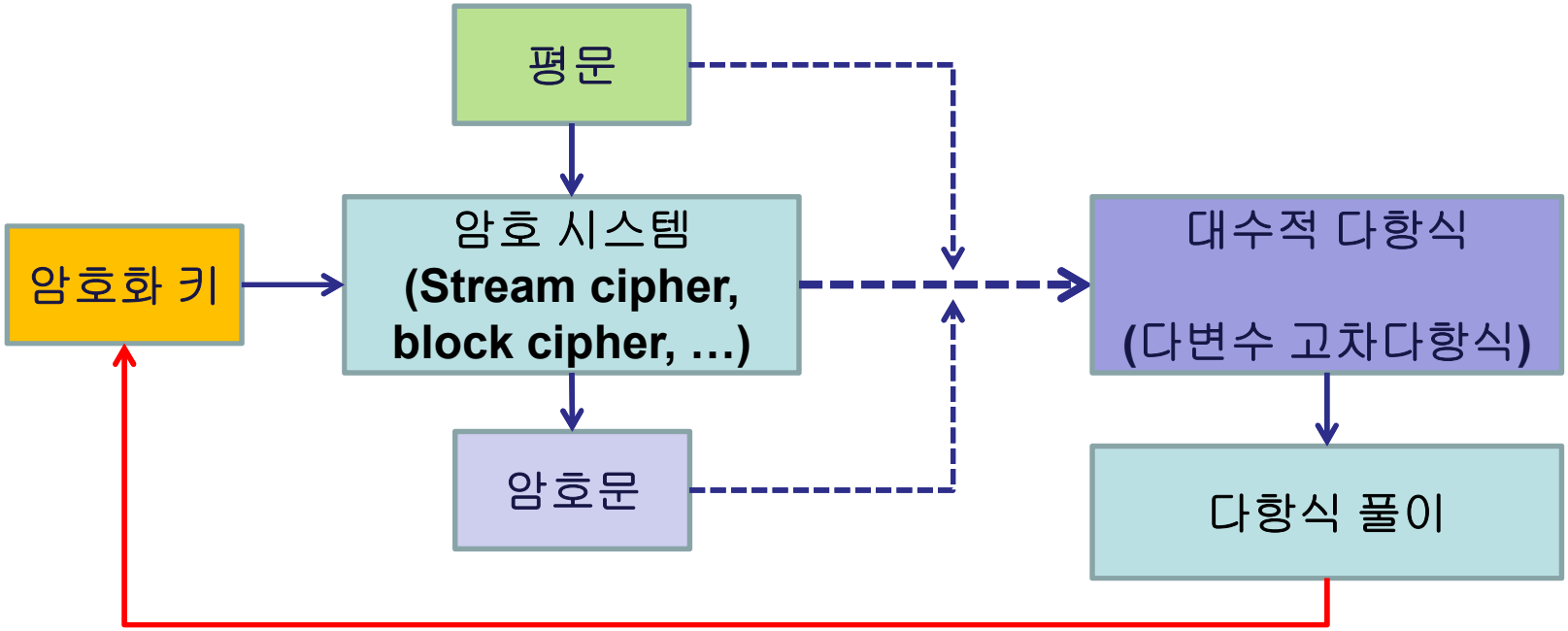
대수적 공격을 위한,
추가다항식을 통한 **GF(2)** 상의 다변수
다항식 계 풀이 속도의 개선

2011 년 12 월 3 일

연세대학교 Coding & Cryptography Lab.

박정열*, 박기현, 박진수, 송홍엽

대수적 공격과 다항식의 풀이



- 다항식의 풀이
  - Groebner 기저 계산 – F4, F5, …
  - 유사 Groebner 기저 계산 – XL, XSL, …
  - 기타 - SAT-solver, …

주 아이디어

- **F4, F5 등은 완성도가 높은 알고리즘**
  - 알고리즘의 개선은 큰 성과를 얻지 못함

- **F4, F5의 동작 환경**
  - 환경 – 유한체 F_q
  - 입력 – F_q 위의 n개의 변수를 가진 다항식 $f_1, f_2, \dots, f_m$
  - 출력 – $f_1, f_2, \dots, f_m$의 Groebner 기저 F

- **F4, F5를 빠르게 고칠 수 없다면, 빠르게 돌아갈 환경을 제공**
  - 입력 다항식의 개수가 많아질 수록 복잡도가 내려감
    - ✓ 추가되는 입력 다항식은
      - Nontrivial
      - Consistent with a system
    - ✓ 어떻게 얻을 것인가?

