Multi-loop System Selection Algorithm

1 Introduction

This algorithm is used to divide a complicated mechanism into several simple subsystems so that it will be much easier to solve. The basic idea is to first convert the system of constraint equations into a graph and then use this graph to find an optimal way to solve loops one by one.

2 Preliminary

An *undirected simple graph*, or a *graph*, is a pair $G = (V, E)$, where V denotes the vertex (or node) set and E denotes the edge set. Each edge $e \in E$ is of the form $e = \{v_1, v_2\}$, where $v_1, v_2 \in V$ and $v_1 \neq v_2$.
Given a graph $G = (V, E)$ and a subset $S \subseteq V$. The *neighbourhood* of S, denoted as $N(S)$, is the set of all vertices who connect to S, or equivalently,

$$N(S) := \{u \in V \mid \exists\ v \in S \text{ such that } \{u, v\} \in E\}.$$

For a vertex $v \in V$, the *degree* of v is defined as the number of elements in $N(\{v\})$. We say a graph $G = (V, E)$ is *bipartite* if V can be written as $V = X \cup Y$ with $X \cap Y = \emptyset$ and for each edge $e \in E$, it connects X and Y. We also says that X and Y are bipartitions of the graph. Therefore, if $S \subseteq X$, then $N(S) \subseteq Y$.
Given a bipartite graph G with one of the bipartitions is X. For any $L_1, \ldots, L_k \subseteq X$, we define $N_{L_1,\cdots,L_k}(A)$ to be the neighborhood of A except the neighborhood of $L_1 \cup \cdots \cup L_k$, that is,

$$N_{L_1,\cdots,L_k}(A) := N(A) \backslash N(L_1 \cup \cdots \cup L_k).$$

3 Create the graph

The first part of this algorithm is to convert equations into a graph. We first input the system of constraint equations and variables, and each equation and variable is a vertex. For every equation P_i in the system, we find all variables (unknowns) in this equation. For each i, we draw edges between equation P_i and the variables in equation P_i.

Here is an example of aircraft landing gear lower loop. The constraint equations are as follow:

$$(D_x - E_x)^2 + (D_y - E_y)^2 - LDE^2 = 0 \tag{1}$$
$$(F_x - E_x)^2 + (F_y - E_y)^2 - LFE^2 = 0 \tag{2}$$
$$F_x^2 + F_y^2 - LAF^2 = 0 \tag{3}$$
$$(D_x - E_x)u_{4x} + (D_y - E_y)u_{4y} = 0 \tag{4}$$
$$F_x u_{5x} + F_y u_{5y} = 0 \tag{5}$$
$$u_{4x}^2 + u_{4y}^2 - 1 = 0 \tag{6}$$
$$u_{5x}^2 + u_{5y}^2 - 1 = 0 \tag{7}$$

The set of parameters is $\{D_x, D_y, F_x, LDE, LFE, LAF\}$. Therefore the set of variables is $\{E_x, E_y, F_y, u_{4x}, u_{4y}, u_{5x}, u_{5y}\}$. The graph we get is as follow:

