نوزدهمین کنفرانس ملی سالانه انجمن کامپیوتر ایران
دانشکده مهندسی کامپیوتر، دانشگاه شهید بهشتی، تهران، ۱۵-۱۳ اسفندماه ۱۳۹۲

یک الگوریتم خود-پایاساز با قابلیت محدودسازی خطا برای ساخت مجموعه غالب مینیمال با هدف خوشه‌بندی در شبکه‌های حسگر بی‌سیم

امیررضا رامتین[۱]، وصال حکمی[۲]، مهدی دهقان[۳]

[۱]دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیرکبیر،
a_ramtin@aut.ac.ir

[۲]دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیرکبیر،
vhakami@aut.ac.ir

[۳] دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیرکبیر،
dehghan@aut.ac.ir

چکیده

در این مقاله، یک الگوریتم با ویژگی‌های خود-پایاسازی و محدودسازی خطا برای ساخت مجموعه غالب مینیمال به منظور خوشه‌بندی در شبکه‌های حسگر بی‌سیم، پیشنهاد می‌شود. الگوریتم‌های مشابه یا به‌کلی از ویژگی محدودسازی خطا بی‌بهره‌اند و یا چون اساساً با دید کاربردی خاصی طراحی نشده‌اند، با محیط عملیاتی شبکه‌های حسگر تناسب ندارند. الگوریتم پیشنهادی قابلیت تنظیم وابستگی گره‌ها به سرخوشه را دارد؛ از پیکربندی‌های تک‌خطایی با پیچیدگی زمانی و مکانی $O(1)$ ترمیم می‌شود و تحت سیاست زمانبندی توزیعی ناعادلانه کار می‌کند که بیشترین مشابهت با معماری شبکه‌های حسگر را دارد. ساخت مجموعه غالب مینیمال، چون عدم مجاورت گره‌های سرخوشه را الزامی نمی‌کند، تعداد تغییر حالت کمتری تا پیکربندی مجاز نیاز دارد و ساختار خوشه‌های حاصل از آن نیز بهینه‌تر است. کاهش تعداد پیام‌های بروزرسانی، ایجاد ساختار خوشه‌بندی کارآمدتر و پایدارسازی با حداقل تغییر در ساختار توپولوژیکی از ویژگی‌های اساسی الگوریتم می‌باشند. نتایج حاصل از شبیه‌سازی نشان خواهد داد که صرف نظر از تعداد و تراکم گره‌ها، روش پیشنهادی علاوه بر ترمیم سریع در مقابل خطاهای مقیاس کوچک، زمانِ رسیدن به پایداری با شروع از پیکربندی دل‌خواه اولیه را نیز نسبت به روش‌های قبلی بهبود می‌دهد.

کلمات کلیدی

خود-پایاسازی، خوشه‌بندی، شبکه‌های حسگر بی‌سیم، صرفه‌جویی انرژی، محدودسازی خطا.

۱- مقدمه

در شبکه‌های حسگر بی‌سیم میزان انرژی گره یک منبع محدود است و تبادل بسته بین گره‌ها بیشترین مصرف انرژی را دارد. در نتیجه کاهش سربار ارتباطی یک راهکار ضروری به نظر می‌رسد. برای نیل به این هدف یک همکاری محلی شده بین گره‌ها نیاز می‌باشد تا میزان تبادلات در شبکه را کاهش دهد. تجزیه کل شبکه به قسمت‌های به‌هم‌متصل خوشه معمول‌ترین راه کار برای رسیدن به این همکاری است. خوشه‌بندی به دلیل ایجاد معماری و مسیریابی سلسله‌مراتبی، شبکه را مقیاس‌پذیر نموده و قابلیت خود-سازماندهی و کنترل ارتباطات را فراهم می‌کند. پویایی شبکه حسگر و نبود یک زیرساختار ثابت، روش‌های خوشه‌بندی با پایداری بالا و برخوردار از قابلیت بازپیکربندی را می‌طلبد.

خود-پایاسازی از رویکردهای مطرح در راستای ایجاد قابلیت تحمّل‌پذیری خطا، به ویژه در سیستمهای توزیعی پویا می‌باشد که اخیراً در حوزه پژوهشی شبکه‌های حسگر نیز مورد توجه ویژه واقع شده است [2]. یک الگوریتم خود-پایدار بدون توجه به حالت اولیه تضمین می‌کند که سیستم بعد از تعداد محدودی گام به طور خودکار و بدون دخالت عامل خارجی به یک حالت معتبر همگرا شود. خوشه‌بندی با ویژگی خود-پایاسازی نه تنها طراحی شبکه را بی‌نیاز از پیکربندی اولیه ساخته، همچنین امکان ترمیم خودکار خطاهای گذرا ناشی از تغییرات محیطی، خرابی حسگرها یا تغییر وضعیت داخلی

آن‌ها را میسر کرده و نسبت به تغییرات پیکربندی و گسستگی ساختار ارتباطی که ناشی از پویایی سیستم است، مصونیت ایجاد می‌کند.