추가다항식 ↑, 정규도 ↓, F5 수행시간 ↓

- $I = \langle f_1, f_2, \ldots, f_m \rangle \subset k[x_1, \ldots, x_n]$
- I의 정규도 d_{reg}
  - I의 d차 다항식의 최고차항이 d차 단항 전체가 되는 d의 최소값
  - 다항식 개수가 많을수록, 차수가 낮을수록 감소
- 정규도와 **F5**의 수행시간
  - F5 알고리즘 수행 중 등장하는 S-다항식의 최대차수 : d_{max}
  - 준정규 수열일 경우 $d_{max} = d_{reg}$
  - $O\left(\binom{n}{d_{max}}^{\omega}\right) = O\left(\binom{n}{d_{reg}}^{\omega}\right)$ (ω는 2~3 사이의 값)
- 즉 추가다항식이 늘어날수록
  - 정규도가 작아짐 → F5의 수행속도가 빨라짐

준정규 수열, F5 → F4

- 준정규(**semi-regular**) 수열
  - 개념 : $f_1, \dots, f_m$이 d_{reg} 차 미만까지는 서로 관계가 없음
- 무작위로 생성된 다항식들은 거의 다 준정규 수열
  - F5 알고리즘을 수행하기 위한 기본 조건
- $\boldsymbol{R_n} = {}^{\mathbb{F}_2[x_1,\dots,x_n]}\!/_{\langle x_1^2+x_1,\dots,x_n^2+x_n\rangle}$ 상에서는…
  - Strong Nullstellensatz
    - ✓ $f_1, \dots, f_m, f_1^+, \dots, f_k^+$는 준정규 수열이 아님
  - F5 알고리즘 수행 불가
  - F4 알고리즘 수행 가능
- 따라서 **F5**가 아닌 **F4** 상의 복잡도를 생각해야 함

추가다항식 ↑, d_{max} ↓, F4 수행시간 ↓

- **F4**에서는,
  - S-다항식의 최대 차수를 알면,
    F4의 복잡도도 F5와 복잡도와 비슷하게 구해진다고 알려짐

$$\approx O\left(\binom{n}{d_{max}}^{\omega}\right)$$

  - S-다항식의 최대차수를 d_{max}와 d_{max}^{+}라 하면,

$$d_{max} \geq d_{max}^{+}$$

  - 따라서 F4도 추가다항식으로 인해 빨라질 수 있음
    (단, 복잡도 계산이 힘듬)

- 따라서 추가다항식이 **F4** 알고리즘에도 의미가 있음

추가다항식의 생성

- 주어진 n 비트 벡터 $v = (v_1, v_2, \ldots, v_n)$에 대해,

$$F(v)(x_1, x_2, \ldots, x_n) = (x_1 + \overline{v_1})(x_2 + \overline{v_2}) \cdots (x_n + \overline{v_n})$$

- $F(v)$의 성질
  - $F(v)(x) = 0 \Leftrightarrow x \neq v$
  - $f = \sum_{v \notin V(f)} F(v) = \sum_{v \in V(1+f)} F(v)$

- $v \notin V(f_1, \ldots, f_m)$인 v에 대해
  $F(v)$를 적당히 더하면 $f_1, \ldots, f_m$의 추가다항식을 만들 수 있음

- d차 추가다항식 : $O(2^{n-d})$ 시간 필요

추가다항식 생성 알고리즘 1

- $F(v)$를 이용한 추가다항식의 생성 알고리즘 1
  - 주어진 최고차항을 가지는 추가다항식을 생성
  - 복잡도는 대략 $O(2^{n-d})$

