Polynomial Silent Self-Stabilizing Maximal p-Star Decomposition

Mohammed Haddad, Colette Johnen, Sven Köhler

Polynomial Silent Self-Stabilizing Maximal p-Star Decomposition

Mohammed Haddad[1] et Colette Johnen[2] [†] et Sven Köhler[3] [‡]

[1] *Université Claude Bernard Lyon 1, Villeurbanne, France*
[2] *Université of Bordeaux, Talence Cedex, France*
[3] *University of Freiburg, Freiburg, Germany*

Une p-étoile est un arbre de hauteur 1 avec exactement p feuilles. Une décomposition maximale d'un graphe en p-étoiles disjointes est une partition des noeuds du graphe telle que toutes les parties sauf au plus une sont des p-étoiles. L'unique partie n'étant pas une p-étoile, ne contient pas de p-étoile.

Nous présentons un algorithme auto-stabilisant silencieux qui réalise une décomposition maximale du graphe en en p-étoiles disjointes quel que soit l'ordonnancement des actions. Le nombre maximum d'actions nécessaire à la construction des p-étoiles est : $12\Delta.m + O(m)$. De plus, nous avons établi un borne sur le nombre de rounds : $5\lfloor\frac{n}{p+1}\rfloor + 5$. Ce travail a été publié (en version courte) dans [4].

Mots-clefs : algorithmique distribuée, auto-stabilisation, décomposition de graphe, p-étoile, temps de convergence

Introduction

Fault-tolerance is among the most important requirements for distributed systems. Self-stabilization is a fault-tolerance technique that deals with transient faults. Starting in an arbitrary configuration, a self-stabilizing distributed system converges to a legitimate configuration in finite time without any external intervention. This makes self-stabilization an elegant approach for non-masking fault-tolerance.

An H-decomposition of a graph G subdivides a graph into disjoint components which are isomorphic to H. A p-star is a complete bipartite graph with a set containing a single node (called the center) and the other set containing p nodes (the leaves).

One of the famous and well studied graph decompositions in literature is star decomposition. A decomposition of a graph into stars is a way of expressing the graph as the union of disjoint stars [2]. The problem of star decomposition has several applications including scientific computing, scheduling, load balancing, parallel computing [1], and important nodes detection in social networks [6]. Decomposing a graph into stars is also used in parallel computing and programming.

Observe that the maximal node-disjoint p-star decomposition problem when restricted to $p = 1$ is equivalent to the maximal matching problem in graphs. Thus the general problem when $p \geq 1$ is NP-complete since generalized matching is proved to be NP-complete in [5]. The maximal matching problem has received much interest due the abundant number of applications in fields as diverse as transversal theory, assignment problems, network flows, and scheduling. Many studies have addressed this problem even in the field of self-stabilization ; the best known move complexity for maximal matching problem is $O(m)$ and was obtained by [7] where m is the number of edges.

The first self-stabilizing algorithm for the maximal node-disjoint p-star decomposition problem was proposed in [9]. This algorithm converges in $2\lfloor\frac{n}{p+1}\rfloor + 2$ rounds under any scheduler. However, the algorithm proposed in [9] always converges to a unique legitimate configuration according to the input graph and does not guarantee a polynomial move complexity. Another algorithm was proposed in [8] where authors dealt with the uniqueness of legitimate state and proved their algorithm to converge within $O(\Delta^2 m)$ moves

[†] This study has been partially supported by the ANR projects DESCARTES (ANR-16-CE40-0023), ESTATE (ANR-16-CE25-0009)
[‡] This study has been partially supported by Sustainability Center Freiburg, Germany

under the unfair distributed scheduler where Δ is maximum node degree in the graph. A bound on the round complexity of the algorithm in [8] is not given.

Our Results. In this paper, we propose the first algorithm on which an upper bound on the round and move complexity are given. Our algorithm matchs the round complexity of the algorithm of [9] : our algorithm converges in at most $5\lfloor\frac{n}{p+1}\rfloor+5$ moves. Moreover, our algorithm improve the move complexity of the algorithm of [8] to $12\Delta m+O(m+n)$.

