱]، HG، گراف میزبان[۲۲]، و R، مجموعه قوانین است.

گراف نوع، کلیت سـیسـتم را مشـخص می‌کند و به عبارت بهتر فرامدل سیستم است.

گراف میزبان، گرافی است که نشان‌دهنده وضعیت اولیه سیستم اسـت که باید نمونه‌ای از گراف نوع باشـد. به‌عبارت‌دیگر، گراف میزبان باید همریخت گراف نوع باشد [۴۰].

یک قانون تبدیل گراف به صورت p: L→R و شامل دو گراف L و R است که همریخت گراف نوع هستند. مشخصات یال‌ها، مطابق با مشخصات یال‌ها در گراف نوع و رئوس ابتدا و انتهای آن‌ها نیز مطابق با گراف نوع، دارای همان انواع هستند. L، بیانگر پیش‌شرط‌های قوانین است و R، پس شرط‌ها را توصیف می‌کند. وظیفه قوانین در سیستم‌های تبدیل گراف، تعمیم و تبدیل است به‌این‌صورت که در گراف داده‌شده، با وقوع زیرگراف L، آن را با R جایگزین می‌کند [۴۰].

۴-۳- الگوریتم پرندگان

این الگوریتم از رفتار دسته‌جمعی پرندگان به هنگام جستجوی غذا الهام گرفته شده است. گروهی از پرندگان به دنبال غذا می‌گردند درحالی‌که تنها یک‌تکه غذا وجود دارد. ایده اصلی الگوریتم پرندگان دنبال کردن پرنده‌ای است که کم‌ترین فاصله را تا غذا دارد. در این الگوریتم، هر راه حل، معادل یک پرنده است. هر ذره یک مقدار شایستگی دارد که هرچه به غذا، که در هر مسئله، هدف است، نزدیک‌تر باشد، ارزش شایستگی بیش‌تر خواهد بود. هر ذره دارای یک موقعیت در فضای جستجو و یک سرعت است که موجب هدایت حرکت و تغییر موقعیت آن ذره می‌شود.

روند الگوریتم به این صورت است که ابتدا، گروهی از ذرات که همان راه‌حل‌های احتمالی هستند، به‌طور تصادفی ایجاد می شوند. در هر تکرار، برای هر ذره میزان شایستگی محاسبه می‌شود. سپس درصورتی‌که شرط خاتمه برآورده نشد، *pbest* و *gbest* به‌روز می‌شوند. *pbest* در هر مرحله، بهترین موقعیتی است که ذره تاکنون در آن بوده است و *gbest*، بهترین موقعیت در بین تمام ذرات است که تاکنون تجربه کرده‌اند. پس از یافتن این دو بهترین مقدار، سرعت و موقعیت هر ذره با استفاده از روابط (۱) و (۲) به‌روز شده و الگوریتم وارد تکرار بعدی خواهد شد. الگوریتم تا جایی ادامه می‌یابد که یا شرط خاتمه برآورده شود و یا تعداد معینی تکرار صورت گیرد.

$$v[t+1] = w \times v[t] + C1 \times rand(t) \times (pbest[t] - posision[t]) + C2 \times rand(t) \times (gbest[t] - position[t]) \quad (۱)$$

$$position[t+1] = position[t] + v[t+1] \quad (۲)$$

w، وزن اینرسی است که جهت کنترل تأثیر سرعت ذره در مرحله قبل بر سرعت فعلی آن استفاده می‌شود. این پارامتر باعث برقراری تعادل بین جستجوی عمومی و محلی خواهد شد. *C1*، در کنار عدد تصادفی *rand()*، تعیین‌کننده این است که ذره تا چه میزان به تجربه شخصی خود وابسته است. *C2*، در کنار عدد تصادفی *rand()*، تعیین‌کننده این است که ذره تا چه میزان به تجربه جمعی وابسته است. *rand* ، یک عدد تصادفی بین ۰ و ۱ است و $0 \le C1 + C2 \le 4$. اکثر اوقات C1= C2 = 2 در نظر گرفته می‌شود [۴۱].

سرعت ذرات در هر بعد، به یک مقدار V_{max} محدود می‌شود. اگر به‌روزرسانی‌ها باعث شوند که سرعت در یک بعد از V_{max} بیش‌تر شود، مقدار سرعت در آن بعد برابر با V_{max} قرار می‌گیرد.

همان‌طور که اشاره شد، در الگوریتم پرندگان، ذرات به‌مرور به سمت بهترین راه‌حل پیدا شده، متمایل می شوند. اگر این راه‌حل، یک بهینه محلی باشد، ذرات به سمت آن می‌روند و در اطراف آن جمع می‌شوند و گاهی این روند باعث می‌شود که تکرارهای الگوریتم به پایان برسد و هنوز ذرات از دام این بهینه محلی رها نشده باشند و لذا راه‌حل اصلی، در تکرارهای الگوریتم یافت نمی‌شود. این مشکل که "به دام افتادن در بهینه های محلی" نام دارد، بزرگ‌ترین مشکل الگوریتم پرندگان است.

۴-۴- الگوریتم جستجوی گرانشی

در این الگوریتم، فضای جستجو شامل مجموعه‌ای از اجرام است به‌گونه‌ای که هر جرم شامل خصوصیت مکان، جرم گرانشی و جرم اینرسی و میزان شایستگی است. مکان یا موقعیت هر جرم، یک راه‌حل از مسئله است که هدف، بهینه‌کردن آن است. جرم‌گرانشی و جرم اینرسی هرکدام توسط روابط (۳) و (۴) قابل محاسبه هستند [۴۲].

$$m_i(t) = \frac{fitness_i(t) - worst(t)}{best(t) - worst(t)} \quad (۳)$$

fitness میزان شایستگی ذره، *best* بهترین جرم ازنظر میزان شایستگی و *worst* بدترین جرم ازنظر میزان شایستگی در هر تکرار از اجرای الگوریتم است.

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \quad (۴)$$

اگر در سیستمی n جرم وجود داشته باشد، موقعیت جرم i به‌صورت رابطه (۵) بیان می‌شود که x_i^d، بعد d از موقعیت جرم i است.

$$X_{i=}(x_i^1, \dots, x_i^d, \dots, x_i^n) \quad for\ i=1,\dots,n \quad (۵)$$

در این سیستم در زمان t از سوی جرم j به جرم i در راستای بعد d نیرویی برابر $F_{ij}^d(t)$ وارد می‌شود که طبق رابطه (۶) محاسبه می شود و در آن G برابر ضریب گرانش در زمان t، $M_i(t)$ و $M_j(t)$ جرم ذرات i و j و $R_{ij}(t)$ فاصله بین این دو جرم است.

$$F_{ij}^d(t) = G(t)\frac{M_i(t) \times M_j(t)}{R_{ij}(t)}(x_j^d(t) - x_i^d(t)) \quad (۶)$$

طبق قانون گرانش، نیروی وارد بر جرم i در زمان t، از مجموع نیروهای وارد شده از طرف سایر اجرام موجود در سیستم بر این جسم ارزیابی می شود. الگوریتم جستجوی گرانشی برای رسیدن به بهینگی بهتر در سرعت و دقت، تنها k جرم بهتر را در نظر می‌گیرد. لذا نیروی وارد بر جسم i با رابطه (۷) قابل محاسبه است.

$$F_i^d(t) = \sum_{j \in K_{best}, j \neq i} F_{ij}^d(t) \quad (۷)$$

در رابطه (۷)، K-best مجموعه K جرم بهتر اســت. الگوریتم، در ابتدا نیاز به جســتجوی ســراســری دارد. در این حالت مقدار K برابر میزان جمعیت است و با افزایش تکرارها به‌صورت خطی کاهش می‌یابد.

طبق قانون دوم نیوتن، نیروی وارد بر جرم i در راســتای بعد d، شتابی مطابق با رابطه (۸) را پدید می‌آورد.

$$a_i^d = \frac{F_d^i}{M(t)} \quad (۸)$$

سپس این شتاب منجر به تغییر سرعت جرم می شود. مقدار این سرعت طبق رابطه (۹) محاسبه می‌شود.

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t) \quad (۹)$$

که در این رابطه، *rand* یک عدد تصادفی بین ۰ و ۱ است.