	알고리즘 1. Generate_f_LM
입력	T : 항 $f_1,...,f_m$: 다항식 계를 이루는 다항식들
출력	f : $LT(f)=T$, $\deg(f-LT(f))<\deg T$ 인 추가다항식
1	$I_T \leftarrow \{i \in \{1,...,n\} ; x_i \mid T\}$
2	$v \leftarrow_R \mathbb{Z}_2^n \setminus V(f_1, ...,f_m)$
3	만일 $E_{I_T}(v) \cap V(f_1, ...,f_m) \neq \phi$이면 2번으로 돌아감
4	$f \leftarrow \sum_{u \in E_{I_T}(v)} F(v)$
5	f를 반납

추가다항식 생성 알고리즘 2

- $F(v)$를 이용한 추가다항식의 생성 알고리즘 2
  - 주어진 차수를 가지는 추가다항식을 생성
  - 복잡도는 대략 $O(2^{2n-d})$

	알고리즘 2. Generate_f
입력	d : 다항식의 차수 $f_1,...,f_m$: 다항식 계를 이루는 다항식들
출력	f : 차수 d인 추가다항식
1	$v \leftarrow_R \mathbb{F}_2^n \setminus V(f_1, ...,f_m)$
2	$f \leftarrow F(v)$
3	$\deg f > d$인 동안 다음을 반복한다.
4	$g \leftarrow$ Generate_f_LM($LT(f)$, $\{f_1,...,f_m\}$)
5	$f \leftarrow f+g$
6	f를 반납한다.

추가다항식 생성 알고리즘 3

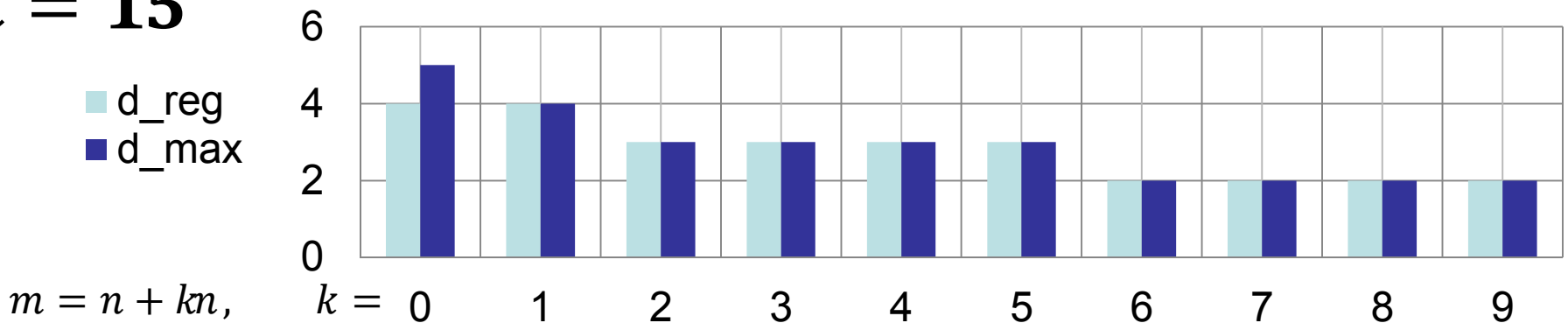
- $F(v)$를 이용한 추가다항식의 생성 알고리즘 3
  - 주어진 차수의 추가다항식을 여러 개 생성
  - 알고리즘 1, 2를 이용하며 얻어낸 다항식들을 최대한 재사용함
  - 복잡도는 대략 $O(k2^{2n-d})$

	알고리즘 3. Generate_several_f
입력	d : 다항식의 차수 k : 추가다항식의 개수 $f_1,...,f_m$: 다항식 계를 이루는 다항식들
출력	$f_1^+,...,f_k^+$: 차수 d인 k개의 추가다항식
1	$S \leftarrow \{\}$
2	$i=1,...,k$에 대해
3	$v \leftarrow_R \mathbb{F}_2^n \setminus V(f_1,...,f_m)$
4	$f_i^+ \leftarrow F(v)$
5	$\deg f_i^+ > d$인 동안 다음을 반복한다.
6	만일 어떤 j에 대해 $\overline{f_i^+}^{f_j} \neq f_i^+$이면 $f_i^+ \leftarrow \overline{f_i^+}^{f_j}$
7	아니면
8	$t = LT(f_i^+)$인 순서쌍 $(t,g_t) \in S$이 존재하면
9	$f_i^+ \leftarrow f_i^+ + g_t$
10	아니면
11	$g \leftarrow$ Generate_f$(LT(f_i^+), \{f_1,...,f_m\})$
12	$S \leftarrow S \cup \{(LT(f_i^+), g)\}$
13	$f_i^+ \leftarrow f_i^+ + g$
14	f와 E를 반납한다.