خوشه‌بندی خود-پایاساز همراه با ویژگی محدودسازی حوزه خطا مناسبت بیشتری با مشخصه‌های شبکه‌های حسگر دارد، که به ندرت به آن پرداخته شده است. معمولا درصد زیادی از خطاهای گذرا در یک شبکه حسگر به تنها یک گره محدود می‌شوند. ویژگی محدودسازی حوزه خطا ترمیم سریع‌تر سسیتم خود-پایاساز از کلیه پیکربندی‌های تک‌خطایی را در پی دارد؛ در عین حال که قابلیت بازیابی نهایی سیستم از خطاهای گسترده‌تر تحت تأثیر قرار نمی‌گیرد. لحاظ نمودن قابلیت محدودسازی خطا، تعداد بِروزرسانی‌ها، همه‌پخشی‌ها و تغییر حالت‌ها را کاهش داده و از یک سو موجب صرفه‌جویی در مصرف انرژی و بالا رفتن عمر شبکه می‌گردد و از سوی دیگر، با جلوگیری از انتشار خطا در سطح شبکه، زمان رسیدن به پایداری را کاهش می‌دهد. کاهش تعداد تغییر حالت‌ها از بهم‌ریختگی آرایش توپولوژیکی شبکه جلوگیری نموده و گره‌ها حتی‌الامکان در خوشه خود باقی می‌مانند.

هدف از این مقاله، ارائه الگوریتم خود-پایاسازی جهت خوشه‌بندی شبکه حسگر از طریق ساخت مجموعه غالب مینیمال در گراف نظیر شبکه می‌باشد که در شرایط وقوع خطاهای مقیاس کوچک از گسترش خطا در سطح شبکه جلوگیری کند. جهت خوشه‌بندی در گراف نظیر شبکه از مجموعه مستقل ماکسیمال نیز می‌توان استفاده نمود؛ اما مجموعه غالب از دو نظر به مجموعه مستقل رجحان دارد: الف) وجود متغیر وابستگی در مجموعه غالب موجب می‌شود که در پایان الگوریتم، هر گره، سرخوشه نظیر خود را بداند. ب) می‌توان نشان داد که در اکثر موارد تعداد خوشه‌های حاصل از مجموعه غالب مینیمال و تعداد تغییر حالات تا پایداری، بدلیل عدم وجود محدودیت در مجاورت سرخوشه‌ها، کمتر است. در طراحی الگوریتم پیشنهادی، ویژگی محدودسازی خطا [3,10] جهت محدود ساختن حوزه مکانی و زمانی تأثیر خطاهای مقیاس کوچک، استفاده شده است. در این الگوریتم از طریق حذف حالت میانی که در روش‌های پیشین تعبیه شده، میزان قابلیت دسترسی سیستم افزایش و زمان رسیدن به پایداری کاهش می‌یابد. تدوین قوانین پایداری در طراحی الگوریتم پیشنهادی با هدف تشکیل توپولوژی خوشه‌بندی کارآمدتر نسبت به روش‌های پیشین، صورت گرفته است؛ به گونه‌ای که تعداد گره‌های سرخوشه بی‌فایده در توپولوژی نهایی به حداقل برسد.

ادامه مطالب مقاله بدین شرح است: در بخش ۲، به معرفی اجمالی مفاهیم پایه و مرور کارهای پیشین خواهیم پرداخت. در بخش ۳، الگوریتم پیشنهادی تشریح شده و بُرهان‌های مربوط به بررسی صحت کارکرد آن ارائه می‌گردد. بخش ۴ نیز به ارزیابی و مقایسه عملکرد الگوریتم پیشنهادی اختصاص داده شده است. ارائه نتیجه‌گیری پایان‌بخش مطالب مقاله خواهد بود.

۲- پیشینه نظری و کارهای مرتبط

یک سیستم دارای قابلیت خود-پایاسازی است، اگر و فقط اگر، با شروع از هر وضعیت اولیه دلخواه و با انتخاب قانون‌ها به صورت غیر-قطعی دو شرط برقرار شود: با تعداد متناهی حرکت سیستم به یک پیکربندی مجاز سراسری برسد (همگرایی) و تا زمانی که خطایی رخ ندهد، سیستم در پیکربندی مجاز باقی بماند (بَستار) [1,8].

الگوریتم‌های خود-پایاساز متعددی برای ساخت مجموعه‌های مستقل و غالب در حوزه نظریه گراف ارائه شده است [5]، البته حجم قابل ملاحظه‌ای از این کارها با فرض اِعمال سیاست زمانبندی متمرکز طراحی شده‌اند که عملا پیاده‌سازی آن در شبکه‌های حسگر غیر ممکن می‌باشد. همچنین بدلیل عدم نگاه کاربردی، بیشتر مطالعات توجه خود را تنها به مجموعه مستقل یا نهایتاً ساخت مجموعه غالب بدون در نظر گرفتن متغیر وابستگی معطوف کرده‌اند [9]. در حوزه کارهای مرتبط با ایده مطرح در این مقاله، تنها الگوریتمی که از پیچیدگی زمانی خطی برخوردار بوده و با فرض سیاست زمانبندی توزیعی به حل مسأله می‌پردازد، روش ارائه شده در [4] می‌باشد. قابلیت خود-پایاسازی در [4] از طریق تعبیه حالت‌های میانی در کارکرد الگوریتم صورت گرفته که البته این امر نیز خود به صورت بالقوه منبع خطا محسوب می‌گردد و می‌تواند منجر به کاهش قابلیت دسترس‌پذیری سیستم شود.