ســپس موقعیت جرم نیز مطابق رابطه (۱۰) تغییر خواهد کرد و جرم حرکت می‌کند.

$$\boldsymbol{x_i^d(t+1) = x_i^d(t) + v_i^d(t+1)} \quad \textbf{(۱۰)}$$

در هر مرحله از اجرای الگوریتم و با گذشــت زمان، مقدار ضــریب گرانش G طبق رابطه (۱۱) کاهش خواهد یافت.

$$G(t) = G_0 e^{-a\frac{t}{T}} \quad (۱۱)$$

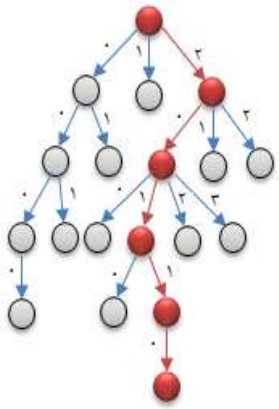
در این رابطه، G_0 ثابت و برابر ۱۰۰، a ثابت و برابر ۲۰ و T دفعات تکرار الگوریتم اســت. الگوریتم تا جایی ادامه می‌یابد که یا با به دســت آمدن شایستگی مورد نظر، شرط خاتمه برآورده شود و یا با تعداد معین تکرار به پایان برسد.

۵- ترکیب وارسی مدل و الگوریتم‌های پیشنهادی

۵-۱- موقعیت هر پرنده و تابع شایستگی

فضــای حالت، مجموعه‌ای از حالت‌ها و انتقال‌های بین آن‌ها اســت که به‌صــورت یک درخت به‌عنوان درخت فضــای حالت، ترســیم می‌شــود. هدف، یافتن یک حالت خطا، به‌طور مثال؛ بن‌بســت، در فضــای حالت گسترده مسئله است. برای رسیدن به بن‌بست، مجموعه‌ای از حالت‌ها و انتقال‌ها باید طی شــود. در راهکار پیشــنهادی، برای یافتن یک مســیر بهینه به یک حالت بن‌بســت، از الگوریتم پرندگان و ســپس ترکیبی از این الگوریتم به همراه الگوریتم جستجوی گرانشی استفاده شده است.

با توجه به این موضوع، در راهکار پیشنهادی، فرض شده است که هر پرنده یا ذره که همان راه‌حل احتمالی است، نماینده چنین مسیری اســت. لذا، موقعیت هر پرنده، به‌صــورت دنباله‌ای از اعداد نمایش داده می شود که هر یک از این اعداد، یک عدد تصادفی تولید شده بین ۰ تا حداکثر n - تعداد انتقال‌های خروجی در هر مســئله اســت. مقدار n در هر مســئله متفاوت اســت و برای تخمین دقیق‌تر آن می‌توان از ابزار Simulator نرم‌افزار Groove، قســمت LTS، بخش State Space، بهره جست. در شکل ۲، برای روشن‌شدن نحوه انتخاب ذرات، یک مسیر، به رنگ قرمز در فضای حالت فرضی نشان داده شده است. این مسیر نماینده یک پرنده در الگوریتم پرندگان اســت. در این حالت، موقعیت پرنده عبارت است از : ۰, ۱, ۱, ۰, ۲. علاوه بر موقعیت ذرات، سرعت و شتاب ذرات نیز آرایه‌هایی از اعداد متناسب با ابعاد مسئله هستند.



شکل ۲: یک راه‌حل احتمالی که معادل یک ذره است

در روش‌های هوش مصنوعی که در الگوریتم پیشنهادی استفــاده خــواهــد شــد، بــرای تعیین میزان شایستگی هر یک از راه‌حل‌های احتمالی، باید از یک تابع شایستگی متناسب باهدف مسئله اســتــفاده شــود. با توجه به اینکه این پژوهش متمرکز بر یافتن حالت‌های شــامل بن‌بســت اســت، تابع شــایســتگی طبق رابطه (۱۲)، مجموع تعداد کل انتقال‌های خروجی که در طول یک مسیر طی شده وجود دارد، در نظر گرفته شده است؛ چرا که یک استراتژی هوشمند ساده می‌تواند این باشــد که هر چه تعداد خروجی‌های حالت‌های مختلف در یک مســیر کمتر باشد، احتمال رسیدن به بن‌بست افزایش می‌یابد. جهت محاسبه شایستگی و کشف بن‌بست از استراتژی تعریف شده در [۱] استفاده شده است. این استراتژی برای هر ذره توسط ابزار Generator نرم‌افزار Groove فراخوانی شــده و میزان شــایســتگی آن محاســبه می‌گردد. بن‌بســت در صــورت وجود، کشــف و نتیجه برای ادامه کار به الگوریتم بازگردانده می‌شود.

$$F(x) = \sum_{i=1}^{EL} L_i[۱] \quad (۱۲)$$

۵-۲- راهکار مبتنی بر الگوریتم پرندگان

در این پژوهش، جهت کشــف بن‌بســت در مدل‌های مبتنی بر تبدیل گراف، ابتدا راهکاری بر پایه الگوریتم پرندگان ارائه می‌شود. روال کار در راهکار پیشنهادی به این صورت است که پس از مشخص شدن گرامر، گراف میزبان و عمق مورد نظر جهت جستجو، ابتدا تعداد p ذره (پرنده) به‌طور تصــادفی به‌عنوان جمعیت اولیه تولید شــده، مقداردهی اولیه می‌شــوند. هر ذره شــامل دو ویژگی ســرعت و موقعیت اســت. لذا در مقداردهی اولیه، ابتدا ســرعت هر ذره صــفر و موقعیت آن نیز مطابق با روالی که در بخش قبل به آن اشاره شد، مقداردهی می شود. سپس، میزان شایستگی هر یک از ذرات، با استفاده از رابطه ۱۲ محاسبه شده و چنانچه خطا کشــف شــد، اجرا پایان می‌یابد. در غیر این صــورت، بهترین تجربه شخصی هر ذره (*pbest*) و بهترین تجربه جمعی مجموعه

ذرات (*gbest*) با توجه به مقادیر شایستگی ذرات و همچنین سرعت و موقعیت هر ذره با توجه به روابط ۱ و ۲ به‌روز می‌گردند. بدین معنی که، مسیرهای موجود، به سمت مسیرهای بهتر تغییر جهت می‌دهند و این روال تکرار می شود تا این‌که بن‌بست کشف شود یا تکرارها پایان پذیرد. شبه‌کد این راهکار به شرح زیر است:

۱. تعداد p ذره به‌طور تصادفی به‌عنوان جمعیت اولیه تولید می‌شود.
۲. هریک از ذرات جمعیت اولیه، مقداردهی اولیه می‌شوند.
۳. میزان شایستگی هر یک از ذرات محاسبه شده و چنانچه خطا کشف شد برو به ۷.
۴. *pbest* و *gbest*، با توجه به مقادیر شایستگی ذرات، به‌روز می‌شوند.
۵. سرعت و موقعیت هر ذره به‌روز می‌شوند.
۶. برو به مرحله ۳.
۷. پایان.

نحوه عملکرد محاسبه تابع شایستگی و کشف بن‌بست در مرحله ۳ از شبه‌کد راهکار پیشنهادی، از استراتژی مطرح‌شده در [۱]، استفاده شده است. استراتژی به‌کارگرفته شده، به این صورت عمل می‌کند که هر بار تنها یک ذره یا به عبارتی یک مسیر خاص را به نرم‌افزار Groove می‌فرستد. به عبارتی، Groove به‌جای اینکه در فرآیند وارسی مدل، کل فضای حالت مدل را تولید و عملیات وارسی را جهت کشف بن‌بست انجام دهد، تنها بخشی از فضای حالت را در هر سطح، طبق ذره ارسالی، تولید می‌کند. سپس فضای حالت تولید شده با استفاده از ذره یا مسیر ارسالی را برای وجود بن‌بست یا عدم وجود آن وارسی می‌کند. همچنین استراتژی مطرح‌شده، میزان تابع شایستگی آن مسیر یا ذره را نیز محاسبه کرده و این دو را به‌عنوان پارامترهای خروجی به الگوریتم برمی‌گرداند.

۵-۳- راهکار مبتنی بر ترکیب الگوریتم پرندگان و الگوریتم جستجوی گرانشی

در این بخش، راهکار پیشنهادی حاصل از ترکیب الگوریتم پرندگان و الگوریتم جستجوی گرانشی موردبررسی قرار گرفته است.