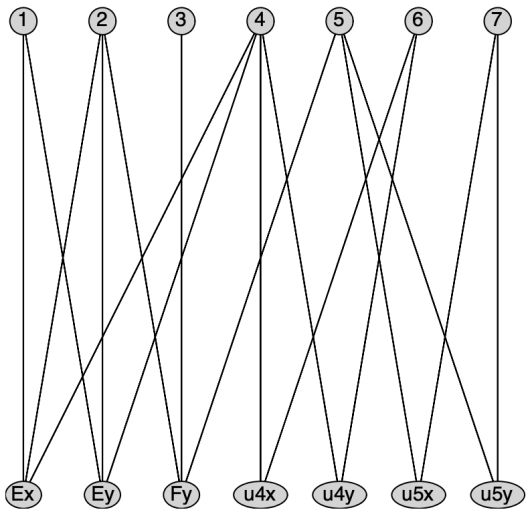

Figure 1: Graph for aircraft landing gear lower loop

Equation (1) is $(D_x - E_x)^2 + (D_y - E_y)^2 - LDE^2 = 0$ and contains variables E_x and E_y, so we draw two edges: from (1) to E_x and from (1) to E_y. Equation (4) is $(D_x - E_x)u_{4x} + (D_y - E_y)u_{4y} = 0$ and it contains four variables E_x, E_y, u_{4x}, u_{4y}, so we draw four edges: from (4) to E_x, E_y, u_{4x}, u_{4y} separately. In this way, we convert a set of variables and equations into a graph. In particular, the graph we constructed is bipartite.

4 Best loop selection

4.1 Sequence of Loops

In this section, we will define valid, complete, and the best sequence of loops respectively. A sequence of loops is a sequence of sets where each set represents a loop. We will first define valid and complete sequence of loops.

Definition 4.1.1. *Given a bipartite graph $G = (V, E)$ with X is one of the bipartition, a sequence of loops $L_1, \cdots, L_n$, where each L_i is a subset of X, is called valid if it satisfies following conditions:*

(i) for all $i, j \in \{1, \cdots, n\}$ where $i \neq j$, $L_i \cap L_j = \emptyset$, i.e. those loops are disjoint;

(ii) for any $i \in \{1, \cdots, k\}$, we have $|L_i| = |N_{L_1, \cdots, L_{i-1}}(L_i)|$.

We also say it is complete w.r.t to a set of constraint equations if it is valid and satisfies the third condition:

(iii) $\bigcup_{i=1}^{n} L_i$ is equal to the set of constraint equations.

The second condition can be considered as after solving those equations from loops $L_1, \cdots, L_{i-1}$, the number of equations in the loop L_i is equal to the number of unknowns. The definition of the best sequence is dependent on an order defined as follow.

Definition 4.1.2. *Suppose we have two complete sequences of loops $A : L_1^{(A)}, \ldots, L_{n_A}^{(A)}$ and $B : L_1^{(B)}, \ldots, L_{n_B}^{(B)}$ of the same graph. Let $c_1^{(A)}, \cdots, c_{n_A}^{(A)}$ and $c_1^{(B)}, \cdots, c_{n_B}^{(B)}$ be the size of each loop in A and B in decreasing order respectively, then the order $\prec$ of the sequences is defined by*

$$A \prec B \text{ iff } c_i^{(A)} < c_i^{(B)} \text{ for the first } i \text{ where } c_i^{(A)} \text{ and } c_i^{(B)} \text{ differs.}$$

We also write $A \succ B$ if $B \prec A$, and write $A \sim B$ if neither $A \succ B$ nor $B \prec A$ is true.

The smaller (w.r.t. $\prec$) a sequence of loop is, the better it is. So when we say that we want to find the *best* sequence of loops, it means that we want to find the *smallest* sequence of loop with respect to $\prec$. The principle behind this criterion is related to the complexity of F4 algorithm. The complexity of F4 algorithm is $\mathcal{O}(d^{3n})$ where d is the maximal degree of equations and n is the number of equations. In our system of constraint equations, d is usually 2 because of the fact that constraint equations derived by natural coordinates are basically length constraints and dot product. Therefore, the complexity only depends on n. Given a sequence of loop, if the maximal loop of it has less variables, then the maximal n will be smaller, which means the computational time for F4 algorithm will be reduced. In the following examples, suppose now we have a mechanical system where we model this system by 13 variables, and there are three complete sequence of loops A, B and C.

Example 4.1.3. Suppose $A = L_1^{(A)}, L_2^{(A)}, L_3^{(A)}$ with size $7, 1, 3$, respectively, and $B = L_1^{(B)}, L_2^{(B)}, L_3^{(B)}, L_4^{(B)}$ with size $2, 5, 1, 3$, respectively. Since the largest size in A is larger than the largest size of B, we get $B \prec A$, which means the B is better than A.