از دیگر ایراداتی که به کلیه الگوریتم‌های موجود در این زمینه وارد است، عدم تمایز در چگونگی مدیریت خطاها بر مبنای گستردگی آن‌هاست. در نظر نگرفتن میزان گستردگی خطا در طراحی الگوریتم‌های خوشه‌بندی خود-پایاساز، موجب صرف زمان طولانی تا رسیدن به پایداری(کاهش قابلیت دسترسی سیستم)، بروزرسانی‌های متعدد سرتاسری (هر بروزرسانی معادل یک همه‌پخشی)، مصرف انرژی بیشتر جهت بازگشت به حالت معتبر، به بهم‌ریختگی آرایش شبکه و تغییرات شدیدتر در ساختار توپولوژیکی خوشه‌بندی می‌شود؛ به گونه‌ای که سرخوشه‌ها به ناگاه تعویض شده و گره‌ها ناگزیر می‌بایست به خوشه‌های جدید بپیوندند.

۳- الگوریتم پیشنهادی

در این بخش، یک توصیف سطح بالا از الگوریتم خوشه‌بندی مبتنی بر ساخت مجموعه غالب مینیمال که دارای هر دو ویژگی خود-پایاسازی و محدودسازی خطاست (با نام اختصاری MDSfc) ارائه می‌شود.

۳-۱- شرحی بر طراحی و نحوه کارکرد الگوریتم

اگر فرض شود حالت ۱-خطایی با خطا در گره v تنها با یک تغییر نامطلوب در مقدار یکی از متغیرهای گره v حاصل شده، در چنین

حالتی دو شرط برقرار است: (۱) یکی از قانون‌ها در گره v فعال خواهد بود. (۲) می‌توان با اجرای یکی از قانون‌ها تنها در گره v به پایداری رسید. هدف، شناسایی و رفع حالت‌های ۱-خطایی با اجرای قانون تنها در همان گره خطادار و ممانعت از انتشار خطا با اجرای ناخواسته در گره‌های همسایگی v، $N(v)$، است.

در مجموعه غالب مینیمال دو حالت ۱-خطایی برای متغیر $state$ خواهیم داشت: خطای خارج شدن یک گره عضو از مجموعه (یعنی IN به OUT)، خطای ورود یک گره غیرعضو به داخل مجموعه (یعنی OUT به IN)؛ از ویژگی‌های قابل ملاحظه الگوریتم پیشنهادی حذف حالت میانی WAIT است که در [4] مطرح شده بود. علاوه بر متغیّر $state$، هر گره یک متغیّر کمکی به نام $dependent$ نیز دارد که به گره دیگری اشاره می‌کند. در حالتی که گره خارج از مجموعه تنها یک گره عضو مجموعه در همسایگی خود داشته باشد، $dependent.v = w$ خواهد بود و در صورتی که بیش از یک همسایه عضو داشته باشد، $dependent.v = \emptyset$ خواهد بود. برای گره v عضو مجموعه نیز $dependent.v = \emptyset$. با توجه به توضیحات برای این متغیر سه حالت ۱-خطایی داریم.

تعریف تعدادی گزاره در شکل (۱) آمده است که شامل پیش‌شرط‌های اجرای عملیات تغییر وضعیت در گره‌ها و تعدادی مجموعه برای خوانایی بهتر شِبه‌کُد الگوریتم است.

1. $\boldsymbol{inNeighbor(v)} \equiv \exists w \in N(v): w.state = IN$
2. $\boldsymbol{dependentNeighbors(v)} \equiv \exists w \in N(v): w.dependent = v$
3. $\boldsymbol{conflictIn(v)} \equiv v.state = IN \wedge (dependentNeighbors(v) = \emptyset) \wedge \exists w \in N(v): w.state = IN$
4. $\boldsymbol{Pending(v)} \equiv state.v = OUT \wedge \sim inNeighbor(v)$
5. $\boldsymbol{pendigNeighbors(v)} \equiv w \in N(v): Pending(w)$
6. $\boldsymbol{inNeighborWithLowerId(v)} \equiv \exists w \in N(v): w.state = IN \wedge w.id < v.id$
7. $\boldsymbol{canOut(v)} \equiv confilictIn(v) \wedge \sim dependentNeighbors(v) \wedge \big(execution\ of\ (v.state := OUT) \Longrightarrow (\forall w \in N(v): \sim conflictIn(w))\big)$
8. $\boldsymbol{soloPending(v)} \equiv Pending(v) \wedge \forall w \in N(v): \sim Pending(w)$

شکل (۱) : گزاره‌ها و مجموعه‌های مورد استفاده در شِبه‌کُد الگوریتم MDS*fc*.

قوانین الگوریتم MDS*fc* با هر دو ویژگی خود-پایداری و محدودسازی خطا نیز در شکل (۲) نمایش داده شده‌اند. قانون‌های ۱ تا ۸ مربوط به متغیّر $state$ گره هستند و از میان آن‌ها قانون‌های ۱ تا ۴ برای خارج شدن گره دل‌خواه v از مجموعه می‌باشند. سه قانون‌ها اول تشخیص حالت ۱-خطایی است و قانون ۴ تشخیص عدم وجود حالت ۱-خطایی در گره v و همسایه‌های آن است. گزاره $confilictIn(v)$ در تمامی این قوانین برقرار است (شرط لازم).