روال عملکرد این الگوریتم به این صورت است که ابتدا مطابق با راهکار ارائه‌شده در بخش ۵-۲ و پس از مشخص شدن گرامر مسئله مورد نظر، گراف میزبان و عمق مورد نظر جهت کاوش، تعداد p ذره به‌طور تصادفی به‌عنوان جمعیت اولیه تولید می‌شوند. سرعت هر ذره صفر و موقعیت آن با تولید اعداد تصادفی بین ۰ تا حداکثر تعداد انتقال‌های خروجی از یک حالت، مقداردهی می‌شوند. سپس، میزان شایستگی هر ذره، با استفاده از رابطه ۱۲ محاسبه شده، چنانچه خطا کشف شد، اجرا پایان می‌یابد و در غیراین‌صورت، *pbest*، *gbest* و سپس، سرعت و موقعیت هر ذره با توجه به روابط ۱ و ۲ به‌روز می‌شوند.

تا این مرحله، روند اجرای الگوریتم مطابق با الگوریتم ارائه شده در بخش ۵-۲ اجرا شده و پس‌ازآن، ذرات به الگوریتم جستجوی گرانشی سپرده می شوند. به‌عبارت‌دیگر ذراتی که تاکنون پرنده بوده‌اند از اینجا به بعد تبدیل به جرم شده و درگیر قانون گرانش می‌شوند. به این صورت که، ابتدا جرم هر ذره با استفاده از روابط ۳ و ۴ محاسبه می‌گردد. سپس به تعداد k، ذراتی که جرم بیش‌تری دارند، انتخاب می‌شوند. مابقی جمعیت که متعلق به این k ذره نیستند، تحت عنوان جمعیت تحت تأثیر انتخاب شده و پس‌ازآن تأثیر نیروی k ذره بهتر، بر روی جمعیت تحت تأثیر با استفاده از رابطه ۷ محاسبه می شود. نیروی وارد شده از طرف k ذره بهتر بر ذرات تحت تأثیر، باعث ایجاد شتاب در ذرات متعلق به جمعیت تحت تأثیر می شود که میزان آن تو سط رابطه ۸ محاسبه می‌گردد. به دلیل تأثیر شتاب به‌وجودآمده، سرعت ذرات متعلق به جمعیت تحت تأثیر، طبق رابطه ۹ تغییر می‌کند. این سرعت باعث حرکت این ذرات در فضای جستجو شده و موقعیت آن‌ها را طبق رابطه ۱۰ تغییر می‌دهد. در هر تکرار از الگوریتم، مقدار G، مطابق با رابطه ۱۱ به‌روز می‌شود. روال ذکرشده تا رسیدن به اولین حالت بن‌بست و یا پایان یافتن تعداد تکرارها، ادامه می‌یابد. روند اجرای راهکار ترکیبی مطرح‌شده به‌صورت زیر است:

۱. تعداد p ذره به‌طور تصادفی به‌عنوان جمعیت اولیه تولید می‌شود.
۲. هریک از ذرات جمعیت اولیه، مقداردهی اولیه می‌شوند.
۳. میزان شایستگی هر یک از ذرات محاسبه شده و چنانچه خطا کشف شد برو به ۱۵.
۴. *pbest* و *gbest*، با توجه به مقادیر شایستگی ذرات، به‌روز می‌شوند.
۵. سرعت و موقعیت هر ذره مطابق روابط ۱ و ۲ به‌روز می‌شوند.
۶. جرم هر پرنده با استفاده از روابط ۳ و ۴ محاسبه می‌شود.
۷. k جرم بهتر انتخاب می‌شوند. هرچه جرم بیش‌تر باشد بهتر است.
۸. مابقی جمعیت که متعلق به k جواب بهتر نیستند، به‌عنوان جمعیت تحت تأثیر انتخاب می‌شوند.
۹. تأثیر نیروی k جرم بهتر بر روی جمعیت تحت تأثیر با استفاده از رابطه ۷ محاسبه می‌شود.
۱۰. شتاب جمعیت تحت تأثیر، توسط رابطه ۸ محاسبه می‌شود.
۱۱. سرعت جمعیت تحت تأثیر، با استفاده از رابطه ۹ به‌روز می‌شود.
۱۲. موقعیت جمعیت تحت تأثیر، توسط رابطه ۱۰ به‌روز می‌شود.
۱۳. مقدار G با استفاده از رابطه ۱۱ به‌روز می‌شود.
۱۴. بازگشت به مرحله ۳
۱۵. پایان

همان‌طور که ذکر شد، جهت بررسی هر ذره و محاسبه شایستگی آن از استراتژی ارائه‌شده در [۱]، استفاده شده است.

۵-۴- انتخاب پارامترها

ازآنجاکه مقدار پارامترهای الگوریتم پرندگان، تأثیر بسزایی در سرعت رسیدن به پاسخ در راهکار پیشنهادی دارد، لذا پس از مطالعه در زمینه

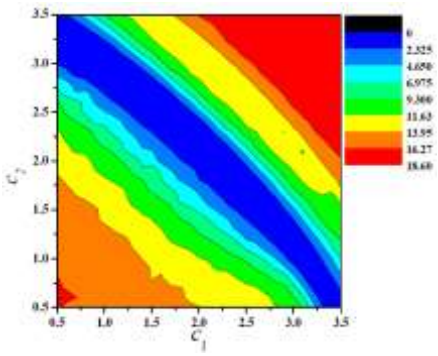
انتخاب بهینه این پارامترها [۵۰-۴۴]، آزمایش‌هایی بدین منظور جهت یافتن بهترین مقادیر برای پارامترها در این راهکار انجام شده است.

C1، C2 و w، پارامترهای محاسبه سرعت (رابطه ۱) در الگوریتم پرندگان هستند که انتخاب مقادیر آن‌ها تأثیر زیادی در سرعت و کارایی این الگوریتم و بنابراین راهکار پیشنهادی دارد.

۵-۴-۱- انتخاب پارامترهای C1 و C2

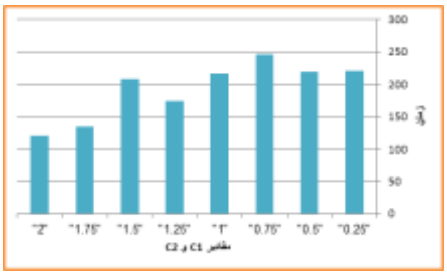
در مقاله اصلی الگوریتم پرندگان که اولین بار در سال ۱۹۹۵ ارائه شد [۴۱]، این دو پارامتر در محاسبه فرمول سرعت، وجود ندارند و به‌جای آن‌ها اعداد ثابت ۲ قرار داده شده است. این دو پارامتر در نسخه تغییریافته الگوریتم پرندگان که در سال ۱۹۹۸ ارائه شد [۴۷]، وارد محاسبات الگوریتم پرندگان شدند و با محاسباتی، شرط C1+C2<=4 را برای انتخاب آنان لحاظ کرده‌اند و اعلام کرده‌اند که بسته به اینکه وابستگی به تجربیات شخصی ذرات برایمان در مسئله مهم‌تر است یا تجربیات جمعی، می‌توان مقادیر C1 و C2 را انتخاب کرد. بااین‌وجود در کاربردهای مختلف، C1 و C2، معمولاً مساوی با یکدیگر و برابر با ۲ در نظر گرفته می‌شوند [۴۹، ۵۰].

در [۴۴] آنالیزی در این زمینه انجام شده که نتیجه این تحلیل را در نموداری مطابق شکل ۳ منتشر کرده است. در این نمودار، مقادیر یک تابع ارزیاب، متناسب با تغییر C1 و C2، به تصویر کشیده شده است و محدوده‌های مختلف مقادیر این تابع ارزیاب با رنگ‌های مختلف از یکدیگر جدا شده‌اند. همان‌طور که مشخص است، بهترین مقادیر، در محدوده‌ای به رنگ آبی پررنگ به تصویر کشیده شده است.



شکل ۳: نمودار حاصل از بررسی مقادیر C1 و C2 [۴۴]

با توجه به اینکه در الگوریتم پیشنهادی، هم تجربیات شخصی ذرات و هم تجربیات جمعی کل جمعیت حائز اهمیت است، لذا تصمیم محققان بر این است که C1 و C2 را برابر با یکدیگر در نظر بگیرد که با توجه به موارد ذکر شده و نیز با توجه به شکل ۳ می‌توان گفت بهترین مقدار برای این انتخاب C1=C2=2 است. بااین‌حال، برای اطمینان از درستی انتخاب مقادیر مناسب در راهکار پیشنهادی، محدوده معتبری از مقادیر C1 و C2 با فرض C1=C2 در راهکار پیشنهادی آزمایش شده است تا از انتخاب مقادیر درست اطمینان حاصل گردد که نتیجه آن به‌صورت نمودار شکل ۴ قابل مشاهده است. همان‌طور که از نمودار برمی‌آید، بهترین مقادیر همان C1=C2=2 است.



