Chordality Preserving Incremental Triangular Decomposition and Its Implementation

Changbo Chen[1,2(✉)]

[1] Chongqing Key Laboratory of Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China
chenchangbo@cigit.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, China
http://www.arcnl.org/cchen

Abstract. In this paper, we first prove that the incremental algorithm for computing triangular decompositions proposed by Chen and Moreno Maza in ISSAC' 2011 in its original form preserves chordality, which is an important property on sparsity of variables. On the other hand, we find that the current implementation in Triangularize command of the RegularChains library in Maple may not always respect chordality due to the use of some simplification operations. Experimentation show that modifying these operations, together with some other optimizations, brings significant speedups for some super sparse polynomial systems.

Keywords: Triangular decomposition · Chordal graph · Incremental algorithm · Regular chain

1 Introduction

The method of triangular decomposition pioneered by Ritt [19] and Wu [23] has become a basic tool for computing the solutions of polynomial systems over an algebraically closed field. Given a finite set of polynomials F, this method decomposes F into finitely many systems of triangular shape such that the union of their zero sets is the same as that of F. With such decomposition in hand, many information on the solution set, such as emptiness, dimension, cardinality, etc., can be easily obtained. Triangular decomposition has been studied and gradually improved by many others in both theory [1,2,14,15,25] and algorithms [7,10,12,13,16,17,22,24]. Efficient implementations exist in a Maple built-in package RegularChains as well as many other libraries and softwares, such as Epsilon, Wsolve, Magma, and so on.

Nowadays, triangular decomposition has also become an important back-end engine for several algorithms in studying semi-algebraic sets, such as real root classification [24] and comprehensive triangular decomposition of parametric semi-algebraic sets [5], computing sample points of semi-algebraic sets [4],

A. M. Bigatti et al. (Eds.): ICMS 2020, LNCS 12097, pp. 27–36, 2020.
https://doi.org/10.1007/978-3-030-52200-1_3

describing semi-algebraic sets [4], as well as computing cylindrical algebraic decompositions and performing quantifier elimination [3,8,9]. These algorithms and their implementations make triangular decomposition become an efficient tool in many applications, such as theorem proving, program verification, stability analysis of biological systems, and so on.

To further improve the efficiency of triangular decomposition, one important direction is to explore the structure of input systems, such as symmetry and sparsity. The work of [11] brings the concept of variable sparsity to the world of triangular decomposition by virtue of the chordal graph of polynomial systems. Chordal graph already exists in many other contexts, such as Gauss elimination [20] and semidefinite optimization [21].

It has already been shown that some top-down algorithms [22] (up to minor modification of its original form) preserve chordality [18]. Incremental algorithms [7,16,17] are another important class, which compute triangular decompositions by induction on the number of polynomials. It is a natural question to ask if the incremental algorithms can also preserve chordality. In this paper, we provide an affirmative answer to this question for the incremental algorithm proposed in [6,7]. On the other hand, we find that the current implementation of this algorithm in Triangularize command of the RegularChains library in Maple may not always respect chordality. After a careful examination of the implementation, we point out this is due to the use of some simplification operations. Finally, we show by experimentation that modifying these operations, together with some other optimizations, bring significant speedups for Triangularize on some super sparse polynomial systems.

2 Basic Lemmas

Definition 1 (Graph). *Let* $\mathbf{x} = x_1, \ldots, x_n$ *and* $F \subset \mathbf{k}[\mathbf{x}]$. *The (associated) graph* $\mathfrak{G}(F)$ *of* F *is an undirected graph defined as follows:*

- *The set* V *of vertices of* $\mathfrak{G}(F)$ *is the set of variables appearing in* F.
- *The set* E *of edges of* $\mathfrak{G}(F)$ *is the set of* (x_i, x_j), $i \neq j$, *where* x_i *and* x_j *simultaneously appear in some* $f \in F$.