R1. $canOut(v) \wedge (\forall w \in N(v): \sim canOut(w)) \rightarrow v.state := OUT$
R2. $canOut(v) \wedge (|inNeighbor(v)| > \max_{\neg}(\forall w \in N(v): canOut(w))\ |inNeighbor(w)|\ ) \rightarrow v.state := OUT$
R3. $canOut(v) \wedge (|inNeighbor(v)| = \max_{\neg}(\forall w \in N(v): canOut(w))\ |inNeighbor(w)|\ ) \wedge v.id > w.id \rightarrow v.state := OUT$
R4. $conflictIn(v) \wedge \sim canOut(v) \wedge (\forall w \in N(v): \sim canOut(w)) \wedge inNeighborWithLowerId(v) \rightarrow v.state := OUT$
R5. $Pending(v) \wedge (|pendingNeighbors(v)| > (max)_{\neg}(\forall w \in pendingNeighbors(v)\ )\ |pendingNeighbors(w)|\ ) \rightarrow v.state := IN, dependent.v := \emptyset$
R6. $Pending(v) \wedge (|pendingNeighbors(v)| = (max)_{\neg}(\forall w \in pendingNeighbors(v)\ )\ |pendingNeighbors(w)|\ ) \wedge (dependent.v = \emptyset) \rightarrow v.state := IN, dependent.v := \emptyset$
R7. $Pending(v) \wedge (|pendingNeighbors(v)| = (max)_{\neg}(\forall w \in pendingNeighbors(v)\ )\ |pendingNeighbors(w)|\ ) \wedge (dependent.v \neq \emptyset) \wedge (v.id < w.id) \rightarrow v.state := IN, dependent.v := \emptyset$
R8. $solePending(v) \rightarrow state.v := IN, dependent.v := \emptyset$
R9. $(state.v = IN) \wedge (dependent.v \neq \emptyset) \wedge \sim canOut(v) \rightarrow dependent.v := \emptyset$
R10. $(state.v = IN) \wedge (dependent.v \neq \emptyset) \wedge canOut(v) \wedge (|inNeighbor(v)| < \max_{\neg}(\forall w \in N(v): canOut(w)\ )\ |inNeighbor(w)|\ ) \rightarrow dependent.v := \emptyset$
R11. $(state.v = IN) \wedge (dependent.v \neq \emptyset) \wedge canOut(v) \wedge (|inNeighbor(v)| < \max_{\neg}(\forall w \in N(v): canOut(w))\ |inNeighbor(w)|\ ) \wedge v.id < w.id \rightarrow dependent.v := \emptyset$
R12. $(state.v = OUT) \wedge (|inNeighbor(v)| = 1) \wedge (dependent \neq w) \wedge \sim canOut(w) \rightarrow dependent.v := w$
R13. $(state.v = OUT) \wedge (|inNeighbor(v)| = 1) \wedge (dependent \neq w)\ \wedge (\max_{\neg}(\forall w \in N(v): canOut(w))\ |inNeighbor(w)| < \max_{\neg}(\forall u \in N(w): canOut(u))\ |inNeighbor(u)|\ ) \rightarrow dependent.v := w$
R14. $(state.v = OUT) \wedge (|inNeighbor(v)| = 1) \wedge (dependent.v \neq w) \wedge (\max_{\neg}(\forall w \in N(v): canOut(w))\ |inNeighbor(w)| = \max_{\neg}(\forall u \in N(w): canOut(u))\ |inNeighbor(u)|\ ) \wedge (w.id < u.id) \rightarrow dependent.v := w$
R15. $(state.v = OUT) \wedge (|inNeighbor(v)| > 1) \wedge (dependent \neq \emptyset) \wedge \sim canOut(w) \rightarrow dependent.v := \emptyset$
R16. $(state.v = OUT) \wedge (|inNeighbor(v)| > 1) \wedge (dependent \neq \emptyset) \wedge (\max_{\neg}(\forall w \in N(v): canOut(w))\ |inNeighbor(w)| < \max_{\neg}(\forall u \in N(w): canOut(u))\ |inNeighbor(u)|\ ) \rightarrow dependent.v := \emptyset$
R17. $(state.v = OUT) \wedge (|inNeighbor(v)| > 1) \wedge (dependent \neq \emptyset) \wedge (\max_{\neg}(\forall w \in N(v): canOut(w))\ |inNeighbor(w)| = \max_{\neg}(\forall u \in N(w): canOut(u))\ |inNeighbor(u)|\ ) \wedge (w.id < u.id) \rightarrow dependent.v := \emptyset$

شکل (۲) : قوانین تعبیه شده در طراحی الگوریتم MDS*fc*.