Our algorithm does not converge to an unique legitimate configuration. In fact, there is a legitimate configuration for any valid maximal p-star decomposition. The above results hold with respect to the unfair distributed scheduler, the most powerful adversary. For the definition of the computational model, we direct the reader at [3].

Silent self-stabilizing maximal node-disjoint p-Star Decomposition Algorithm

Algorithm 1 contains the code of our silent self-stabilizing maximal node-disjoint p-Star decomposition. The algorithm is presented as a set of of rules, each of the form $guard(v) \rightarrow action$. In this section, we give a rough overview of how the algorithm works. in which Node v is the center of a well formed p-star if and only if v verifies $correctCenter(v)$; the nodes of $leaves(u)$ are the leaves of the p-star centered at v. We denote $star(v)$ the set of nodes in the p-star centered in v ; the formal definition being if $correctCenter(v)$, $star(v)=\{v\}\cup leaves(v)$ and $star(v)=\emptyset$ otherwise. Every node in $star(v)$ verifies the predicate *correctLeaf* or *correctCenter*.

Each node v indicates to their neighbors whether they are a member of a star or not via the shared Boolean variable $inStar(v)$. The values of the shared variables *center* and *leaves* in its neighborhood are enough to allow a node v to determine the value of the predicates $correctCenter(v)$ and $correctLeaf(v)$; so they suffice to compute the value of $isInStar(v)$.

In addition, each node indicates whether it may be a *viable center* of a new star using the shared Boolean variable $viableCenter(v)$. That is the case only if the node itself and p of its neighbors are not a member of a star, yet. The values of the shared variables $inStar(v)$ in its neighborhood are enough to allow a node v to determine the value of $viableCenter(v)$.

The rule RU is responsible for correcting initial inconsistencies such as invalid identifiers in $leaves(v)$ and for the updating of the variables $inStar(v)$ and $viableCenter(v)$ if the other rules are not enabled. So if the value of $inStar(v)$ or $viableCenter(v)$ are not accurate, then v is enabled (all rule actions update the value of $inStar(v)$ and $viableCenter(v)$).

Each node v keeps track of the viable centers within its closed neighborhood $N[v]=N(v)\cup\{v\}$ where $N(v)$ are node at distance 1 of v. Unless v is a member of a star, it invites the viable center having the minimum identifier (c.f. the macro $bestCenter(v)$) to form new p-star by setting $center(v)$ to the viable center's identifier (rule RI). The invitation is updated as needed if the set of viable centers within the closed neighborhood changes (rule RI and RGI).

Directing the invitation at the viable center with the minimum identifier makes sure that no deadlocks or livedeadlocks occur. Eventually, some viable center u is invited by itself and at least p neighbors. Then u picks p neighbors as the leaves of the star and assigns them to the set $leaves(u)$ (rule RA).

To mitigate the potential issue that an invitation is withdrawn concurrently to the execution of rule RA, the shared Boolean variable $lockedCenter(v)$ is used. If $lockedCenter(v)$ is true, then node v cannot change $center(v)$ during the next step.

Before $center(v)$ can be changed by rule RI, rule RGI must be executed to set $lockedCenter(v)$ to false.

Upper bound on the number of moves. The execution of rule RA by a node u creates a new p-star, this p-star is centered at u. Note that at this point, every node v of this p-star (i.e., a node of $star(u)$) will perform at most one move (rule RU) during the end of the execution. So v satisfies $center(v)=u$ forever.

We conclude that $correctCenter(u) \vee correctLeaf(v)$ stays satisfied along the end of the execution. Therefore, each node v changes the value of $isInStar(v)$ at most 2 times.

The variable $inStar(v)$ is updated at most 3 times. So the value of predicate $isViableCenter(v)$ on v changes at most $3|N(v)|+3$ times.