Example 4.1.4. Suppose $B = L_1^{(B)}, L_2^{(B)}, L_3^{(B)}, L_4^{(B)}$ with size $2, 5, 1, 3$, respectively, and $C = L_1^{(C)}, L_2^{(C)}, L_3^{(C)}, L_4^{(C)}$ with size $2, 5, 2, 2$. The first largest size of them is the same, so we compare the second-largest size, which gives us $C \prec B$. It means that C is better than B.

4.2 How Best loop selection works

This algorithm is aimed at choosing the best complete sequence of loops from a multi-loop mechanical system. Based on bipartite graph we get from the first step, the graph has two bipartitions: variables part and equations part. Note that in our algorithm, we assume the number of variables is equal to the number of constraint equations n, which means the system has finitely many solutions if all equations are independent. The key idea of this algorithm is the function `mSetkUnion`. Given a sequence of sets and two non-negative integers m, k, the function `mSetkUnion` will find m sets in that sequence such that their union has at most k elements. We will apply the idea of `mSetkUnion` to a sequence called `sys_indets`. `sys_indets` is a sequence of sets where each set contains all variables appearing in each constraint equation. Therefore, the length of `sys_indets` is equal to the number of constraint equations.

For i from 1 to n, we want to find i sets in the sequence `sys_indets` such that the union of these i sets has i elements by the function `mSetkUnion`. This means that we have i equations and these equations contain exactly i variables. These form a valid loop where we can solve those i equations and i variables and get finitely many solutions. If we succeed in finding such a loop, we can delete those equations and variables from our original system to form a new system and iterate the algorithm on our new system again. We continue this process until we find all possible loops in the given mechanical system.

4.3 Example

We will use aircraft landing gear lower loop and figure 1 as our example again. In this example,

$$\texttt{sys\_indets} = [\{E_x, E_y\}, \{E_x, E_y, F_y\}, \{F_y\}, \{E_x, E_y, u_{4x}, u_{4y}\}, \{F_y, u_{5x}, u_{5y}\}, \{u_{4x}, u_{4y}\}, \{u_{5x}, u_{5y}\}]$$

$$\texttt{var\_appear} = [\{(1), (2), (4)\}, \{(1), (2), (4)\}, \{(2), (3), (4)\}, \{(4), (6)\}, \{(4), (6)\}, \{(5), (7)\}, \{(5), (7)\}]$$

For $i = 1$, from `sys_indets` we can see that equation (3) only has one variable F_y. Therefore, we will choose $[\{F_x^2 + F_y^2 - LAF^2\}, \{F_y\}]$ (note that the first set represents equations in this loop where we omit the RHS and the second set represents variables in this loop) as our first loop to solve. After choosing the first loop, we delete equation (3) and variable F_y from our original system and our new system gives us

$$\texttt{sys\_indets} = [\{E_x, E_y\}, \{E_x, E_y\}, \{E_x, E_y, u_{4x}, u_{4y}\}, \{u_{5x}, u_{5y}\}, \{u_{4x}, u_{4y}\}, \{u_{5x}, u_{5y}\}]$$

$$\texttt{var\_appear} = [\{(1), (2), (4)\}, \{(1), (2), (4)\}, \{(4), (6)\}, \{(4), (6)\}, \{(5), (7)\}, \{(5), (7)\}]$$

Now we iterate our algorithm on this new system again. When $i = 1$, we can't find any loops. We proceed to $i = 2$. When $i = 2$, from `sys_indets` we can see that the union of first two sets contains exactly two elements, which means equations (1) and (2) contain variables E_x and E_y. Therefore, we can use equations (1) and (2) to solve E_x and E_y easily, which gives us the second loop $[\{(D_x - E_x)^2 + (D_y - E_y)^2 - LDE^2, (F_x - E_x)^2 + (F_y - E_y)^2 - LFE^2)\}, \{E_x, E_y\}]$ to solve. Then we delete equation (1), (2) and variables E_x, E_y from our current system and the new system gives us

$$\texttt{sys\_indets} = [\{u_{4x}, u_{4y}\}, \{u_{5x}, u_{5y}\}, \{u_{4x}, u_{4y}\}, \{u_{5x}, u_{5y}\}]$$