در قانون ۱، حالت ۱-خطایی تنها در گره v و نه در هیچ یک از همسایه‌هایش تشخیص داده می‌شود که تحت این شرایط، برای رسیدن به حالت پایدار تنها کافی است گره v تغییر حالت بدهد و از مجموعه خارج شود. قانون ۲ هنگامی فعال می‌شود که حالت ۱-خطایی هم در گره v و هم در گره(های) همسایه v، مثل w، باشد. بدین معنی که هر کدام از دو گره v یا w از مجموعه خارج شوند، به شرایط پایداری برسیم، اما باید توجه شود که خروج همزمان این دو گره با هم شرایط ناپایداری را رقم می‌زند؛ به منظور شکستن تقارن در قانون ۲ به بررسی این امر پرداختیم که گره‌ای که تعداد همسایه‌های عضوِ بیشتری دارد از مجموعه خارج شود؛ این کار، علاوه بر جلوگیری از تغییر حالت همزمان دو گره، از نظر ساختار توپولوژیکی نیز شکل بهتری را نتیجه می‌دهد. در قانون ۳ نیز باز به منظور شکستِ تقارن، و این بار برای حالتی که علاوه بر ۱-خطایی در گره v و گره(های) همسایه‌اش تعداد همسایه‌های عضوِ هر دو به یک مقدار است، از مقایسه شماره شناسه‌ی دو گره بهره بردیم. توجه به این امر، ضروری است که در هر یک از این قوانین تصمیم به حرکت تنها برای همان گره قابل اجراست و منظور از اجرای قانون در گره همسایه، عدم حرکت گره جاری تا لحظه‌ایست که زمان‌بند به گره همسایه برسد و حرکتش را انجام دهد. قانون ۴ بیان‌گر وضعیتی است که حالت ۱-خطایی نه در گره v و نه در هیچ یک از همسایگانش مشاهده نمی‌شود. اما در شرایط نامعتبر قرار داریم، یعنی $confilictIn(v)$ برقرار است. در نتیجه گره v برای خروج از این وضع از مجموعه خارج می‌شود.

قانون‌های ۵ تا ۸، مربوط به وارد شدن گره دل‌خواه v به مجموعه هستند، یعنی تغییر وضعیت از OUT به IN. در تمامی این قانون‌ها گزاره $Pending(v)$ برقرار است. در قانون ۵ بررسی می‌شود تا گره‌ای عضو مجموعه شود که بیشترین گره را از حالت $pending$ خارج کند. در صورتی که این تعداد برابر بود، از قانون ۶ و ۷ برای شکستن تقارن استفاده می‌شود. قانون‌های ۹ تا ۱۱ برای گره‌هایی که عضو مجموعه هستند و $dependent$ آن‌ها $\emptyset$ نیست بکار می‌روند و در صورت فعال شدن مقدار آن را $\emptyset$ می‌کنند. قانون‌های ۱۲ تا ۱۴ برای گره‌هایی که عضو مجموعه نیستند، و دقیقاً یک همسایه عضو دارند ولی $dependent$ آن‌ها برابر آن همسایه نیست، بکار می‌روند در صورت فعال شدن مقدار آن را معادل آن همسایه قرار می‌دهند. قانون‌های ۱۵ تا ۱۷ برای گره‌هایی که عضو مجموعه نیستند، اما بیش از یک همسایه عضو دارند و $dependent$ آن‌ها $\emptyset$ نیست بکار می‌روند در صورت فعال شدن مقدار آن را $\emptyset$ می‌کنند.

۳-۲- اثبات درستی الگوریتم

لِم ۱ (شرط بَستار): در پیکربندی‌ای که هیچ گره‌ای فعال نمی‌باشد، مجموعه $D = \{v|v.state = \text{IN}\}$ یک مجموعه غالب مینیمال می‌سازد و در حالتی که گره خارج از مجموعه تنها یک گره عضو مجموعه در همسایگی خود داشته باشد، متغیر وابستگی به آن گره اشاره می‌کند و در صورتی که بیش از یک همسایه عضو داشته باشد، $\emptyset$ خواهد بود.

این لِم به سادگی از طریق برهان خلف اثبات می‌گردد.

لِم ۲: در حین اجرای الگوریتم، گره v با حالت اولیه دلخواه، حداکثر ۹ حرکت (تغییر حالت) انجام خواهد داد، و بعد از آن هیچ قانون دیگری را اجرا نمی‌کند.

اثبات: به آسانی از طریق کشیدن دیاگرام حالت سیستم می‌توان اثبات کرد؛ که به دلیل محدودیت فضا از تشریح آن در این بخش صرف نظر شده است.

قضیه ۱ (شرط همگرایی): با شروع از هر پیکربندی دل‌خواه و طی تعداد متناهی گام، دیگر هیچ گره‌ای فعال نخواهد بود و مجموعه $D = \{v|v.state = IN\}$ غالب مینیمال است.

اثبات: در شروع از هر پیکربندی دل‌خواه با توجه به لِم ۲، هر گره بعد از حداکثر ۹ گام دیگر قانونِ فعالی نخواهد داشت. از سویی، طبق لِم ۱، در پیکربندی‌ای که در آن هیچ گره‌ای فعال نباشد، D یک مجموعه غالب مینیمال خواهد بود.

لِم ۳: در یک پیکربندی مجاز وقوع یک خطا در وضعیت گره دل‌خواه j تنها با $O(1)$ تغییر وضعیت به پیکربندی مجاز برمی‌گردیم.