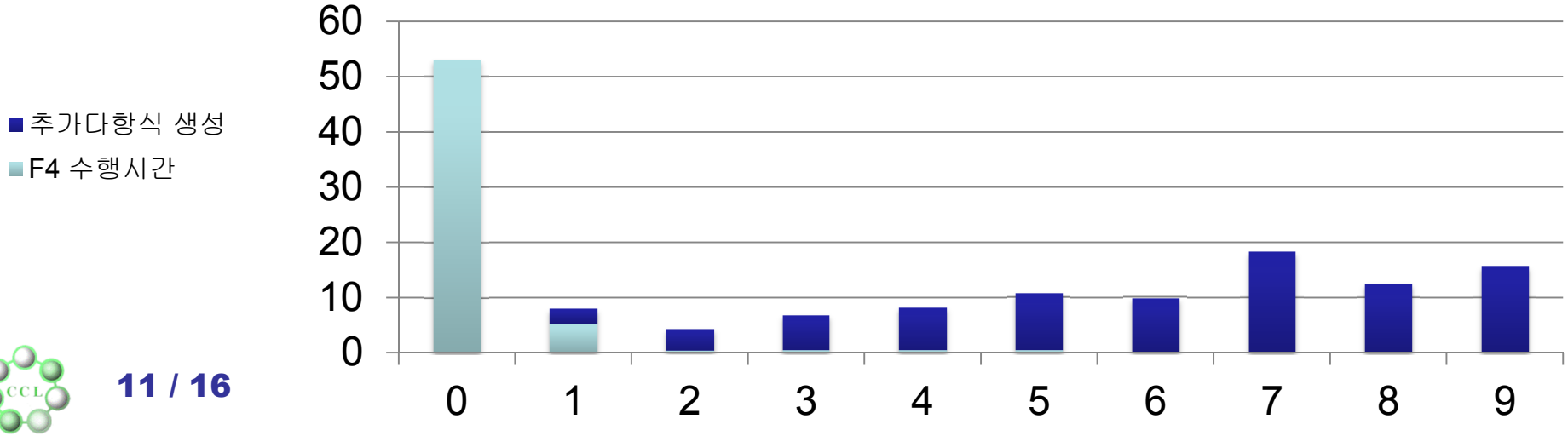
실험 방법 및 환경

- $n = 15, 16, 17, 18$에 대해
  - n개의 미지수로 이루어진 무작위 이차다항식 n개를 생성
  - $k = 0$~9벌의 추가 이차다항식을 생성, F4 알고리즘 수행
  - 각 경우에 대해 다음을 비교
    - ✓ 계산으로 구해지는 정규도 d_{reg}
    - ✓ 실제 계산 중에 확인되는 S-다항식의 최고차수 d_{max}
    - ✓ 얼마나 차이가 나는가?
  - F4 알고리즘 수행시간 + 추가다항식 생성시간
- 실험환경
  - Intel i3 540@3.07GHz, 2GB RAM, Windows7 64bit
    gcc v4.3.4@cygwin, c++ with STL library