$$\texttt{var\_appear} = [\{(4),(6)\},\{(4),(6)\},\{(5),(7)\},\{(5),(7)\}]$$

Now if we continue this process, we will find the next two loops are

$$[\{(D_x - E_x)u_{4x} + (D_y - E_y)u_{4y}, u_{4x}^2 + u_{4y}^2 - 1\}, \{u_{4x}, u_{4y}\}]$$

$$[\{F_x u_{5x} + F_y u_{5y}, u_{5x}^2 + u_{5y}^2 - 1\}, \{u_5 x, u_5 y\}]$$

The result we got after running the algorithm is as follow.

$$1,\ [\ \{Fx^2 + Fy^2 - LAF^2\},\ \{Fy\}\ ]$$
$$2,\ [\ \{(Dx - Ex)^2 + (Dy - Ey)^2 - LDE^2,\ (Fx - Ex)^2 + (Fy - Ey)^2 - LFE^2\},\ \{Ex, Ey\}\ ]$$
$$3,\ [\ \{(Dx - Ex)\ u4x + (Dy - Ey)\ u4y,\ u4x^2 + u4y^2 - 1\},\ \{u4x, u4y\}\ ]$$
$$4,\ [\ \{Fx\ u5x + Fy\ u5y,\ u5x^2 + u5y^2 - 1\},\ \{u5x, u5y\}\ ]$$

Figure 2: Subsystems of aircraft landing gear lower loop

4.4 Correctness of the algorithm

In this section, we will prove the correctness of the algorithm. The idea of the proof is to prove by contradiction. We assume we get sequence A from our algorithm and it's not the best sequence. We further assume there exists another sequence B which satisfies two conditions:

(i) B is a valid sequence;

(ii) B is the best sequence.

We will prove such B doesn't exist by two cases. The first case will contradict with assumption (ii). In other words, case one will tell us either $B \sim A$ or B is not the best. The second case will contradict with assumption (i), which means the second case tells us B is not even a valid sequence. In this proof, we will use an important condition called Hall's condition.

Definition 4.4.1 (Hall's condition)**.** *Given a bipartite graph G with X is one of the bipartitions. We say G satisfies the Hall's condition if for any $A \subseteq X$, $|A| \leq |N(A)|$ holds.*

In our case, X is the set of all constraint equations and the neighbourhood of any subsets is a set of variables. No matter which subset of constraint equations we choose,

the number of equations in this subset is always less than or equal to the neighbourhood of this subset, i.e. the number of equations is always less than or equal to the number of variables that those equations contain. If the number of equations is larger than the number of variables, then this system is actually over determined, which is impossible in our case. Therefore, the bipartite graph we get from the first step always satisfies Hall's condition. Moreover, a loop is valid if and only if the cardinality of the subset of constraint equations is equal to the cardinality of its neighbourhood. Before proving the correctness of the algorithm, we shall prove the following lemma first.

Lemma 4.4.2. *Given a bipartite graph G that satisfies the Hall's condition, let X be one of the bipartitions. If k disjoint subsets $L_1, \cdots, L_k \subseteq X$ satisfies $|L_i| = |N_{L_1,\cdots,L_{i-1}}(L_i)|$ for all $i = 1, \cdots, k$., then for any $S \subseteq X \backslash (\bigcup_{i=1}^{k} L_i)$, we have $|S| \leq |N_{L_1,\cdots,L_k}(S)|$.*