Denote by $\mathfrak{v}(\mathfrak{G}(F))$ *an operation which returns* V.

Definition 2 (Perfect elimination ordering). *Let* $\mathfrak{G} = (V, E)$ *be a graph with vertices* $V = \{x_1, \ldots, x_n\}$. *An ordering* $\mathbf{x} = x_{i_1} > \cdots > x_{i_n}$ *is a perfect elimination ordering for* $\mathfrak{G}$ *if for any* x_{i_j}, *the induced subgraph on the set of vertices* $V_{i_j} = \{x_{i_j}\} \cup \{x_{i_k} \mid x_{i_k} < x_{i_j} \text{ and } (x_{i_j}, x_{i_k}) \in E\}$ *is a clique. If a perfect elimination ordering* $\mathbf{x}$ *exists for* $\mathfrak{G}$, *we say* $\mathfrak{G}$ *is chordal (w.r.t.* $\mathbf{x}$*). We say that a graph* $\overline{\mathfrak{G}}$ *with vertices* $V = \{x_1, \ldots, x_n\}$ *is a chordal completion of* $\mathfrak{G}$, *if* $\overline{\mathfrak{G}}$ *is chordal and* $\mathfrak{G}$ *is a subgraph of* $\overline{\mathfrak{G}}$.

Example 1. *Let* $F := \{x_1^2 - x_2^2 + x_2x_4, x_2^2 - x_3x_4\}$. *Then* $\mathfrak{G}(F)$ *is illustrated in Fig. 1, where the vertex* x_i *is renamed as* i *for short. The ordering* $x_1 > x_2 > x_3 > x_4$ *is a perfect elimination ordering and* $\mathfrak{G}$ *is chordal w.r.t. this ordering. Another ordering* $x_2 > x_1 > x_3 > x_4$ *is not a perfect elimination ordering.*

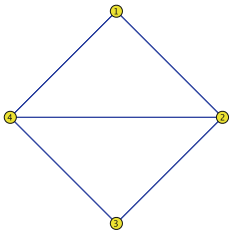

Fig. 1. Chordal graph.

Definition 3. *Let* $\mathbf{x}$ *be a given ordering and* $F \subset \mathbf{k}[\mathbf{x}]$. *Let* $M : 2^{\mathbf{k}[\mathbf{x}]} \to 2^{2^{\mathbf{k}[\mathbf{x}]}}$. *Let* $\overline{\mathfrak{G}(F)}$ *be any chordal completion of* $\mathfrak{G}(F)$ *w.r.t. the ordering* $\mathbf{x}$ *(that is* $\mathbf{x}$ *is a perfect elimination ordering for* $\overline{\mathfrak{G}(F)}$*). We say that* M *preserves chordality in* $\overline{\mathfrak{G}(F)}$ *(resp.* $\overline{\mathfrak{G}(F)}$*) if for any* $S \in M(F)$, *we have* $\mathfrak{G}(S) \subseteq \mathfrak{G}(F)$ *(resp.* $\mathfrak{G}(S) \subseteq \overline{\mathfrak{G}(F)}$*). In the former case, we say that* M *strongly preserves chordality (for* F*). In the latter case, that is* M *preserves chordality in any chordal completion of* $\mathfrak{G}(F)$ *w.r.t.* $\mathbf{x}$, *we say that* M *preserves chordality (for* F*).*

Remark 1. *Let* $O : \mathbf{k}[\mathbf{x}] \to 2^{\mathbf{k}[\mathbf{x}]}$ *be a unary operation which maps* $f \in \mathbf{k}[\mathbf{x}]$ *to* $O(f) \subset \mathbf{k}[\mathbf{x}]$. *Then it induces a map* M *which maps* $\{f\}$ *to* $\{O(f)\}$. *Let* $O : \mathbf{k}[\mathbf{x}] \times \mathbf{k}[\mathbf{x}] \to 2^{\mathbf{k}[\mathbf{x}]}$ *be a binary operation which maps* (f, g) *to* $O(f, g)$. *Then it induces a map* M *which maps* $\{f, g\}$ *to* $\{O(f, g)\}$. *For both cases, if* M *(strongly) preserves chordality, we say that the operation* O *(strongly) preserves chordality.*