شکل ۴: نتیجه آزمایش جهت انتخاب مقادیر مناسب C1 و C2

۵-۴-۲- انتخاب پارامتر w

این پارامتر نیز در مقاله اصلی الگوریتم پرندگان [۴۱]، اصلاً وجود ندارد. بلکه در مقاله دیگری که بعدتر توسط پدیدآورندگان الگوریتم پرندگان ارائه شد [۴۷]، وارد محاسبات این الگوریتم شد. مقدار پیشنهاد شده برای این پارامتر، عددی در بازه [۰/۹-۰/۴] است. برای یافتن مقدار مناسب این پارامتر در راهکار پیشنهادی، مقادیر مختلف آزمایش شده است تا از انتخاب مقدار مناسب اطمینان حاصل شود. نتیجه این آزمایش در شکل ۵ در قالب نموداری به تصویر کشیده شده است. همان‌طور که مشخص است، بهترین کارایی راهکار پیشنهادی، با انتخاب مقدار ۰/۸ برای این پارامتر به‌دست آمده است.

شکل ۵: نتیجه آزمایش‌ها برای انتخاب مقدار مناسب پارامتر w

۶- پیاده‌سازی و ارزیابی

جهت پیاده‌سازی راهکارهای ارائه‌شده، بخشی از آن که وظیفه محاسبه تابع شایستگی و بررسی وجود بن‌بست در یک مسیر (پرنده، جسم) را به عهده دارد، به زبان java نوشته شده و در قالب یک استراتژی به ابزار Groove – برگرفته از [۱]- اضافه شده است. عملکرد این استراتژی به این صورت است که یک فایل حاوی یک دنباله از اعداد را که نماینده یک ذره در راهکارهای پیشنهادی است، به‌عنوان ورودی دریافت کرده، مشخص می‌کند که آیا این مسیر، حاوی بن‌بست است یا خیر. اگر حاوی بن‌بست نبود، میزان شایستگی آن ذره را محاسبه کرده و به‌عنوان پارامتر خروجی برمی‌گرداند.

سایر بخش‌ها که وظیفه پیاده‌سازی الگوریتم پرندگان و نیز الگوریتم جستجوی گرانشی را به عهده دارند، به زبان C# و در محیط NET. پیاده‌سازی شده‌اند. به‌این‌ترتیب که، ابتدا در الگوریتم پیاده‌سازی‌شده در C#، جمعیت اولیه به‌طور تصادفی ایجاد می‌شود، سپس هریک از این ذرات، جهت تعیین اینکه حاوی بن‌بست است یا

خیر و محاسبه میزان شایستگی، به نرم‌افزار Groove ارسال و به استراتژی پیاده‌سازی‌شده در آن تحویل داده می‌شود. پارامترهای خروجی تولید و به C# بازگردانده می‌شود. ادامه مراحل کار الگوریتم‌های پرندگان و الگوریتم ترکیبی، در C# پیاده‌سازی شده است و به همین ترتیب این روند ادامه می‌یابد.

ابزار پیاده‌سازی شده و نیز نرم‌افزار Groove که استراتژی یاد شده به آن اضافه شده است، در آدرس «http://sourceforge.net/projects/pso-gsa-gts/files/» بر روی اینترنت در دسترس هستند.

به‌منظور ارزیابی و بررسی دقت و نحوه عملکرد الگوریتم‌های ارائه‌شده، راهکارهای پیشنهادی بر روی ۴ مسئله معروف در وارسی مدل که ابزار Groove قادر به وارسی و کشف بن‌بست در آن‌ها نیست، مورد آزمایش قرار گرفته‌اند. این ۴ مسئله شامل مسئله غذا خوردن فیلسوف‌ها، مسئله بازی PacMan ، مسئله پازل هشت‌تایی و سیستم جوخه خودرو است. جهت انجام آزمایش‌ها، لازم است ابتدا این سیستم‌ها توسط نرم‌افزار Groove مدل شوند. این مدل‌ها در آدرس «http://sourceforge.net/projects/pso-gsa-gts/files/Models.rar/download» بر روی اینترنت در دسترس هستند. نتایج این آزمایش‌ها با نتایج آن‌ها توسط پژوهش‌های مرتبط دیگر نظیر [۱] و [۴۳] که حوزه موردپژوهش و تمرکز اصلی آن‌ها منطبق با پژوهش حاضر است، به‌عبارت‌دیگر، بر روی مقابله با مشکل انفجار فضای حالت در وارسی مدل سیستم‌های مدل‌شده توسط تبدیل گراف متمرکز شده‌اند، مقایسه شده است. لازم به ذکر است، آزمایش‌ها بر روی سیستمی با پردازنده Core i5 و حافظه 4GB انجام شده است. پارامترهای اولیه جهت پیاده‌سازی و انجام آزمایش‌ها، به شرحی که در جدول ۱ آمده است، انتخاب شده‌اند.

جدول ۱: پارامترهای اولیه

جمعیت	۳۰
تعداد دفعات تکرار الگوریتم	۱۰۰
C1 و C2	۲
w	۰/۸

لازم به ذکر است، نتایج ارائه شده، حاصل میانگین ۲۰ بار اجرای هر راهکار هستند. در جداول نتایج، برای هر راهکار، ستون اول، میانگین زمان پاسخ ۲۰ اجرا و ستون دوم، نسبت تعداد دفعات موفقیت راهکار در کشف بن‌بست، به تعداد دفعات اجرا است. مقایسه بین نتایج حاصل از کاوش مبتنی بر BFS و DFS که استراتژی‌های پیش‌فرض ابزار Groove در کاوش و وارسی فضای حالت مدل هستند، راهکار مبتنی بر الگوریتم ژنتیک (GA) ارائه‌شده در [۱]، راهکار پیشنهادی مبتنی بر الگوریتم پرندگان(PSO) و راهکار مبتنی بر ترکیب الگوریتم پرندگان و الگوریتم جستجوی گرانشی(PSO+GSA) انجام شده است.

۶-۱- مسئله ناهار خوردن فیلسوف‌ها

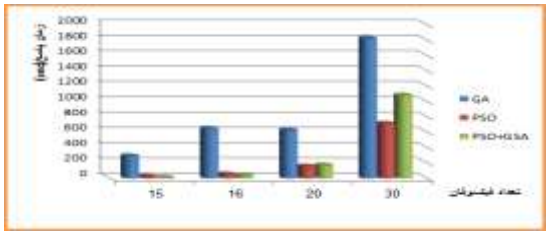
در این مسئله تعدادی فیلسوف دور یک میز نشسته‌اند. هر فیلسوف ابتدا در حال فکر کردن است. سپس گرسنه می‌شود. برای خوردن غذا چنگال سمت چپ، سپس چنگال سمت راست را برمی‌دارد و شروع به غذا خوردن می‌کند، آنگاه چنگال چپ و راست را زمین می‌گذارد و مجدداً شروع به فکرکردن می‌کند و این روال تکرار می‌شود. ازآنجاکه هر چنگال بین دو فیلسوف مجاور مشترک است، اگر همه فیلسوف‌ها چنگال چپ خود را بردارند، سیستم دچار بن‌بست می‌شود. فضای حالت این مسئله با افزایش تعداد فیلسوفان به‌طور نمایی رشد کرده، در نتیجه منجر به انفجار فضای حالت می‌شود و ابزارهای وارسی مدل قادر به تولید کل فضای حالت مسئله و به‌تبع آن، وارسی مدل نمی‌باشند. نتایج حاصل از اجرای الگوریتم‌های ارائه‌شده بر روی این مسئله در جدول ۲ قابل مشاهده است.