اثبات: دو حالت می‌تواند رخ دهد: (الف) در نتیجه وقوع خطا گره j از وضعیت OUT به وضعیت IN تغییر حالت داده است. از آنجا که در پیکربندی موجود تنها همین خطا را داریم، $confilictIn(j)$ و در نتیجه $canOut(j)$ درست هستند. ولی در هیچ گره همسایه j، $canOut(j)$ برقرار نیست، چرا که یا $state.i = OUT$ و یا $dependent.i = \emptyset$، که در هر دو صورت $canOut(i)$ برقرار نخواهد بود. قانون ۱ تنها قانون فعال برای گره j می‌باشد: قانون‌های ۲، ۳، ۱۰ و ۱۱ بدلیل وجود $canOut(i)$ فعال نمی‌شوند. قانون‌های ۴ و ۹ بدلیل وجود $\sim canOut(j)$ فعال نمی‌شوند. قانون‌های ۵-۸ و ۱۲-۱۷ بدلیل وجود $state.j = OUT$ فعال نمی‌شوند. از دیگر سو برای هیچ گره i، $i \in N(j): state.i = IN$، هیچ قانونی فعال نیست: قانون‌های ۱-۳ بدلیل وجود $canOut(i)$ فعال نمی‌شوند. قانون ۴ بدلیل وجود $\sim canOut(j)$ فعال نمی‌شود. قانون‌های ۵-۸ و ۱۲-۱۷ بدلیل وجود $state.i = OUT$ فعال نمی‌شوند. قانون‌های ۹-۱۱ بدلیل وجود $dependent.i \neq \emptyset$ فعال نمی‌شوند. پس تنها با اجرای قانون ۱ در گره j به حالت مجاز و پایدار برمی‌گردیم. (ب) در نتیجه وقوع خطا گره j از وضعیت IN به وضعیت OUT تغییرحالت داده است. از آنجا که پیکربندی مجاز بوده و تنها همین خطا را داریم، $Pending(j)$ برقرار خواهد بود.

اگر تمام همسایگان j همسایه‌ای با وضعیت IN دارند، $soloPending(j)$ برقرار است و در این شرایط تنها قانون ۸ برای گره j فعال خواهد شد و دیگر گره‌ها قانونِ فعالی نخواهند؛ بنابراین تنها با یک تغییر وضعیت در گره j به پیکربندی مجاز بازخواهیم گشت. اگر همسایه یا همسایگانی از j وجود دارند که هیچ همسایهٔ عضوی ندارند، برای مثال گره i در همسایگی j؛ در این حالت گزاره‌های $Pending(j)$ و $Pending(i)$ برقرار خواهند بود. حال یا تعداد اعضای $pendingNeighbors$ در گره i نسبت به گره j بیشتر است؛ در این صورت تنها در گره i قانون ۵ فعال خواهد شد و با یک حرکت به پیکربندی مجاز بازخواهیم گشت؛ و یا در غیر این صورت تنها در گره j قانون ۵ یا ۶ فعال می‌شود و با یک حرکت به پیکربندی مجاز باز می‌گردیم.

لِم ۴: در یک پیکربندی مجاز وقوع یک خطا در وابستگی گره دل‌خواه j تنها با $O(1)$ حرکت در وابستگی به پیکربندی مجاز برمی‌گردیم.

اثبات: در صورت بروز خطا در وابستگی گره یکی از موارد زیر را خواهیم داشت:

- $dependent.j \neq \emptyset \wedge state.j = \text{IN}$
- $dependent.j \neq \emptyset \wedge state.j = \text{OUT} \wedge \exists i,k \in N(j): state.k = state.i = \text{IN}$
- $dependent.j = \emptyset \wedge state.j = \text{OUT} \wedge \exists! i \in N(j): state.i = \text{IN}$

در مورد اول، قانون ۹ تنها قانون فعال برای گره j می‌باشد: قانون‌های ۱-۴ و ۱۰-۱۱ بدلیل $confilictIn(v)$ (داخل گزاره $canOut(j)$) فعال نمی‌شوند. قانون‌های ۵-۸ بدلیل وجود $Pending(j)$ فعال نمی‌شوند. قانون‌های ۱۲-۱۷ بدلیل وجود $state.j = \text{OUT}$ فعال نمی‌شوند. از دیگر سو برای هیچ گره i، $i \in N(j): state.i = \text{IN}$، هیچ قانونی فعال نیست: قانون‌های ۱-۴ بدلیل وجود $confilictIn(v)$ فعال نمی‌شوند. قانون‌های ۵-۸ و ۱۲-۱۷ بدلیل وجود $state.i = \text{OUT}$ فعال نمی‌شوند. قانون‌های ۹-۱۱ بدلیل وجود $dependent.i \neq \emptyset$ فعال نمی‌شوند. برای موارد دوم و سوم نیز به ترتیب تنها قانون ۱۵ و ۱۲ برای گره j فعال می‌باشد. این امر را نیز بسادگی مانند حالت قبل می‌توان نشان داد.