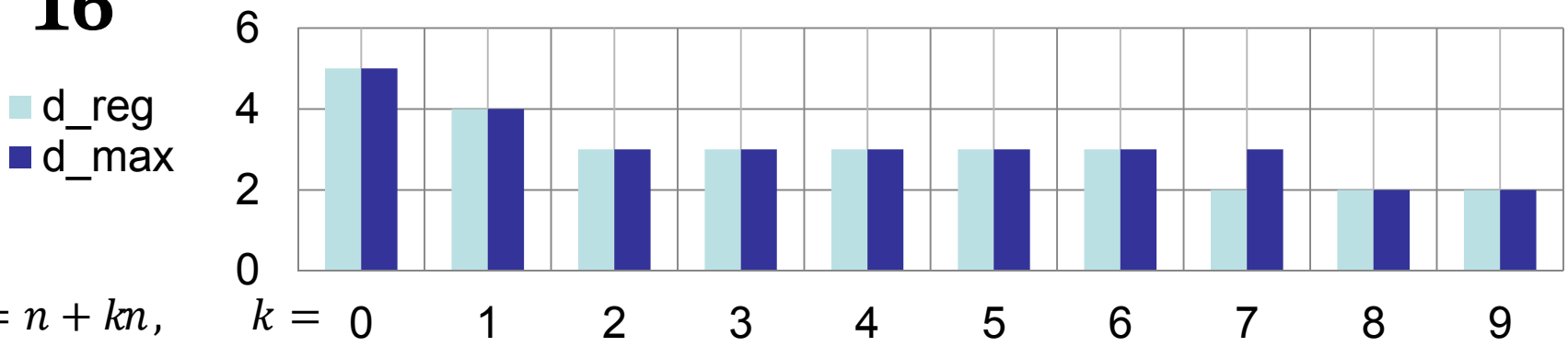
$n = 15$



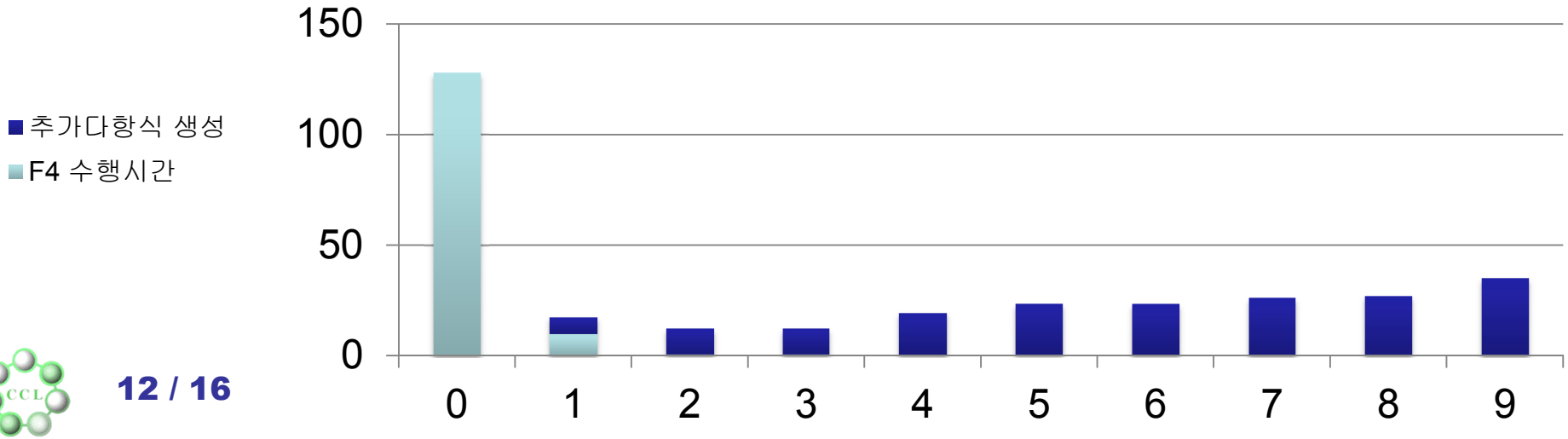
F4 수행 시간 (초)	53.06	5.34	0.45	0.55	0.58	0.56	0.06	0.02	0.02	0.03
추가다항식 생성시간 (초)	0	2.56	3.81	6.24	7.54	10.11	9.67	18.11	12.40	15.60
총 시간 (초)	51.06	7.90	4.26	6.79	8.12	10.67	9.73	18.13	12.42	15.63