Example 2. *Consider* $F := \{{x_1}^3 - 1, {x_2}^3 - 1, {x_3}^3 - 1, {x_4}^3 - 1, {x_1}^2 + x_1x_2 + {x_2}^2, {x_2}^2 + x_2x_3 + {x_3}^2, {x_3}^2 + x_3x_4 + {x_4}^2, {x_1}^2 + x_1x_4 + {x_4}^2\}$. *Figure 1 depicts a chordal completion* $\overline{\mathfrak{G}(F)}$ *of* $\mathfrak{G}(F)$ *w.r.t. the ordering* $x_1 > x_2 > x_3 > x_4$. *Let* M *be the* Triangularize *command in Maple 2019, which computes a set of regular chains as follows:*

$$
\begin{array}{l}
\{\{x_1 + x_4 + 1, x_2 - x_4, x_3 + x_4 + 1, {x_4}^2 + x_4 + 1\}, \\
\{x_1 - 1, x_2 - x_4, x_3 + x_4 + 1, {x_4}^2 + x_4 + 1\}, \\
\{(x_4 - 1)\, x_1 - x_4 - 2, x_2 - 1, x_3 + x_4 + 1, {x_4}^2 + x_4 + 1\}, \\
\{x_1 - 1, {x_2}^2 + x_2 + 1, x_3 - 1, {x_4}^2 + x_4 + 1\}, \\
\{x_1 + x_4 + 1, (x_4 + 2)\, x_2 - x_4 + 1, x_3 - 1, {x_4}^2 + x_4 + 1\}, \\
\{(x_3 + 2)\, x_1 - x_3 + 1, x_2 + x_3 + 1, {x_3}^2 + x_3 + 1, x_4 - 1\}, \\
\{{x_1}^2 + x_1 + 1, x_2 - 1, {x_3}^2 + x_3 + 1, x_4 - 1\}\}.
\end{array}
$$

As we see, it does not preserve chordality since the graph of the fifth regular chain is not a subgraph of $\overline{\mathfrak{G}(F)}$.

Definition 4. *Let* $O : K[\mathbf{x}] \to 2^{K[\mathbf{x}]}$ *be a unary operation. We say that it respects the (elimination) ordering* $\mathbf{x}$ *for* $f \in \mathbf{k}[\mathbf{x}]$ *if* $\mathfrak{v}(O(f)) \subseteq \mathfrak{v}(f)$.

Lemma 1. *If a unary operation* O *respects the ordering, then it strongly preserves chordality.*

Proof. It trivially holds since $\mathfrak{G}(f)$ is a clique for any $f \in \mathbf{k}[\mathbf{x}]$.

Definition 5. *Given $f \in \mathbf{k}[\mathbf{x}]$, let $\mathrm{mvar}(f)$ be the largest variable appearing in f. Let $O : \mathbf{k}[\mathbf{x}] \times \mathbf{k}[\mathbf{x}] \to 2^{\mathbf{k}[\mathbf{x}]}$ be a binary operation. Let $S := O(f, g)$. We say that O respects the ordering $\mathbf{x}$ (for f and g) if the following are satisfied:*

- *If $\mathrm{mvar}(f) = \mathrm{mvar}(g)$, we have $\mathfrak{v}(S) \subseteq \mathfrak{v}(\{f, g\})$.*
- *If $\mathrm{mvar}(f) \neq \mathrm{mvar}(g)$, we have $\mathfrak{v}(S) \subseteq \mathfrak{v}(f)$ or $\mathfrak{v}(S) \subseteq \mathfrak{v}(g)$.*

Lemma 2. *Suppose that O respects the ordering $\mathbf{x}$, then we have the following.*