جدول ۲: نتایج حاصل از اجرای راهکارهای مختلف بر روی مسئله ناهار خوردن فیلسوف‌ها

	BFS/ DFS	GA		PSO		PSO+GSA	
تعداد فیلسوف		**میانگین زمان پاسخ (sec)**	**#موفقیت/# اجرا**	**میانگین زمان پاسخ (sec)**	**#موفقیت/# اجرا**	**میانگین زمان پاسخ (sec)**	**#موفقیت/# اجرا**
۱۵	**کمبود حافظه**	۲۹۲	۲۰/۲۰	۲۴/۵	۲۰/۲۰	۱۲/۲	۲۰/۲۰
۱۶		۶۵۰	۲۰/۲۰	۵۴/۳	۲۰/۲۰	۴۰/۹	۲۰/۲۰
۲۰		۶۲۸	۲۰/۲۰	۱۵۷/۳	۲۰/۲۰	۱۷۱	۲۰/۲۰
۳۰		۱۸۲۶	۲۰/۲۰	۷۱۴/۳	۲۰/۲۰	۱۰۸۴	۲۰/۲۰

همان‌گونه که جدول ۲ نشان می‌دهد، عملکرد الگوریتم پرندگان و الگوریتم ترکیبی، ازنظر سرعت اجرای رسیدن به حالت خطا، بسیار بهتر از الگوریتم ژنتیک است. روش‌های جستجوی BFS و DFS - از الگوریتم‌های پیش‌فرض مورداستفاده نرم‌افزار Groove- به دلیل ماهیت آن‌ها که بایست کل فضای حالت مسئله را تولید و پس‌ازآن به وارسی مدل بپردازد، با مشکل انفجار فضای حالت مواجه شده و نرم‌افزار قادر به پاسخ‌گویی نیست. همان‌طور که مشاهده می‌شود، الگوریتم‌های ارائه‌شده با هدایت هوشمند تولید فضای حالت حتی در بزرگ‌ترین حالت با مشکل انفجار فضای حالت مواجه نشده و از هر ۲۰ بار اجرای الگوریتم‌ها، هر ۲۰ بار موفق به کشف بن‌بست شده‌اند. در اجرای آزمایش توسط الگوریتم‌های پیشنهادی، حداکثر حافظه‌ای که سیستم در بزرگ‌ترین و پیچیده‌ترین مدل تست‌شده (۳۰ فیلسوف) مصرف می‌کند، حدود ۳۰ مگابایت است، لذا می‌توان از این الگوریتم‌ها در وارسی مدل‌های با فضای حالت بزرگ و پیچیده‌تر نیز

استفاده کرد. نمودار شکل ۶ سرعت این الگوریتم‌ها را در این مسئله با یکدیگر مقایسه کرده است. مقایسه نتایج نشان می‌دهد که در این آزمایش سرعت الگوریتم پرندگان، حدود ۷۲ درصد و سرعت الگوریتم ترکیبی حدود ۶۱ درصد بهتر از الگوریتم ژنتیک است.



شکل ۶: نمودار مقایسه سرعت پاسخ‌گویی الگوریتم ژنتیک، الگوریتم پرندگان و الگوریتم ترکیبی حاصل از ترکیب الگوریتم پرندگان و الگوریتم جستجوی گرانشی بر روی مسئله ناهار خوردن فیلسوف‌ها

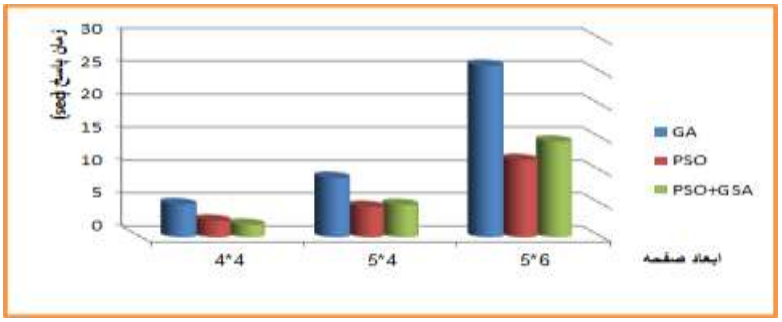
۶-۲- مسئله بازی PacMan

در این بازی، دو عامل pacman و ghost می‌توانند به خانه‌های مجاور بروند. اگر pacman به یکی از خانه‌های مجاور رفت و در آن تیله وجود داشت، آن را می‌خورد و اگر ghost به خانه مجاور رفت و در آن pacman وجود داشت، آن را می‌خورد. بازی زمانی تمام می‌شود که یا تمام تیله‌ها خورده شده باشد که حالت پیروزی است و یا ghost، pacman را بخورد که حالت شکست است. در پیاده‌سازی این مدل، پیروزی و شکست، هر دو به‌عنوان حالت بن‌بست شناخته می شوند. برای انجام این آزمایش، ۴ عدد ghost و ۳ عدد تیله و حداقل ابعاد صفحه بازی، ۴×۴ در نظر گرفته شده است. این ابعاد فضای حالت بزرگ و پیچیده‌ای را ایجاد می‌کند که منجر به مشکل انفجار فضای حالت شده و در نتیجه نرم‌افزار Groove قادر به وارسی فضای حالت این مدل نخواهد بود. نتایج اجرای الگوریتم‌ها بر روی این مسئله در جدول ۳ موجود است. ازآنجاکه در فضای حالت این مسئله، تعداد حالت‌های بن‌بست زیاد است و از مسیرهای مختلف می‌توان به آن‌ها رسید، سرعت رسیدن به حالت خطا و زمان پاسخگویی سریع است. نمودار شکل ۷ سرعت اجرای الگوریتم‌های ذکرشده را در این مسئله با یکدیگر مورد مقایسه قرار داده است.

مقایسه نتایج نشان می‌دهد که در این آزمایش، سرعت رسیدن به یک حالت خطا در مسئله مدل شده، الگوریتم پرندگان، ۵۳ درصد و سرعت الگوریتم ترکیبی ۴۶ درصد بهتر از الگوریتم ژنتیک است.

جدول ۳: نتایج حاصل از اجرای راهکارهای مختلف بر روی مسئله بازی Pacman

	BFS/ DFS	GA		PSO		PSO+GSA	
ابعاد صفحه		میانگین زمان پاسخ (sec)	#موفقیت/# اجرا	میانگین زمان پاسخ (sec)	#موفقیت/# اجرا	میانگین زمان پاسخ (sec)	#موفقیت/# اجرا
۴×۴	کمبود حافظه	۵	۲۰/۲۰	۲/۵	۲۰/۲۰	۱/۹	۲۰/۲۰
۴×۵		۹	۲۰/۲۰	۴/۶	۲۰/۲۰	۴/۹	۲۰/۲۰
۵×۶		۲۶	۲۰/۲۰	۱۱/۷	۲۰/۲۰	۱۴/۵	۲۰/۲۰



شکل ۷: نمودار مقایسه سرعت پاسخ‌گویی الگوریتم ژنتیک، الگوریتم پرندگان و الگوریتم ترکیبی در آزمایش مسئله بازی PacMan

۶-۳- مسئله پازل هشت‌تایی

در مسئله پازل هشت‌تایی، ۸ خانه حاوی مقدار و یک خانه خالی وجود دارد. هدف این است که خانه‌ها به‌صورت مرتب در جای خود قرار گیرند که این حالت، وضعیت بن‌بست است. فضای حالت این مسئله پس از مدل‌سازی در Groove، بسیار گسترده است و Groove قادر به تأمین حافظه موردنیاز جهت وارسی آن نیست. در این مسئله، نحوه چیدمان اولیه اعداد، نقش بسزایی در شانس و زمان رسیدن به هدف ایفا می‌کند، لذا در انجام آزمایش‌های از گراف‌های میزبان با چیدمان متفاوت از اعداد استفاده شده است. اولین حالت، حالتی است که تنها با

یک جابجایی، اعداد مرتب می‌شوند و در حالت‌های بعد، تعداد حرکات لازم برای رسیدن به هدف افزایش یافته است، لذا فضای حالت نیز بسیار گسترده‌تر می‌شود. نتایج اعمال روش‌های پیشنهادی و مقایسه با روش‌های ذکر شده در جدول ۴ و مقایسه سرعت الگوریتم‌های مذکور در نمودار شکل ۸ نمایش داده شده است.

مقایسه نتایج نشان می‌دهد که در این آزمایش، سرعت الگوریتم پرندگان، ۵۶ درصد و سرعت الگوریتم ترکیبی ۶۴ درصد عملکرد بهتری نسبت به الگوریتم ژنتیک داشته است.