Proof. Suppose a graph G satisfies the Hall's condition and $L_1, \cdots, L_k$ are in one of the bipartitions X where $|L_i| = |N_{L_1,\cdots,L_{i-1}}(L_i)|$. Given $S \subseteq X \backslash (\bigcup_{i=1}^{k} L_i)$, we have

$$
\begin{aligned}
& |S| + \sum_{i=1}^{k} |N_{L_1,\cdots,L_{i-1}}(L_i)| \\
&= |S| + \sum_{i=1}^{k} |L_i| \\
&= |L_1 \cup \cdots \cup L_k \cup S| && \text{since they are disjoint sets.} \\
&\leq |N(L_1 \cup \cdots \cup L_k \cup S)| && \text{by Hall 's condition} \\
&= |N_{L_1,\cdots,L_k}(S)| + |N(L_1 \cup \cdots \cup L_k)| \\
&= |N_{L_1,\cdots,L_k}(S)| + \sum_{i=1}^{k} |N_{L_1,\cdots,L_{i-1}}(L_i)|
\end{aligned}
$$

By cancelling $\sum_{i=1}^{k} |N_{L_1,\cdots,L_{i-1}}(L_i)|$ from the first line and the last line, we can get our desired result. □

Theorem 4.4.3. *The algorithm `ChoosingMultiLoop` will always give us the best multiloop selection with respect to $\prec$.*

Proof. Let $A : L_1^{(A)}, \cdots, L_{n_A}^{(A)}$ be the sequence of loops we get from the algorithm where each $L_i^{(A)}$ represents a loop. It is indeed a *complete* sequence of loop by our algorithm. Suppose A is not the best sequence of loop. Then there exists another complete best (with respect to $\prec$) sequence of loops $B : L_1^{(B)}, \cdots, L_{n_B}^{(B)}$. We reorder the sequence B

such that $L_i^{(A)} = L_i^{(B)}$ for $i = 1, \cdots, k$. In other words, we reorder the sequence B such that the first k loops in sequence B are same as the first k loops in sequence A. The reason why we can reorder the sequence B is that we've already known sequence A is a valid sequence so that we can solve the first k loops in sequence A one by one. If there are k loops in sequence B that are exactly same as the first k loops in sequence A, then we can also follow the order of loops in sequence A to solve those k loops in sequence B. Note that $0 \leq k < \min(n_A, n_B)$.

Now what we want to show is that there doesn't exist an α where $n_B \geq \alpha > k$ such that $L_\alpha^{(B)} = L_{k+1}^{(A)}$.

Suppose equations in the loop $L_{k+1}^{(A)}$ are $P_1, \cdots, P_m$. Let α be the smallest number in $\{k+1, \ldots, t\}$ such that $L_\alpha^{(B)} \cap L_{k+1}^{(A)} \neq \emptyset$. We will prove our statement in two cases:
Case 1. $L_\alpha^{(B)} \supseteq L_{k+1}^{(A)}$. Then there are two sub cases:

- **a.** There is no other polynomial in $L_\alpha^{(B)}$, i.e. $L_\alpha^{(B)} = L_{k+1}^{(A)} = \{P_1, \cdots, P_m\}$.
- **b.** There exists polynomial other than $P_1, \cdots, P_m$ in $L_\alpha^{(B)}$, i.e. $L_\alpha^{(B)} \supsetneq L_{k+1}^{(A)} = \{P_1, \cdots, P_m\}$.

Case 2. $L_\alpha^{(B)} \not\supseteq L_{k+1}^{(A)}$. Since $L_\alpha^{(B)} \cap L_{k+1}^{(A)}$ is non-empty, we can reorder $P_1, \cdots, P_m$ such that $L_\alpha^{(B)} \cap L_{k+1}^{(A)} = \{P_1, \cdots, P_w\}$, where $1 \leq w < m$.

For the **case 1a**, if there is no other polynomial in $L_\alpha^{(B)}$, then $L_\alpha^{(B)}$ is exactly same as $L_{k+1}^{(A)}$ which contradicts with our assumption that only the first k loops in sequence A and B are the same. Therefore, **case 1a** doesn't exist.

For the **case 1b**, if there are any other polynomials in $L_\alpha^{(B)}$, i.e. $L_\alpha^{(B)} = \{P_1, \cdots, P_m, Q_1, \cdots, Q_q\}$, then we can find a smaller valid loop $\{P_1, \cdots, P_m\}$ from $L_\alpha^{(B)}$ since we know that $L_{k+1}^{(A)} = \{P_1, \cdots, P_m\}$ is a valid loop from sequence A. However, this contradicts with the property of B being the best sequence of loops. Therefore, **case 1b** doesn't exist.