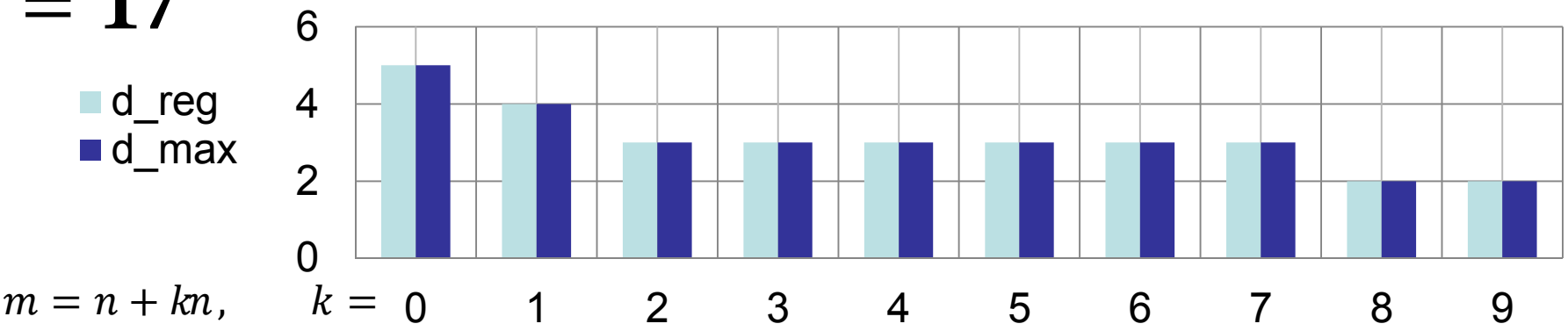
$n = 16$



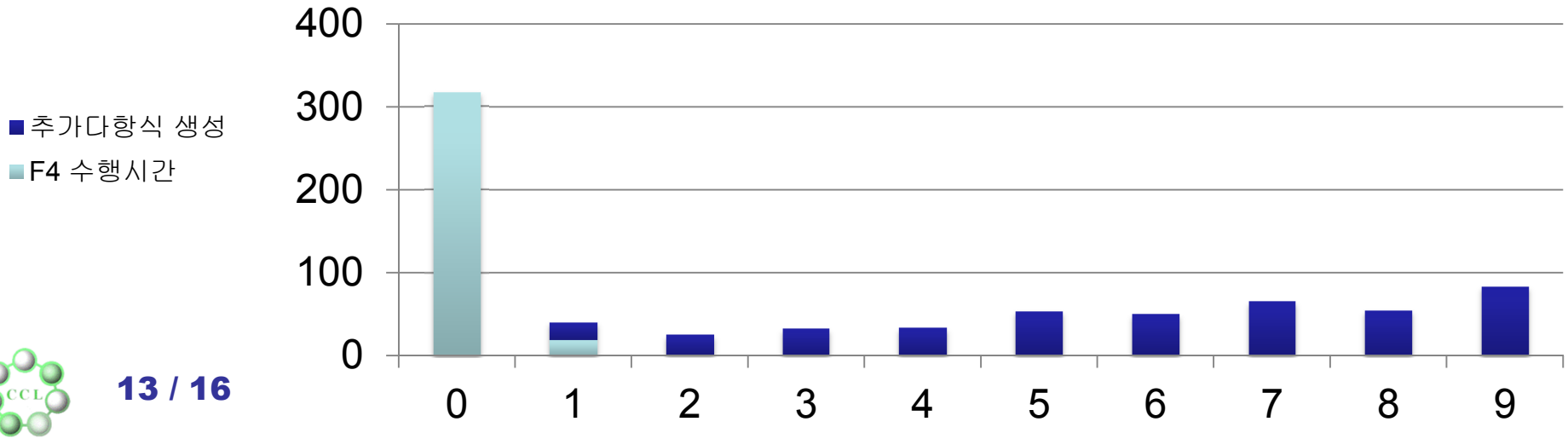
F4 수행 시간 (초)	127.81	9.80	0.66	0.80	0.87	0.87	0.78	0.05	0.03	0.03
추가다항식 생성시간 (초)	0	7.50	11.47	11.32	18.53	22.40	22.35	25.90	26.77	34.98
총 시간 (초)	127.81	17.30	12.13	12.12	19.40	23.27	23.13	25.95	26.80	35.01