جدول ۴: نتایج حاصل از اجرای راهکارهای مختلف بر روی مسئله پازل هشت‌تایی

چیدمان اولیه	BFS/DFS	GA		PSO		PSO+GSA	
		میانگین زمان پاسخ (sec)	#موفقیت/# اجرا	میانگین زمان پاسخ (sec)	#موفقیت# اجرا	میانگین زمان پاسخ (sec)	#موفقت/# اجرا
۳ ۱ ▒ ۶ ۲ ۵ ۸ ۷ ۴	کمبود حافظه	۲۳۹	۲۰/۲۰	۹۴/۷	۲۰/۲۰	۶۲/۵	۲۰/۲۰
۶ ۳ ۲ ۸ ۴ ۱ ۵ ▒ ۷		۳۶۱	۲۰/۲۰	۱۶۵	۲۰/۲۰	۱۴۷/۷	۲۰/۲۰

شکل ۸: نمودار مقایسه سرعت پاسخ‌گویی الگوریتم ژنتیک، الگوریتم پرندگان و الگوریتم ترکیبی حاصل از ترکیب الگوریتم پرندگان و الگوریتم جستجوی گرانشی در آزمایش مسئله پازل هشت‌تایی

۶-۴- مسئله جوخه خودرو[۲۳]

سیستم جوخه خودرو از چندین خودرو تشکیل شده است که با سرعت و فاصله ثابت از یکدیگر در یک بزرگراه حرکت می‌کنند. یک خودرو در جوخه به‌عنوان رهبر[۲۴] است و دیگر خودروها آن را دنبال می‌کنند که به آن‌ها دنبال‌کننده[۲۵] گفته می شود. دنبال کننده‌ها تو سط یک کانال به رهبر متصل‌اند. خصوصیتی که ما در نظر گرفته‌ایم این است که کانال نباید بین دو دنبال‌کننده ایجاد شود. به‌عبارت‌دیگر، یک دنبال‌کننده نمی‌تواند دنبال‌کننده دیگر را دنبال کند. این مسئله نیز به هنگام مدل سازی در ابزار Groove دچار مشکل فضای حالت شده و قادر به کشف بن‌بست و اتمام وارسی مدل نیست. جدول ۵ نتایج اجرای الگوریتم‌های مختلف و مقایسه آن‌ها را نشان می‌دهد. در این آزمایش، به دلیل اینکه کشف بن‌بست در مراحل ابتدایی الگوریتم که مسیرها به‌طور تصادفی تولید و بررسی می‌شوند، صورت می‌گیرد و کار به انجام محاسبات و تکرارهای الگوریتم‌ها کشیده نمی‌شود، زمان‌ها در آزمایش‌ها راهکارهای مختلف، بسیار نزدیک به یکدیگرند.

در [۴۳] نیز، پژوهشگران به‌منظور یافتن حالت شامل دنبال کردن یک دنبال‌کننده توسط دنبال‌کننده دیگر که حالت خطاست، این مسئله را به روش پیشنهادی خود، مورد آزمایش قرار داده‌اند. گرچه محققان، تنها یک مطالعه موردی را مورد آزمایش قرار داده‌اند و نیز به جزئیات کامپیوتری که آزمایش‌ها را بر روی آن انجام داده‌اند اشاره‌ای نکرده‌اند، بااین‌حال مقایسه نتایج این روش با روش‌های پیشنهادی ما در جدول ۶ آمده است.

جدول ۵: نتایج حاصل از اجرای راهکارهای مختلف بر روی مسئله جوخه خودرو

PSO+GSA		PSO		GA		BFS/ DFS	
#موفقیت/#اجرا	میانگین زمان پاسخ(sec)	#موفقیت/#اجرا	میانگین زمان پاسخ(sec)	#موفقیت/#اجرا	میانگین زمان پاسخ(sec)		تعداد خودرو
۲۰/۲۰	۲/۶	۲۰/۲۰	۲/۶	۲۰/۲۰	۲/۹	**کمبود حافظه**	۲۵
۲۰/۲۰	۳۲/۵	۲۰/۲۰	۳۰/۹	۲۰/۲۰	۳۳/۱		۴۹

جدول ۶: نتایج حاصل از اجرای راهکار ارائه‌شده در [۴۳] و راهکارهای پیشنهادی

PSO+GSA	PSO	BMC via SMT	
میانگین زمان پاسخ(sec)	**میانگین زمان پاسخ(sec)**	**میانگین زمان پاسخ(sec)**	**تعداد خودرو**
۲/۴	۲/۸	۱۰۰	۸

۶-۵- ارزیابی توسط آزمون آماری ویلکاکسون

برای آنکه در این پژوهش اثبات شود، بین نتایج حاصل از راهکارهای پیشنهادی، نسبت به راهکار مبتنی بر الگوریتم ژنتیک، تفاوت معنی‌داری وجود دارد، کافی است نتایج توسط آزمونی نظیر Wilcoxon signed-rank ، مورد آزمایش قرار گیرد. در این آزمون که توسط ابزار SPSS قابل انجام است، چنانچه مقدار معناداری (sig.) کم‌تر از ۰/۰۵ باشد، نشانه آن است که تفاوت معنی‌داری بین دودسته از داده‌ها وجود دارد. نتیجه انجام این آزمایش در این پژوهش، در جدول ۷ ارائه شده است. مقدار موردبحث، در مورد داده‌های آزمایش‌های انجام‌شده، در مورد مقایسه راهکار مبتنی بر الگوریتم پرندگان و الگوریتم ژنتیک، برابر ۰/۰۲ و در مورد مقایسه بین راهکار ترکیبی و الگوریتم ژنتیک، برابر ۰/۰۰۸ شد. لذا می‌توان نتیجه گرفت که در هر دو مورد، تفاوت معنی‌داری بین زمان‌های پاسخ حاصل از راهکارهای پیشنهادی و الگوریتم ژنتیک وجود دارد.

جدول ۷: خروجی آزمون Wilcoxon

	VAR0003-VAR0001	VAR0005-VAR0001
Z	۲/۱۹۲[a]	۲/۶۶۶[a]
Asymp.Sig. (2-Tailed)	۰/۰۲۸	۰/۰۰۸

۶-۶- اهمیت راهکار مبتنی بر ترکیب الگوریتم پرندگان و الگوریتم جستجوی گرانشی

با افزایش ابعاد مسائل، وارسی مدل در سیستم تبدیل گراف، با استفاده از الگوریتم‌های ارائه شده، ممکن است دچار کاهش دقت شود. به این معنی که این احتمال وجود دارد در برخی از اجراها، موفق به کشف خطا؛ به‌طور مثال بن‌بست، نشود. به‌طور مثال؛ در مسئله پازل هشت‌تایی، همان‌گونه که قبلاً اشاره شد، ازآنجایی‌که نحوه چیدمان اعداد در گراف میزبان، نقش بسزایی در شانس و زمان رسیدن به هدف ایفا می‌کند، لذا از گراف های میزبان متفاوتی برای انجام آزمایش‌ها استفاده شده است. به‌این‌ترتیب که اولین حالت، حالتی درنظرگرفته شده که تنها با یک جابجایی، اعداد مرتب می‌شوند و در حالت‌های بعد، تعداد حرکات لازم برای رسیدن به هدف افزایش یافته است. چنانچه گراف میزبان، چیدمانی پیچیده و کاملاً به‌هم‌ریخته از این مسئله باشد لذا، تعداد حرکات جابجایی لازم به‌منظور مرتب کردن پازل بسیار زیاد خواهد بود که منجر به فضای حالت بسیار بزرگ و پیچیده‌ای خواهد شد. در این حالت ممکن است الگوریتم‌های ارائه‌شده برای کشف حالت خطا در چنین فضای حالت بزرگ و پیچیده‌ای دچار ضعف شوند و در برخی از اجراها قادر به کشف بن‌بست موجود در فضای حالت مسئله نباشند.

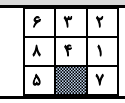
شکل ۹، یک چیدمان پیچیده از پازل هشت‌تایی را نشان می‌دهد. این چیدمان به‌عنوان گراف میزبان در نظر گرفته شده و نتایج حاصل، با الگوریتم‌های مختلف مورد مقایسه قرار گرفته است. این نتایج در جدول ۸ نشان داده شده‌اند.

۶	۱	۲
۴	۷	۳
۵		۸

شکل ۹: یک چیدمان پیچیده از پازل هشت‌تایی

همان‌طور که نتایج جدول ۷ نشان می‌دهند، راهکار پیشنهادی مبتنی بر الگوریتم پرندگان، اگرچه سرعت بهتری نسبت به الگوریتم ژنتیک در کشف بن‌بست دارد، اما مشاهده می‌شود که همانند الگوریتم ژنتیک که در ۵ اجرا از ۲۰ اجرا موفق به کشف بن‌بست نشده است، در ۴ اجرا موفق به کشف بن‌بست نشده است. لذا همان‌گونه که قبلاً اشاره شد، با توجه به اینکه یکی از مهم‌ترین معایب الگوریتم پرندگان، مشکل به دام افتادن در بهینه‌های محلی است، لذا این الگوریتم در جستجوی محلی دچار ضعف است. به نظر می‌رسد، دلیل احتمالی عدم موفقیت این الگوریتم در کشف بن‌بست در برخی از اجراها، وجود همین مشکل باشد. به‌منظور بررسی دلیل احتمالی ذکرشده، از یک الگوریتم دیگر که در جستجوی محلی، پرقدرت‌تر عمل می‌کند، در ترکیب با الگوریتم پرندگان استفاده شده است. این پیشنهاد در ارائه راهکار دوم، ترکیب الگوریتم پرندگان با الگوریتم جستجوی گرانشی، صورت گرفته است. همان‌طور که در ستون آخر جدول ۸ مشاهده می‌شود، الگوریتم پیشنهادی ترکیبی در ۱۹ تا از ۲۰ اجرا، موفق به کشف بن‌بست شده است.