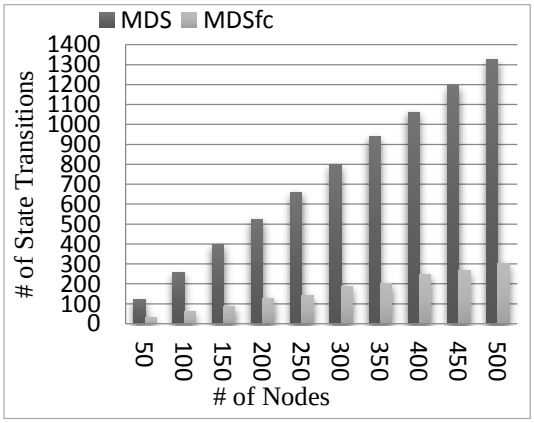
قضیه ۲ (شرط محدودسازی): با شروع از هر پیکربندی ۱-خطایی، بعد از $O(1)$ به پایداری می‌رسیم.

اثبات: در الگوریتم MDSfc، هر گره دو متغیّر حالت عضویت و وابستگی دارد. با توجه به *لِم ۳*، در حالت ۱-خطایی در وضعیت عضویت گره، الگوریتم با یک حرکت به حالت پایدار باز می‌گردد. برای دیگر متغیّر (کمکی) گره یعنی وابستگی نیز در *لِم ۴* نشان دادیم حالت ۱-خطایی در این متغیّر با یک حرکت در همان گره خطادار به پایداری باز می‌گردد.

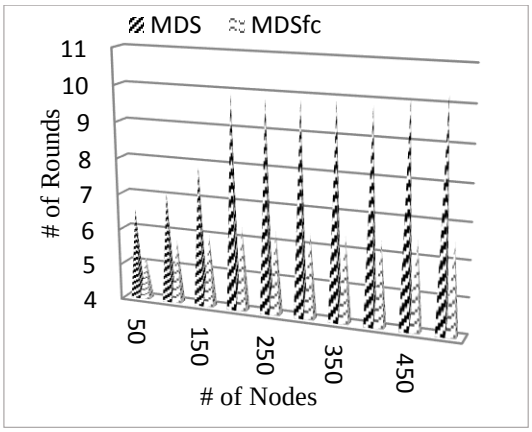
۴- ارزیابی کارایی

در این بخش، کارایی الگوریتم خوشه‌بندی MDSfc به لحاظ تعداد تغییر حالت‌ها، تعداد سیکل‌های زمانی لازم تا پایداری و ساختار توپولوژیکی خوشه‌بندی با الگوریتم ارائه شده در [4] (MDS) مقایسه می‌شود. روال کلی انجام شبیه‌سازی‌ها بر دو مبنای شروع از پیکربندی اولیه (همه گره‌ها در حالت OUT و متغیّر وابستگی‌شان تهی) و شروع از پیکربندی چندخطایی بوده که تحت سیاست زمانبندی توزیعی ناعادلانه تحت آزمایش قرار می‌گیرند. کلیه آزمایشات شبیه‌سازی با استفاده از نرم‌افزار MATLAB صورت گرفته و نتایج به ازای ۱۰۰ مرتبه آزمایش در هر سناریو گزارش می‌شود.

شکل‌های (۳) و (۴)، مقایسه عملکرد MDS و MDSfc از طریق مقایسه تعداد کل تغییر حالت‌ها تحت سیاست زمانبندی توزیع‌شده ناعادلانه نمایش می‌دهد که در آن متوسط درجه همبندی به طور ثابت ۷ فرض شده و تعداد گره‌ها متغیّر می‌باشد. تعداد کل تغییر حالت‌ها در این سناریو برابر است با مجموع تعداد تغییر وضعیت عضویت‌ها بعلاوه تعداد تغییر وضعیت وابستگی‌ها.



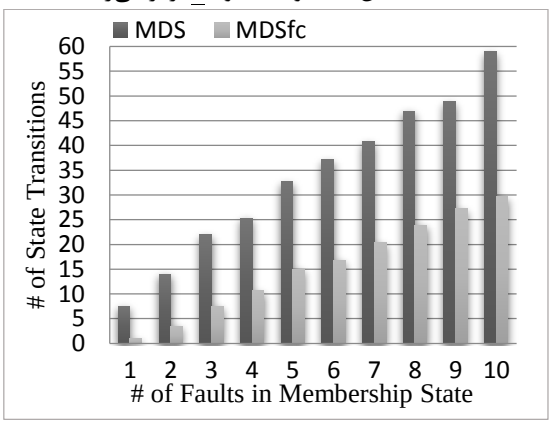
شکل (۳) : مقایسه تعداد تغییر حالت‌ها در MDSfc و MDS.



شکل (۴) : مقایسه تعداد سیکل‌های زمانی در MDSfc و MDS.

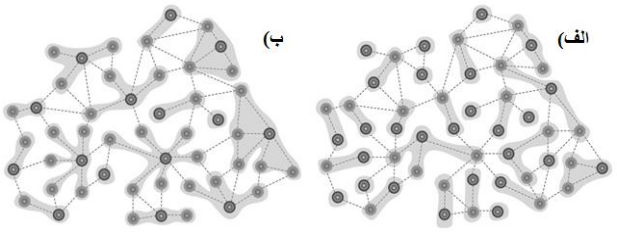
از نمودارهـا، می‌توان نتیجه گرفت که نسـبت کارایی MDSfc به MDS با افزایش تعداد گره‌ها بیشـتر می‌شـود. همچنین، صرف نظر از مقدار درجه همبندی، نسـبت تغییر حالت به تعداد گره‌ها در MDS همواره برابر ثابت ۳ می‌باشـد؛ در حالیکه این نسبت در MDSfc کمتر از ۱ است.