Algorithm 1 : Rules of each node $v \in V$

Shared variables of each node $v \in V$

— $center(v)$ — a node identifier or $\perp$ - The center of the p-star that v belongs to or the viable center that v invites to form new p-star. The value $\perp$ is used if v is not a member of a p-star and is not inviting any node.
— $leaves(v)$ — a set of up to p node identifiers - The set is empty if v is not the center of a pIt finds a maximal decomposition into node-disjoint p-stars.-star. Otherwise it contains the leaves of the p-star.
— $inStar(v) \in Boolean$ - Indicates whether v is a member of a p-star.
— $viableCenter(v) \in Boolean$ - Indicates whether v is a viable center for a new p-star.
— $lockedCenter(v) \in Boolean$ - Indicates whether the value of $center(v)$ is locked or not.

Predicates

— $isCenter(v) \equiv |leaves(v)| = p$
— $incorrectCenter(v) \equiv (leaves(v) \neq \emptyset) \wedge$
$((center(v) \neq v) \vee (\exists u \in leaves(v) : center(u) \neq v) \vee \neg isCenter(v) \vee (leaves(v) \not\subseteq N(v)))$
— $correctLeaf(v) \equiv (center(v) \in N(v)) \wedge isCenter(center(v)) \wedge (v \in leaves(center(v))$
— $correctCenter(v) \equiv isCenter(v) \wedge \neg incorrectCenter(v)$
— $isInStar(v) \equiv correctLeaf(v) \vee correctCenter(v)$
— $isViableCenter(v) \equiv \neg isInStar(v) \wedge (|\{\ u \in N(v) \mid \neg inStar(u)\ \}| \geq p)$

Macros

— $bestCenter(v)$ is the element of $\{u \in N[v] \mid viableCenter(u) \wedge leaves(u) = \emptyset\}$
having the smallest identifier or $\perp$ if the set is empty
— $potentialLeaves(v)$ is the set $\{u \in N(v) \mid center(u) = v \wedge lockedCenter(u)\}$

Guard Predicates

— $starToUpdate(v) \equiv \neg isInStar(v) \wedge (|potentialLeaves(v)| \geq p) \wedge (v = center(v))$
— $centerToUpdate(v) \equiv \neg isInStar(v) \wedge (center(v) \neq bestCenter(v) \vee \neg lockedCenter(v))$
— $variablesToUpdate(v) \equiv (inStar(v) \neq isInStar(v)) \vee (viableCenter(v) \neq isViableCenter(v)) \vee$
$incorrectCenter(v)$

Procedures

— $updateBooleans(v)$: $inStar(v) := isInStar(v)$; $viableCenter(v) := isViableCenter(v)$;
— $updateVariables(v)$: if $incorrectCenter(v)$ then $leaves(v) := \emptyset$; $updateBooleans(v)$;

Rules

RA(v) : $starToUpdate(v) \longrightarrow$
$leaves(v) :=$ subset of $potentialLeaves(v)$ with exactly p elements ; $updateBooleans(v)$;

RI(v) : $\neg starToUpdate(v) \wedge centerToUpdate(v) \wedge \neg lockedCenter(v) \longrightarrow$
$lockedCenter(v) :=$ true ; $center(v) := bestCenter(v)$; $updateVariables(v)$;

RGI(v) : $\neg starToUpdate(v) \wedge centerToUpdate(v) \wedge lockedCenter(v) \longrightarrow$
$lockedCenter(v) :=$ false ; $updateVariables(v)$;

RU(v) : $\neg starToUpdate(v) \wedge \neg centerToUpdate(v) \wedge variablesToUpdate(v) \longrightarrow updateVariables(v)$;

The first execution of rule RU may be triggered by initial inconsistencies. All further executions of RU are caused by changes of $isInStar(v)$ and $isViableCenter(v)$. The latter can occur at most $3|N(u)|+2+2$ times. We conclude that a node v executes the rule RU at most $3|N(v)|+5$ times.

$bestCenter(v)$ changes only if $setBC(v)$ changes. $setBC(v)$ is the set of nodes of $N[v]$ satisfying (i) $viableCenter(u) \wedge leaves(u) = \emptyset$. The value of $leaves(u)$ changes at most two times during any execution. So, the condition (i) on node u changes its value at most $3|N(u)|+3+2 \leq 3\Delta+5$ times on any execution. So, $setBC(v)$ changes at most $nb_{BC}(v) = |N[v]|(3\Delta+5)$ times.