جدول ۸: نتایج حاصل از اجرای راهکارهای مختلف بر روی مسئله پازل هشت‌تایی با شروع از یک حال پیچیده به‌عنوان گراف میزبان

PSO+GSA		PSO		GA		BFS/DFS	
#موفقیت/ #اجرا	میانگین زمان پاسخ(sec)	#موفقیت/ #اجرا	میانگین زمان پاسخ(sec)	#موفقیت/ #اجرا	میانگین زمان پاسخ(sec)		حالت شروع
۱۹/۲۰	۱۵۲۰	۱۶/۲۰	۱۳۴۵	۱۵/۲۰	۱۶۹۲	کمبود حافظه	

به‌عنوان مثالی دیگر، حالت پیچیده‌ای از مسئله غذا خوردن فیلسوف‌ها (۳۸ فیلسوف) که دارای فضای حالت بسیار گسترده‌ای است، مورد آزمایش قرار گرفت. جدول ۹، نتایج به‌دست‌آمده را توسط الگوریتم‌های مختلف نشان می‌دهد. همان‌طور که مشاهده می‌شود، برخلاف راهکار مبتنی بر الگوریتم پرندگان و الگوریتم ژنتیک که در تعداد بیش‌تری از اجراها موفق به کشف بن‌بست نشده‌اند، الگوریتم پیشنهادی ترکیبی، در ۱۸ اجرا از ۲۰ اجرا موفق عمل کرده است.

جدول ۹: نتایج حاصل از اجرای الگوریتم‌ها، بر روی مسئله ناهار خوردن فیلسوف‌ها با گراف میزبان ۳۸ فیلسوف

PSO+GSA		PSO		GA		BFS/ DFS	
#موفقیت/ #اجرا	میانگین زمان پاسخ(sec)	#موفقیت/ #اجرا	میانگین زمان پاسخ(sec)	#موفقیت/ #اجرا	میانگین زمان پاسخ(sec)		حالت شروع
۱۸/۲۰	۲۳۲۲	۱۵/۲۰	۱۹۶۸	۱۴/۲۰	۳۹۳۶	کمبود حافظه	۳۸ فیلسوف

۷- نتیجه‌گیری و کارهای آتی

هدف از این پژوهش، مقابله با مشکل انفجار فضای حالت در وارسی مدل سیستم‌های تبدیل گراف است که در راستای کشف بن‌بست، پیاده‌سازی شده است. در این راستا راهکاری مبتنی بر الگوریتم پرندگان جهت بهینه‌سازی روش‌های موجود ارائه شد. نتایج آزمایش‌ها بیانگر این است که راهکار پیشنهادی، بهینگی محسوسی را در سرعت کشف خطا موجب می‌شود. به‌گونه‌ای که به‌طور میانگین، حدود ۶۰ در صد سرعت بهتری نسبت به الگوریتم ژنتیک در کشف بن‌بست دارد.

بااین‌حال در ابعاد بسیار بزرگ مسائل که فضای حالت، بسیار پیچیده و گسترده می‌شود، گاهی الگوریتم پرندگان در برخی از اجراها به دلیل گیرافتادن در دام بهینه‌های محلی، در کشف خطا ناتوان است. جهت بهبود این مشکل از الگوریتم جستجوی گرانشی به‌عنوان یک الگوریتم پرقدرت در جستجوی محلی، به‌صورت ترکیبی با الگوریتم پرندگان استفاده شده است. نتایج آزمایش‌ها نشان می‌دهند که الگوریتم ترکیبی ارائه‌شده، گامی مؤثر در بهبود این مشکل است. علاوه بر این، حدود ۵۷ درصد سرعت بهتری نسبت به الگوریتم ژنتیک در کشف بن‌بست دارد.

لازم به ذکر است که باوجود راهکارهای ارائه شده، مشکل انفجار فضای حالت مخصوصاً در مسائل با ابعاد بزرگ و پیچیده، هنوز به‌طور کامل حل نشده است و این مشکل همچنان به‌عنوان یک مشکل بحرانی باقی است، اما در راهکارهای ارائه‌شده سعی شده است گامی در راستای مقابله با این مشکل برداشته شود. نتایج راهکارهای ارائه‌شده نشان می‌دهند وارسی مدل‌های با فضای حالت بسیار بزرگ که ابزارهای موجود نظیر Groove قادر به وارسی آن‌ها نبودند را ممکن ساخته‌اند.

نکته دیگر اینکه راهکارهای ارائه‌شده، همانند راهکارهای پیشین این عرصه [۱; ۴۳]، راهکارهای قطعی در کشف بن‌بست نیستند. به این معنی که چنانچه این راهکارها موفق به کشف بن‌بست نشوند با قطعیت نمی‌توان ادعا کرد که مدل تست‌شده حاوی بن‌بست نیست. اما عکس آن برقرار است. یعنی چنانچه بن‌بست کشف شود با قطعیت می‌توان گفت مدل مورد آزمایش حاوی بن‌بست است.

در این پژوهش، فرآیند وارسی مدل در سیستم تبدیل گراف، متمرکز بر کشف ویژگی بن‌بست است و پیاده سازی در این راستا صورت گرفته است. ازآنجایی‌که بن‌بست به‌نوعی یک ویژگی Safety است، لذا می‌توان پژوهش‌های مشابهی را برای بررسی دیگر خصوصیات Safety با ایده گرفتن از این راهکار، انجام داد.

همچنین با توجه به اینکه خصوصیت Reachability، عکس خصوصیت Safety است، با اعمال تغییرات کوچکی در راهکار پیشنهادی، می‌توان خصوصیات Reachability را نیز توسط راهکار ارائه‌شده بررسی کرد.

به‌عنوان پیشنهاد در زمینه کارهای آتی، می‌توان با تغییر در تابع شایستگی و نحوه محاسبه آن، از این راهکار در کشف خصوصیات دیگر نیز بهره گرفت. در حال حاضر برای درستی‌یابی این خصوصیات، لازم است طراح، مدل را به‌گونه‌ای تغییر دهد که خصوصیت مورد انتظار به یک حالت بن‌بست تبدیل شود. در صورت کشف بن‌بست

می‌توان نتیجه گرفت که خصوصیت موردنظر ارضا گردیده است. همچنین ممکن است با انتخاب یک تابع شایستگی جدید نتایج را بهبود بخشید.

در راهکار پیشنهادی، پیاده سازی به‌گونه‌ای انجام شده است که بخشی از عملیات وارسی به‌منظور کشف بن‌بست، در قالب یک نرم‌افزار کاربردی در بستر NET. و مجزا از واسط گرافیکی ابزار Groove طراحی شده است. چرا که پیاده‌سازی الگوریتم‌های پیشنهادی به دلیل تعداد دفعات تکرار و اجرای جداگانه هر ذره یا راه‌حل برای رسیدن به هدف، با توجه به وجود روابط بازگشتی و نحوه پیاده‌سازی ساختار ابزار Groove، به‌سادگی امکان‌پذیر نبود. لذا تصمیم بر آن شد تا در مرحله اول برای بررسی قدرت کارایی الگوریتم پیشنهادی، بخشی از عملیات لازم به‌صورت جدا از ابزار در نظر گرفته شود و این امر باعث ایجاد یک سربار در زمان اجرای الگوریتم شده است. بدیهی است یکپارچه بودن کل عملیات در داخل ابزار وارسی مدل، بسیار مفیدتر و استفاده از آن ساده‌تر خواهد بود. لذا پیشنهاد این است که در کارهای آتی، این قابلیت به‌صورت یکپارچه در ساختار اصلی نرم‌افزار وارسی کننده پیاده‌سازی شود به‌گونه‌ای که از طریق منوهای داخلی ابزار Groove قابل دست‌یابی باشد.

مراجع

[1] C.Baier and J.Katoen, Principles of Model Checking, MIT Press, 2008.