- *If $\mathrm{mvar}(f) \neq \mathrm{mvar}(g)$, then O strongly preserves chordality.*
- *If $\mathrm{mvar}(f) = \mathrm{mvar}(g)$, then O preserves chordality.*

Proof. Let $\overline{\mathfrak{G}}$ be any chordal completion of $\mathfrak{G}(\{f, g\})$ and x_i be the common main variable of f and g. By the definition of chordal graph, the subgraph of $\overline{\mathfrak{G}}$ induced by the set of vertices $\mathfrak{v}(\overline{\mathfrak{G}}) \setminus \{x_i\}$ is a clique. So the conclusion holds.

Corollary 1. *The irreducible factorization* Factor *strongly preserves chordality. If the two input polynomials have the same main variable, then the subresultant chain* SubRes*, the resultant* res*, the pseudo remainder* prem *and the pseudo quotient* pquo *operations w.r.t. the main variable preserve chordality.*

If the input two polynomials do not have the same main variable, then these operations may destroy chordality, see Example 3.

Example 3. *Consider the system $F := \{f_1, f_2\}$ again from Example 1 and the ordering $x_1 > x_2 > x_3 > x_4$. We have $\mathrm{mvar}(f_1) = x_1$ and $\mathrm{mvar}(f_2) = x_2$. Then $\mathsf{prem}(f_1, f_2, x_2) = {x_1}^2 + x_2x_4 - x_3x_4$. Clearly* prem *does not preserve chordality for f_1 and f_2.*

Lemma 3. *Let $F \subset \mathbb{Q}[\mathbf{x}]$. Let p be a polynomial in F. Assume that F is chordal w.r.t. the variable order $\mathbf{x}$. Let $F' := F \setminus \{p\}$. Then there exists a chordal completion $\overline{G(F')}$ of $G(F')$ (with the same set of vertices) w.r.t. the order $\mathbf{x}$ such that $\overline{G(F')} \subseteq G(F)$.*

Proof. Let $\overline{G(F')} := \{(u, v) \mid (u, v) \in G(F), v < u, u \in G(F') \text{ and } v \in G(F')\}$. For any u in $\mathfrak{v}(F')$, the set $\{(u, v), v < u, v \in \mathfrak{v}(F')\}$ is a also clique since $\{(u, v) \mid (u, v) \in G(F), v < u, v \in \mathfrak{v}(F)\}$ is a clique by the assumption that F is chordal. Thus $\overline{G(F')}$ is a chordal completion of $G(F')$.

3 The Incremental Algorithm Preserves Chordality

In this section, we prove that the incremental algorithm, namely Algorithm 1, proposed in [7] preserves chordality. The main subroutine of Algorithm 1 is the Intersect operation, which takes a polynomial p and a regular chain T as input, and returns a sequence of regular chains $T_1, \ldots, T_r$ such that

$$V(p) \cap W(T) \subseteq \cup_{i=1}^{r} W(T_i) \subseteq V(p) \cap \overline{W(T)}, \tag{1}$$

where $V(p)$ is the variety of p, $W(T)$ is the quasi-component of T and $\overline{W(T)}$ is the Zariski closure of $W(T)$. Due to limited space, we refer the reader to [7] for a precise definition of these concepts and a detailed description of the algorithm Intersect and its subroutines Extend, IntersectAlgebraic, IntersectFree, CleanChain, and RegularGcd.

Algorithm 1: Triangularize(F, R)

```
1 if F = { } then return {∅};
2 Choose a polynomial p ∈ F ;
3 for T ∈ Triangularize(F \ {p}, R) do  output Intersect(p, T, R);
```

Lemma 4. *One can transform Algorithm* Triangularize *into an equivalent one with the original flow graph illustrated by the left subgraph of Fig. 2 replaced by the one depicted in the right subgraph of Fig. 2.*

Proof. The transformation can be done in two steps. Firstly one can easily replace the direct recursions in Extend, Triangularize and IntersectAlgebraic by iterations. Secondly one can make the function calls to Extend, IntersectAlgebraic, IntersectFree, CleanChain, RegularGcd inline. As a consequence, one obtains an equivalent form of Triangularize with the flow graph of function calls depicted in the right subfig of Fig. 2.