When the algorithm assigns true to $lockedCenter(v)$ (rule RI), then it also assigns $bestCenter(v)$ to $center(v)$. So $center(v) \neq bestCenter(v)$ and $lockedCenter(v) =$ true is either due to initial inconsistencies or due to the fact that $bestCenter(v)$ has changed. Therefore, the rule RGI is executed by node v at most $nb_{BC}(v)+1$ times and the rule RI is executed by node v at most $nb_{BC}(v)+2$ times.

Theorem 1 *The algorithm terminates after at most* $12\Delta m + O(m+n)$ *moves under the unfair distributed scheduler. These moves happen within at most* $5\left\lfloor \frac{n}{p+1} \right\rfloor + 5$ *(asynchronous) rounds.*

Correctness of the algorithm. The algorithm is silent, i.e., it eventually reaches a terminal configuration under any schedule. In a terminal configuration, the guards of all rules evaluate to false for every node. So, in a terminal configuration, if $leaves(v) \neq \emptyset$ then v is the center of a well formed p-star containing the nodes of $star(v)$ (v does not verify $incorrectCenter(v)$). So a node that does not belong to any p-star (i.e., $v \in V - \bigcup_{v \in V} star(v)$) verifies $leaves(v) = \emptyset$ and $inStar(v) = false$. It can be concluded that in a terminal configuration, no viable centers exist and thus the decomposition is maximal.

Memory space. The memore space required by our algorithm on each node is $(p+1)log(n)+3$ bits. Notice this requirement is similar at the memory space requirement by the algorithm of [9] (i.e., $p \cdot log(n)$ bits) and at the memory space requirement by the algorithm of [8] (i.e., $p \cdot log(n)+3$ bits).

Conclusion. We revisited the problem of decomposing a graph into node-disjoint p-stars from a self-stabilization point of view. This problem is a generalization of maximal matching. The proposed algorithm performs better than both previously proposed algorithms. In fact, we improved the move complexity while also solving the uniqueness legitimate configuration problem that [9] suffered from, without losing linearity of round complexity. As future work, we aim to generalize the proposed algorithm to the weighted p-star decomposition problem.

The complet presentation of the algorithm with the proofs of correctness under the unfair scheduler and the computation of the upper bound on the number of rounds and moves can be found in the technical report [3].

Références

[1] Konstantin Andreev and Harald Räcke. Balanced graph partitioning. In *SPAA*, pages 120–124, 2004.

[2] Darryn E. Bryant, Saad I. El-Zanati, and Charles Vanden Eynden. Star factorizations of graph products. *J. Graph Theory*, 36(2) :59–66, 2001.

[3] Mohammed Haddad, Colette Johnen, and Sven Köhler. Polynomial silent self-stabilizing p-star decomposition. Research report, hal-01514323, 2016.

[4] Mohammed Haddad, Colette Johnen, and Sven Köhler. Polynomial silent self-stabilizing p-star decomposition (short paper). In *SSS, LNCS 10083, Springer*, pages 185–189, 2016.

[5] David G. Kirkpatrick and Pavol Hell. On the completeness of a generalized matching problem. In *STOC*, pages 240–245, 1978.

[6] Slimane Lemmouchi, Mohammed Haddad, and Hamamache Kheddouci. Robustness study of emerged communities from exchanges in peer-to-peer networks. *Computer Communications*, 36(10-11) :1145–1158, 2013.

[7] Fredrik Manne, Morten Mjelde, Laurence Pilard, and Sébastien Tixeuil. A new self-stabilizing maximal matching algorithm. *Theoretical Computer Science*, 410(14) :1336–1345, 2009.

[8] Brahim Neggazi, Mohammed Haddad, and Hamamache Kheddouci. A new self-stabilizing algorithm for maximal p-star decomposition of general graphs. *Information Processing Letters*, 115(11) :892–898, 2015.

[9] Brahim Neggazi, Volker Turau, Mohammed Haddad, and Hamamache Kheddouci. A self-stabilizing algorithm for maximal p-star decomposition of general graphs. In *SSS, LNCS 8255, Springer*, pages 74–85, 2013.