[2] A. Rensink, The GROOVE Simulator: A Tool for State Space Generation, Applications of Graph Transformations with Industrial Relevance (AGTIVE), 3062 LNCS (2003), 2003, pp. 479–485.

[3] H.Peng and S.Tahar, A Survey on Compositional Verification, Journal of circuits, systems, and computers, 1998.

[4] W.D.Roever, The Need for Compositional Proof Systems: A Servey, International Symposium on Compositionality: The Significant Difference, 1998, pp. 1-22.

[5] P.Wolper and P.Godefroid, Using partal orders for the efficient verification of deadlock freedom and safety properties, Formal Methods in System Design - Special issue on computer-aided verification:special methods II, 1991, pp.149-164.

[6] P.Godefroid, Using Partial orders to improve Automatic Verification Methods, 2nd International Workshop on Computer Aided Verification, 1991, pp. 176-185.

[7] E.M.Clarke, R.Enders, T.Filkorn, and S.Jha, Exploiting symmetry in temporal logic model checking, Formal Methods in System Design, 1996, pp. 77-104.

[8] V.Gyuris and A.Sistla, On-the-fly model checking under fairness that exploits symmetry, *Formal Methods in System Design*, 1999, pp. 217-238.

[9] E.A.Emerson, A.PSistla, and Weyl, Symmetry and Model Checking, 1996, pp. 105-131.

[10] G.Francesca, A.Santone, G.Vaglini, and M.L.Villani, Ant Colony Optimization for Deadlock Detection in Concurrent Systems, IEEE Computer Society, 2011, pp. 108-117.

[11] E.Alba and F.Chicano, Ant ColonyOptimization in Model Checking, 11th international conference on Computer aided systems theory, Springer-Verlag, 2007, pp. 523-530.

[12] R.Behjati, M.Sirjani, and M.N.Ahmadabadi, Bounded Rational Search for On-the-Fly Model Checking of LTL Properties, Fundamentals of Software Engineering, 5961 (2010), pp. 292-307.

[13] P.Godefroid and S.Khurshid, Exploring Very Large State Spaces Using Genetic Algorithms, Software Tools for Technology Transfer (STTT) - Special section on tools and algorithms for the construction and analysis of systems, 2004, pp. 117-127.

[14] E.Alba, F.Chicano, M.Ferreira, and J.Gomez-Pulido, Finding deadlocks in large concurrent java programs using genetic algorithms, 10th annual conference on Genetic and evolutionary computation, 2008 ,pp. 1735-1742.

[15] L.M.Duarte, L.Foss, R.Wagner, and T.Heimfarth, Model Checking the Ant Colony Optimisation, Distributed, Parallel and Biologically Inspired SystemsIFIP Advances in Information and Communication Technology, 329 (2010) 221-232.

[16] J.Kennedy and R.Eberhart, Particle Swarm Optimization, Proceedings of IEEE International Conference on Neural Networks, 1995, pp. 1942-1948.

[17] E.Rashedi, H.Nezamabadi-pour, and S.Saryazdi, GSA: A Gravitational Search Algorithm, Information Siences, 2009,pp. 2232-2248.

[18] C.Courcoubetis, M.Vardi, P.Wolper, and M.Yannakakis, Memory-efficient algorithms for the verification of temporal properties, Formal Methods in System Design - Special issue on computer-aided verification: general methods, 1992, pp. 275-288.

[19] B.L.Webster, Solving combinatorial optimization problems using a new algorithm based on gravitational attraction, Florida Institue of Technology, 2004.

[20] D.Dill, U.Stern , A New Scheme for Memory-Efficient Probabilistic Verification, IFIPTC6/WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification, 1996, pp. 333-348.

[21] U.Stern and D.Dill , Improved probabilistic verification by hash compaction, Correct Hardware Design and Verification Methods, 987 (1995), pp. 206-224.

[22] H.Sivaraja and G.Gopalakrishnan, Random Walk Based Heuristic Algorithms for Distributed Memory Model Checking, Electronic Notes in Theoretical Computer Science, 89 (2003), pp. 51-67.

[23] P.Wolper and D.Leroy , Reliable hashing without collision detection, COMPUTER AIDED VERIFICATION.*5TH INTERNATIONAL CONFERENCE*, 1993, pp. 59-70.

[24] C.H.Yang, Prioritized Model Checking in, Stanford University, 1998.

[25] S. Edelkamp, A. L. Lafuente, S. Leue, Directed explicit model checking with HSF-SPIN, SPIN Workshop, 2001, pp. 57-79.

[26] C.Han.Yang. D.L.Dill, Validation with guided search of the state space, The 35th Annual Design Automation Conference, 1998.

[27] S. Edelkamp,S. Jabbar,A. L. Lafuente, Heuristic search for the analysis of graph transition systems, Graph Transformation (ICGT), 2006.

[28] G.J.Holzmann, The Modelchecker SPIN, IEEE Transactions on Software Engineering, 1997, pp. 279-295.

[29] R.Yousefian, V.Rafe, M.Rahmani, A Heuristic Solution for Model Checking Graph Transformation Systems,

Applied Soft Computing, Elsevier, 24(2014), pp. 169-180.

[30] G. Engels, J. H. Hausmann, R. Heckel, S. Sauer, Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML, Springer, 2000, pp. 323-337.

[31] L. Baresi, R. Heckel, S. Thöne, D. Varro ,Modeling and validation of service-oriented architectures: application vs. style, he 9th European Software Engineering Conference Held Jointly With 11th ACM SIGSOFT International Symposium On Foundations Of Software Engineering (ESEC/FSE-11), 2003, pp.68-77.

[32] S.Thone, R.Heckel, Behavioral Refinement of Graph Transformation-Based Models, Electronic Notes In Theoretical Computer Science (ENTCS), (2005), pp. 101-111.

[33] T.Mens, On the Use of Graph Transformations for Model Refactoring, in: Generative andTransformational Techniques in Software Engineering (GTTSE'05), Springer, (2005), pp. 219-257.

[34] G. Taentzer, K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovszky, et al. K.Ehrig, Model Transformation by Graph Transformation: A Comparative Study, *Workshop Model Transformation in Practice, Softwareand System modelings (Sosym)*, 2005.

[35] M.R.Nadaf. V.Rafe, Performance Modeling and Analysis of Software Architectures Specified Through Graph Transformations, Computing and Informatics, (2013), pp. 797-826.

[36] T.Isenberg, Bounded Model Checking of Graph Transformation Systems via SMT Solving, 2014.

[37] N.M.Sabri, M.Puteh, and M.R.Mahmood, "A Review of Gravitational Search Algorithm" ,International Journal of Advance in Soft Computing, 5(2013), pp.1-39.

[38] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "Filter modeling using gravitational search algorithm," Engineering Applications of Artificial Intelligence, 24(2011) , pp. 117–122.

[39] Z.Li-ping, Y.Huan-jun, and H.Shang-xu, Optimalchoice of parameters for particle swarm optimization, Journal of Zhejiang University SCIENCE A, 6 (2005) 528-534.

[40] R.Heckel, Graph Transformation in a Nutshell, Electronic Notes in Theorical Computer Sience (ENTCS), 148 (2006) 187-198.

[41] Y.Shi and R.Eberhart, A modified particle swarm optimizer, in: The International Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence,1998, pp. 69-73.

[42] Z.H.Zhan , S.Yat-sen , J.Xiao , J.Zhang , and W.N.Chen, Adaptive control of acceleration coefficients for particle swarm optimization basedon clustering analysis, Evolutionary Computation(CEC), 2007, pp. 3276-3282.

[43] D.Bratton and J.Kennedy, Defining a Standard for Particle Swarm Optimization, Swarm Intelligence Symposium, 2007. SIS 2007. IEEE, (2007), pp. 120-127.

[44] M.Settles, "An Introduction to Particle Swarm Optimization", University of Idaho, 1999.

زیرنویس‌ها

۱ Compositional Verification
2 Partial Order Reduction
3 Symmetry Reduction
4 Particle Swarm Optimization Algorithm
5 Gravitational Search Algorithm
6 Deadlock
7 Heuristic
8 Exhastive
9 Ant Colony Algorithm
10 Linear Temporal Logic
۱۱ Genetic Algorithm
12 Reactive Systems
13 Meta-Modeling
14 Atchitectural Style Representation
15 Refinement
16 Refactoring
17 Performance Analysis
18 Bounded Model Checking
19 Satisfiability Modulo Theories
20 Forbidden Graph
21 Type Graph
22 Host Graph
23 Car Platoon
24 Leader
25 Follower