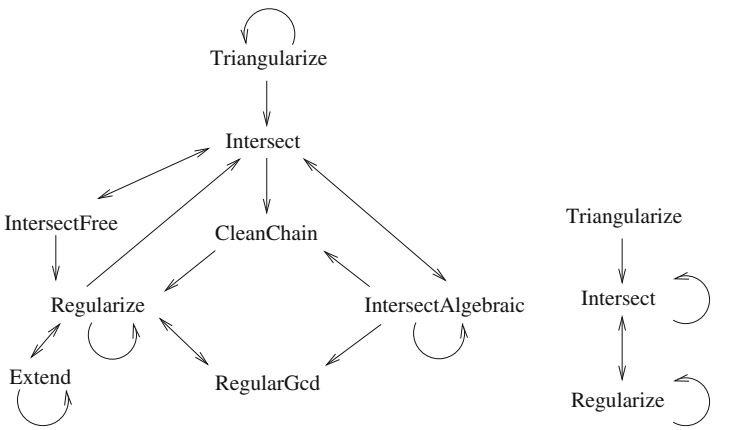

Fig. 2. Transform flow graph of the algorithm.

Lemma 5. *For each algorithm in Fig. 2, any polynomial in it, if appearing in the output of the algorithm or appearing as the output or input of subroutines in Fig. 2 of the algorithm, is obtained through chordality preserving operations. Moreover, if we remove step* 1 *in* Intersect*, which calls* prem *to test if* p *belongs to the saturated ideal of* T *(the algorithm is still correct), all basic operations appearing in the algorithm preserve chordality.*

Lemma 6. *Let* $F \subset \mathbb{Q}[\mathbf{x}]$ *be finite and assume that* F *is chordal. Let* p *be a polynomial and* T *be a regular chain of* $\mathbb{Q}[\mathbf{x}]$ *such that* $\mathfrak{G}(p) \subseteq \mathfrak{G}(F)$ *and* $\mathfrak{G}(T) \subseteq \mathfrak{G}(F)$*. Let* $(p_1, T_1), \ldots, (p_e, T_e)$ *be the processes in the output of* Intersect *or* Regularize*, then we have* $\mathfrak{G}(p_i, T_i) \subseteq \mathfrak{G}(F)$*. (The output* T_i *is treated as a special process* $(0, T_i)$*.) If this is true, we say that* Intersect *and* Regularize *preserves chordality (w.r.t.* F*).*

Proof. We prove this by induction on the rank of the process (p, T).

- *Base:* For each returned process (p_i, T_i), if it is obtained without relying on the output of recursive calls to Intersect or Regularize, then $\mathfrak{G}(\{p_i\} \cup T_i) \subseteq \mathfrak{G}(\{p\} \cup T)$ holds.
- *Induction:* For each recursive calls to Intersect or Regularize, the rank of input process is less than that of (p, T). By the induction hypothesis, the recursive calls preserve chordality. Moreover, by Lemma 5, we notice that the input process for each recursive call is obtained by chordality preserving operations as well as the output process of each recursive call is processed by chordality preserving operations. Thus the lemma holds.

Theorem 1. *Algorithm* Triangularize *preserves chordality.*

Proof. We prove it by induction on the number of elements of F. The base case trivially holds. Let $p \in F$ and $F' := F \setminus \{p\}$. By Lemma 3, there exists a chordal completion $\overline{G(F')}$ of $G(F')$ such that $\overline{G(F')} \subseteq G(F)$. By induction hypothesis, for each regular chain T in the output of the recursive call to Triangularize, we have $G(T) \subseteq \overline{G(F')} \subseteq G(F)$. Then the conclusion follows from Lemma 6.