For the **case 2**, we first claim that $|N_{L_1^{(A)}, \cdots, L_k^{(A)}}(\{P_1, \cdots, P_w\})| = |N_{L_1^{(A)}, \cdots, L_k^{(A)}, \tilde{L}^{(B)}}(\{P_1, \cdots, P_w\})|$, where $\tilde{L}^{(B)}$ is defined as $L_{k+1}^{(B)} \cup \cdots \cup L_{\alpha-1}^{(B)}$. To show the claim, it suffices to show $N_{L_1^{(A)}, \cdots, L_k^{(A)}}(\tilde{L}^{(B)})$ and $N_{L_1^{(A)}, \cdots, L_k^{(A)}}(L_{k+1}^{(A)})$ are disjoint. We will prove this claim by contradiction. Suppose $N_{L_1^{(A)}, \cdots, L_k^{(A)}}(\tilde{L}^{(B)}) \cap N_{L_1^{(A)}, \cdots, L_k^{(A)}}(L_{k+1}^{(A)}) \neq \emptyset$.

Since both A and B consist of valid loops, we know

$$|\tilde{L}^{(B)}| = |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)})|, |L_{k+1}^{(A)}| = |N_{L_1^{(A)},\cdots,L_k^{(A)}}(L_{k+1}^{(A)})|$$

and $L_{k+1}^{(B)}, \cdots, L_{\alpha-1}^{(B)}, L_{k+1}^{(A)}$ are all disjoint. If the $N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)})$ does contain any elements in $N_{L_1^{(A)},\cdots,L_k^{(A)}}(L_{k+1}^{(A)})$, that is,

$$|N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)}) \cap N_{L_1^{(A)},\cdots,L_k^{(A)}}(L_{k+1}^{(A)})| > 0$$

then

$$\begin{aligned}
|\tilde{L}^{(B)} \cup L_{k+1}^{(A)}| &= |\tilde{L}^{(B)}| + |L_{k+1}^{(A)}| \\
&= |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)})| + |N_{L_1^{(A)},\cdots,L_k^{(A)}}(L_{k+1}^{(A)})| \\
&= |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)}) \cup N_{L_1^{(A)},\cdots,L_k^{(A)}}(L_{k+1}^{(A)})| + |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)}) \cap N_{L_1^{(A)},\cdots,L_k^{(A)}}(L_{k+1}^{(A)})| \\
&> |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)}) \cup N_{L_1^{(A)},\cdots,L_k^{(A)}}(L_{k+1}^{(A)})| \\
&= |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)} \cup L_{k+1}^{(A)})|
\end{aligned}$$

i.e. $|\tilde{L}^{(B)} \cup L_{k+1}^{(A)}| > |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\tilde{L}^{(B)} \cup L_{k+1}^{(A)})|$, which contradicts to lemma 4.4.2. Therefore, our claim holds.

By the process of the algorithm and $w < m$, we get $w < |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\{P_1, \cdots, P_w\})|$. If $w = |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\{P_1, \cdots, P_w\})|$, then $L_{k+1}^{(A)}$ will be a subset of $\{P_1, \cdots, P_w\}$, which is not true, and if $w > |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\{P_1, \cdots, P_w\})|$ then it will contradict with the lemma 4.4.2. Thus we have

$$\begin{aligned}
w &< |N_{L_1^{(A)},\cdots,L_k^{(A)}}(\{P_1, \cdots, P_w\})| \\
&= |N_{L_1^{(A)},\cdots,L_k^{(A)},\tilde{L}^{(B)}}(\{P_1, \cdots, P_w\})| \qquad \text{by our claim} \\
&= |N_{L_1^{(B)},\cdots,L_{\alpha-1}^{(B)}}(\{P_1, \cdots, P_w\})| \qquad \text{since } L_1^{(A)} \cup \cdots \cup L_k^{(A)} \cup \tilde{L}^{(B)} = L_1^{(B)} \cup \cdots \cup L_{\alpha-1}^{(B)} \\
&= \tilde{w}
\end{aligned}$$