در سـنـاریویی دیگر، عملکرد این دو الگوریتم بـا تزریق خطا به پیکربندی پایدار مقایسـه شده است؛ در این آزمایش که نمودار مربوط به آن در شـکل (۵) نشـان داده شـده، تعداد خطاها متغیّر و از ۱ تا ۷ خطا روی یک توپولوژی متشکل از ۱۰۰ گره و متوسط درجه همبندی ۷ می‌باشـد. الگوریتم MDS*fc* حالت‌های تک‌خطایی را تنها طی یک تغییر حالت به پایداری می‌رساند؛ در حالیکه، MDS به طور متوسط به ۶ تغییر حالت نیازمند اسـت. با افزایش تعداد خطاها نسـبت کارایی MDS*fc* به MDS کاهش یافته و به حدود ۲ برابر می‌رسد.



شکل (۵) : تعداد تغییر حالت‌ها در MDS*fc* و MDS (با تزریق خطا به متغیر عضویت در پیکربندی).

شـکل (۶) سـاختار خوشـه‌بندی حاصـل از اجرای دو الگوریتم MDS*fc* و MDS را روی یـک توپولوژی اولیـه متشـکـل از ۵۰ گره نمایش می‌دهد. تعداد خوشه‌های ایجاد شده در نتیجه اجرای الگوریتم MDS ۲۷ عدد می‌باشـد که از این میان، تعداد ۱۰ خوشـه به صـورت تک‌عضوی و تنها شامل یک گره سرخوشه هستند. این در حالیست که ماحصل اجرای الگوریتم MDS*fc*، تعداد کل ۱۵ خوشه و تنها ۲ خوشه تک‌عضوی خواهد بود.



شکل (۶) : مقایسه توپولوژی حاصل از MDS و MDS*fc*؛ (الف): MDS، (ب): MDS*fc*

۵- نتیجه‌گیری

خصـوصـیات شـبکه‌های حسـگر از یک سـو نیاز به یک معماری سـلسله‌مراتبی نظیر خوشه‌بندی را ایجاب می‌کند و سویی دیگر بدلیل عدم دسـترسـی به سـاختار بعد از کارگذاری شبکه، امکان برخورداری شـبکه از قابلیت بازپیکربندی بدون دخالت خارجی را می‌طلبد. در این مقاله، یک روش خوشـه‌بندی مبتنی بر ساخت مجموعه غالب مینیمال با هر دو قابلیت خود-پایاسازی و محدودسازی حوزه خطا ارائه شده که علاوه بر پایداری سریع در مقابل خطاهای مقیاس کوچک، زمانِ رسیدن به پایداری با شروع از پیکربندی دل‌خواه اولیه را نیز بهبود می‌دهد. در مطـالعات آتی می‌توان طراحی الگوریتم مجموعه غالب متصـل خود-پایاساز توأم با محدودسازی حوزه خطا را هدف قرار داد. مجموعه غالب متصـل برای ایجاد سـتون‌فقرات مجازی در شـبکه حسـگر اسـتفاده می‌شود.

مراجع

[1] S. Tixeuil, "Self-stabilizing algorithms," *Algorithms and theory of computation handbook*, M. J. Atallah and M. Blanton, Eds. Chapman & Hall/CRC, 2010, pp. 26–26.

[2] V. Turau and C. Weyer,"Fault tolerance in wireless sensor networks through self-stabilisation," *Int. J. Communication Networks and Distributed Systems*, Vol.2, No. 1, pp. 78-98, 2009.

[3] S. Ghosh, et al, "Fault-containing self-stabilizing distributed protocols," *Distributed Computing*, vol. 20, no. 1, pp. 53–73, 2007.

[4] V. Turau, "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler," *Information Processing Letters*, Vol. 103, pp. 88-93, 2007.

[5] N. Guellati and H. Kheddouci, "A Survey on Self-Stabilizing Algorithms for Independence, Domination, Coloring, and Matching in Graphs," *J. of Parallel and Distributed Computing*, Vol. 70, No. 4, pp. 406-415, 2010.

[6] J. C. Lin and T. C. Huang, "An Efficient Fault-Containing Self-Stabilizing Algorithm for Finding a Maximal Independent Set," *IEEE Trans. Parallel and Distributed Systems*, Vol. 14, pp. 742-754, 2003.

[7] S. M. Hedetniemi, et al., "Self-stabilizing Algorithms for Minimal Dominating Sets and Maximal Independent Sets," *Computers & Mathematics with Applications*, Vol. 46, pp. 805-811, 2003.

[8] B. Hauck, "Time- and space-efficient self-stabilizing algorithms," Technische Universität Hamburg-Harburg, Denickestr. 22, 21071 Hamburg, 2012.

[9] Z. Xu, et al., "A Synchronous Self-stabilizing Minimal Domination Protocol in an Arbitrary Network Graph," pp. 832-832, 2003.

[10] A. Dasgupta, "Extensions and refinements of stabilization," PhD Theses, University of Iowa, 2009.