4 Modifying the Implementation of Triangularize

Algorithm 1 has been implemented in the RegularChains library with the same name. After carefully tracing the code, we find that the only operation in the implementation of Triangularize that may destroy the chordality is the pseudo-reminder operation prem. As illustrated by Example 3 in Sect. 2, if two input polynomials do not have the same main variable, prem can destroy chordality. In particular, the operation prem was empolyed in several places for performing the simplification $q := \mathsf{prem}(p, T)$, where p, q are two polynomials and T is a regular chain. Such simplification does not hurt the correctness of the algorithm, although it may affect the efficiency and should be empolyed with caution. For instance, it was empolyed as a preprocessing step for Intersect but only if the initials of polynomials in T are constants, which may reduce the degree of the

polynomial. Note that we have $V(p) \cap W(T) = V(q) \cap W(T)$, thus the correctness of Intersect is not hurt by Eq. (1). However, if q is involved in producing polynomials as input or output of some algorithms in Fig. 2, then Triangularize may not preserve chordality. Thus, for all these places, we simply do not call prem to preserve chordality. Note that $q = 0$ if and only if p is contained in the saturated ideal of T (membership testing) [6]. If prem is only used for membership testing, we do not suspend the call to prem as it does not affect chordality. There are some other operations, such as iterated resultant, which may not preserve the chordality either. But since they do not produce polynomials as input or output of algorithms in Fig. 2, we keep the calls to them unchanged.

There are several other changes we made to improve the efficiency of the code for chordal input. One is to control the generation of redundant regular chains in Algorithm 1 after each recursive call. Another is to change the order of polynomials in F to solve in Algorithm 1. In the current implementation of Triangularize, one first solves polynomials with smaller rank (in particular with smaller main variables). But this strategy seems to increase the chance of calling operations not preserving chordality. So we instead now first solve polynomials with larger rank. As an example, for lattice-r-10 in Table 1, the two different strategies respectively lead to calling $\mathsf{prem}(p, T)$ 2659 and 964 times.

The implementation preserving chordality is available in Triangularize in the updated RegularChains library (downloadable from http://www.arcnl.org/cchen/software/chordal) through option chordal = true.

5 Experiments

Table 2 compares its performance with Triangularize in Maple 2019. We also include the performance of the regser command of the Epsilon library as a reference. In Table 1, the examples minor-k, lattice-k and coloring-k are chosen from [11]. The examples minor-r-k (resp. lattice-r-k) is a slight modification of minor-k (resp. lattice-k), but have the same associated graph as minor-k (resp. lattice-k).

Table 1. Benchmark examples.

minor-k	$\{x_{2i-1}x_{2i+2} - x_{2i}x_{2i+1} \mid i = 1, \ldots, k\}$
minor-r-k	$\{x_{2i-1}x_{2i+2} - x_{2i}x_{2i+1} + x_{2i} + x_{2i+1} \mid i = 1, \ldots, k\}$
lattice-k	$\{x_i x_{i+3} - x_{i+1}x_{i+2} \mid i = 1, \ldots, k\}$
lattice-r-k	$\{x_i x_{i+3} - x_{i+1}x_{i+2} + x_{i+3}^2 \mid i = 1, \ldots, k\}$
coloring-k	$\{x_i^3 - 1 \mid i = 1, \ldots, k\} \cup \{\sum_{j=0}^{2} x_i^{2-j} x^j_{(i \bmod k)+1} \mid i = 1, \ldots, k\}$

In Table 2, we write Triangularize for short as Tri and Triangularize with chordal = true as Tri-C. Denote by K and L respectively the Kalkbrener and Lazard triangular decomposition. For each system F, let n be the number of

variables, m be the number of polynomials in F, d be the maximum degree of polynomials in F, t be the computation time (in seconds), c be the number of components in the output. The timeout ($-$) is set as one hour. As we can see from the table, for these particular sparse systems, Tri-C significantly outperforms Tri. Meanwhile, Tri-C and regser each have their own favorite examples.