However, we also have

$$\begin{aligned}
|\{P_{w+1}, \cdots, P_m\}| &= m - w \\
&> m - \tilde{w} \\
&= |N_{L_1,\cdots,L_k}(\{P_1, \cdots, P_m\})| - |N_{L_1^{(B)},\cdots,L_{\alpha-1}^{(B)}}(\{P_1, \cdots, P_w\})| \\
&\geq |N_{L_1^{(B)},\cdots,L_{\alpha-1}^{(B)}}(\{P_1, \cdots, P_m\})| - |N_{L_1^{(B)},\cdots,L_{\alpha-1}^{(B)}}(\{P_1, \cdots, P_w\})| \\
&= |N_{L_1^{(B)},\cdots,L_{\alpha-1}^{(B)},\{P_1,\cdots P_w\}}(\{P_{w+1}, \cdots, P_m\})| \\
&\geq |N_{L_1^{(B)},\cdots,L_{\alpha-1}^{(B)},L_\alpha^{(B)}}(\{P_{w+1}, \cdots, P_m\})|,
\end{aligned}$$

which contradicts to the lemma 4.4.2. Hence B doesn't exist. This shows that our algorithm indeed gives us the best loop.

□

4.5 Pseudocodes of the algorithm

Algorithm 1 ChoosingMultiLoop

Input:

sys: A list of polynomials,
var: A list of variables.

Output:

A list of loops that will be solved in turn ▷ We use list to represent the sequence. Besides storing the equations in a loop, we also store the variables at the same time.

```
1: function ChoosingMultiLoop(sys,var)
2:     sys_indets ← A list, the ith element is the set of variables appearing in the ith equation
3:     choose_idx ← ChoosingSmallestLoop(sys_indets)
4:     equ_solved_this_loop ← subsequence of sys with index choose_idx
5:     var_solved_this_loop ← all variables appears in equ_solved_this_loop
6:     loop ← [equ_solved_in_this_loop, var_solved_in_this_loop]
7:     new_sys ← sys \ equ_solved_in_this_loop
8:     new_var ← var \ var_solved_in_this_loop
9:     if new_sys = [] then
10:        return [loop]
11:    else
12:        return [loop,op(ChoosingMultiLoop(new_sys,new_var))]
13:    end if
14: end function
```

Algorithm 2 ChoosingSmallestLoop

Input:

n_set: a sequence of sets

Output:

The smallest loop in current system

```
1: function ChoosingSmallestLoop(n_sets)
2:     for i = 1 to n do
3:         get_Loop ← mSetkUnion(n_sets,i,i)
4:         if get_Loop is not false then
5:             return [get_Loop]
6:         end if
7:     end for
8: end function
```

Algorithm 3 mSetkUnion

Input:

n_set: a sequence of sets

m, k: non-negative integers

Output:

The set of m elements in n_sets such that the union of those m sets has at most k elements. (In our practical use, it must be equal to k by our lemma.) If such m elements don't exist, then this function will return $false$.

```
1: function mSetkUnion(n_sets,m,k)
2:     n ← length(n_sets)
3:     if m > n then
4:         return false
5:     end if
6:     candidate ← the index of n_sets that contains at most k elements.
7:     if m = 1 and length(candidate) > 0 then
8:         return {candidate[1]}
9:     end if
10:    for idx in candidate do
11:        new_indets ← n_sets[idx]
12:        remaining_idx ← the remaining indexes that are larger than idx
13:        remaining ← the sets in n_set with index is in remaining_idx
14:        chosen_idx ← mSetskUnion(remaining,m-1,k-length(new_indets))
15:        if chosen_idx is not false then
16:            chosen ← the sets in n_set with index in chosen_idx
17:            return chosen ∪{idx}
18:        end if
19:    end for
20: end function
```