Table 2. Benchmark.

Sys	n	m	d	regser		Tri (K)		Tri (L)		Tri-C (K)		Tri-C (L)	
				Time	c	Time	c	Time	c	Time	c	Time	c
minor-10	22	10	2	2.334	455	27.08	89	467.6	875	6.738	89	24.75	767
minor-15	32	15	2	86.13	8236	1701.9	987	–	–	1147.7	987	–	–
minor-18	38	18	2	1402.1	46810	–	–	–	–	–	–	–	–
minor-20	42	20	2	–	–	–	–	–	–	–	–	–	–
minor-r-10	22	10	2	32.05	2214	10.72	1	227.0	498	1.700	1	11.98	351
minor-r-12	26	12	2	336.7	11667	41.30	1	1859.6	1713	5.366	1	75.51	1081
minor-r-14	30	14	2	–	–	153.4	1	–	–	17.87	1	584.8	3329
minor-r-15	32	15	2	–	–	–	–	–	–	33.23	1	1762.4	5842
lattice-10	13	10	2	0.140	18	7.065	15	7.6	15	0.417	24	0.417	24
lattice-20	23	20	2	1.640	154	–	–	–	–	8.93	187	8.913	187
lattice-30	33	30	2	19.47	1285	–	–	–	–	409.4	1549	406.2	1549
lattice-40	43	40	2	259.6	10733	–	–	–	–	–	–	–	–
lattice-r-10	13	10	2	0.459	13	21.41	1	26.27	13	0.436	1	0.506	13
lattice-r-15	18	15	2	27.19	18	–	–	–	–	2.134	1	2.297	18
lattice-r-18	21	18	2	–	–	–	–	–	–	150.2	1	152.7	21
lattice-r-20	23	20	2	–	–	–	–	–	–	–	–	–	–
coloring-10	10	20	3	6.45	123	13.45	123	9.89	123	4.916	102	4.636	102
coloring-12	12	24	3	56.87	322	92.67	322	56.96	322	23.06	267	22.79	267
coloring-14	14	28	3	986.5	843	667.7	843	380.1	843	128.8	699	130.2	699
coloring-15	15	30	3	–	–	–	–	–	–	315.7	1131	312.7	1131

6 Conclusion

In this paper, we first proved that the incremental algorithm for computing triangular decompositions proposed in [7] preserves chordality. Then we pointed out that some simplification operations used in the implementation may destroy chordality. We resolve this problem by carefully modifying the implementation in Triangularize and the experimentation shows that significant speedups are obtained for some very sparse polynomial systems. Finally, we remark that more extensive experimentations on diverse polynomial systems are needed to decide the best use of these simplifications with the guidance of theory and possibly the help of artificial intelligence rather than simply relying on experience.

Acknowledgments. The authors would like to thank anonymous referees for helpful comments. This research was supported by NSFC (11771421, 11671377, 61572024), CAS "Light of West China" Program, the Key Research Program of Frontier Sciences of CAS (QYZDB-SSW-SYS026), and cstc2018jcyj-yszxX0002 of Chongqing.

References

1. Aubry, P., Lazard, D., Moreno Maza, M.: On the theories of triangular sets. J. Symb. Comput. **28**(1–2), 105–124 (1999)
2. Boulier, F., Lemaire, F., Moreno Maza, M.: Well known theorems on triangular systems and the D5 principle. In: Proceedings of Transgressive Computing 2006, Granada, Spain (2006)
3. Chen, C., Moreno Maza, M.: An incremental algorithm for computing cylindrical algebraic decompositions. In: Computer Mathematics: Proceedings of ASCM 2012, pp. 199–222 (2014)
4. Chen, C., Davenport, J.H., May, J.P., Moreno Maza, M., Xia, B., Xiao, R.: Triangular decomposition of semi-algebraic systems. J. Symb. Comput. **49**, 3–26 (2013)
5. Chen, C., Golubitsky, O., Lemaire, F., Maza, M.M., Pan, W.: Comprehensive triangular decomposition. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2007. LNCS, vol. 4770, pp. 73–101. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75187-8_7
6. Chen, C., Moreno Maza, M.: Algorithms for computing triangular decompositions of polynomial systems. In: Proceedings of ISSAC, pp. 83–90 (2011)
7. Chen, C., Moreno Maza, M.: Algorithms for computing triangular decomposition of polynomial systems. J. Symb. Comput. **47**(6), 610–642 (2012)
8. Chen, C., Moreno Maza, M.: Quantifier elimination by cylindrical algebraic decomposition based on regular chains. J. Symb. Comput. **75**, 74–93 (2016)
9. Chen, C., Moreno Maza, M., Xia, B., Yang, L.: Computing cylindrical algebraic decomposition via triangular decomposition. In: Proceedings of ISSAC 2009, pp. 95–102 (2009)
10. Chen, X.F., Wang, D.K.: The projection of quasi variety and its application on geometric theorem proving and formula deduction. In: Winkler, F. (ed.) ADG 2002. LNCS (LNAI), vol. 2930, pp. 21–30. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24616-9_2
11. Cifuentes, D., Parrilo, P.A.: Chordal networks of polynomial ideals. SIAM J. Appl. Algebra Geom. **1**(1), 73–110 (2017)
12. Dahan, X., Moreno Maza, M., Schost, E., Wu, W., Xie, Y.: Lifting techniques for triangular decompositions. In: Proceedings of ISSC, pp. 108–115 (2005)
13. Gao, X.S., Chou, S.C.: Computations with parametric equations. In: Proceedings of ISSAC, pp. 122–127 (1991)
14. Hubert, E.: Notes on triangular sets and triangulation-decomposition algorithms I: polynomial systems. In: Winkler, F., Langer, U. (eds.) SNSC 2001. LNCS, vol. 2630, pp. 1–39. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45084-X_1
15. Kalkbrener, M.: A generalized euclidean algorithm for computing triangular representations of algebraic varieties. J. Symb. Comput. **15**(2), 143–167 (1993)
16. Lazard, D.: A new method for solving algebraic systems of positive dimension. Discrete Appl. Math. **33**(1–3), 147–160 (1991)
17. Moreno Maza, M.: On triangular decompositions of algebraic varieties. Technical report, TR 4/99, NAG Ltd., Oxford, UK (1999). Presented at MEGA-2000

18. Mou, C., Bai, Y.: On the chordality of polynomial sets in triangular decomposition in top-down style. In: Proceedings of ISSAC, pp. 287–294 (2018)
19. Ritt, J.F.: Differential equations from the algebraic standpoint, vol. 14. American Mathematical Society (1932)
20. Rose, D.J.: Triangulated graphs and the elimination process. J. Math. Anal. Appl. **32**(3), 597–609 (1970)
21. Vandenberghe, L., Andersen, M.S.: Chordal graphs and semidefinite optimization. Found. Trends Optim. **1**(4), 241–433 (2015)
22. Wang, D.: Elimination Methods. Springer, Vienna (2001). https://doi.org/10.1007/978-3-7091-6202-6
23. Wu, W.T.: Basic principles of mechanical theorem proving in elementary geometries. J. Auto. Reasoning **2**(3), 221–252 (1986)
24. Yang, L., Hou, X., Xia, B.: A complete algorithm for automated discovering of a class of inequality-type theorems. Sci. China Seri. F Inf. Sci. **44**(1), 33–49 (2001)
25. Yang, L., Zhang, J.: Searching dependency between algebraic equations: an algorithm applied to automated reasoning. Technical report, International Centre for Theoretical Physics (1990)