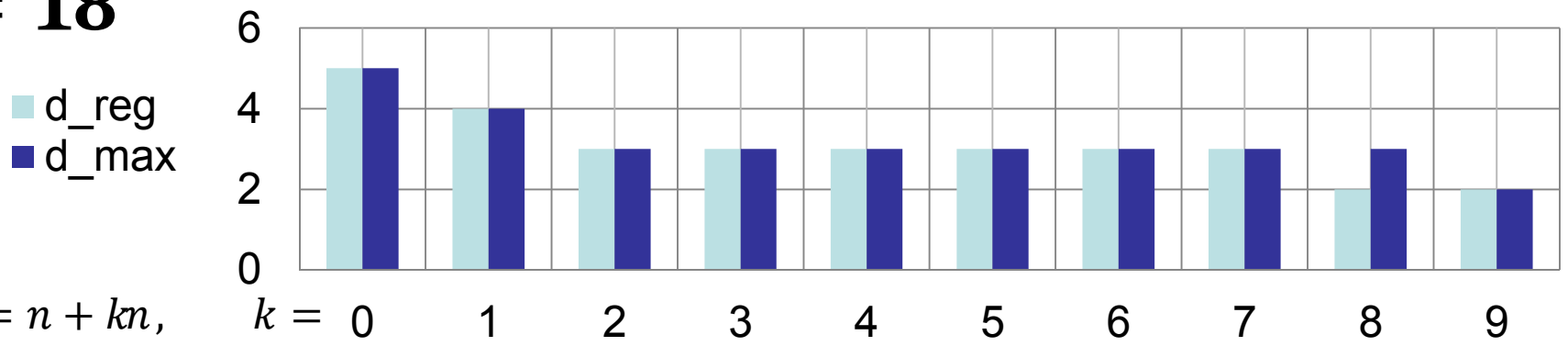
$n = 17$



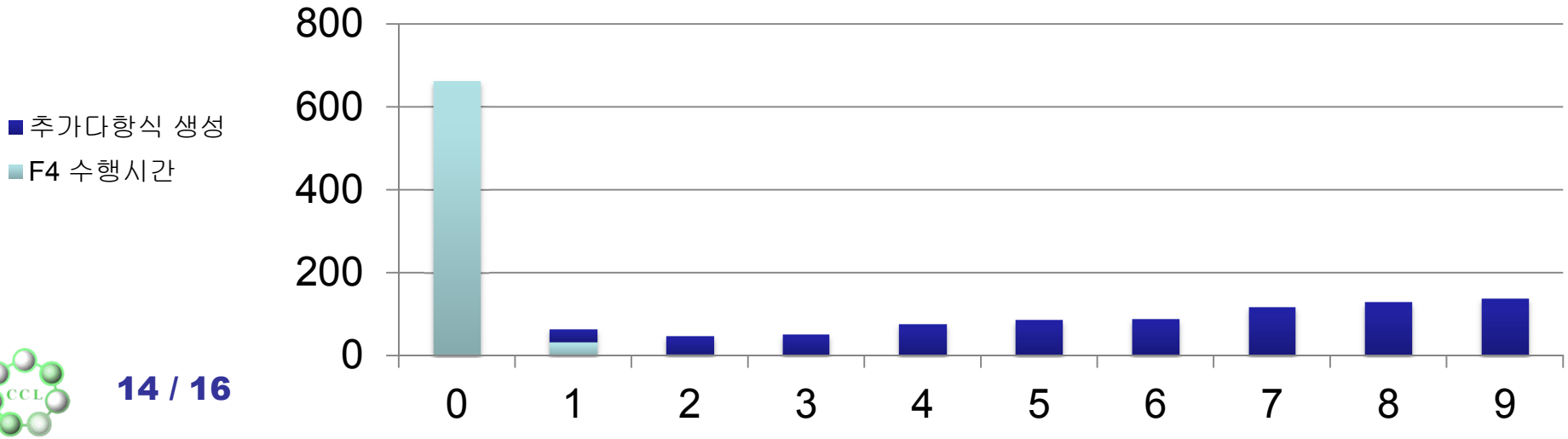
F4 수행 시간 (초)	316.99	18.92	1.22	1.11	1.26	1.33	1.23	0.90	0.03	0.05
추가다항식 생성시간 (초)	0	20.95	23.65	31.08	32.00	51.95	48.83	63.66	54.37	83.04
총 시간 (초)	316.99	39.87	24.87	32.19	33.26	63.28	50.06	64.56	54.40	83.09



$n = 18$



F4 수행 시간 (초)	662.99	32.40	3.20	1.58	1.83	1.93	1.86	1.61	0.08	0.05
추가다항식 생성시간 (초)	0	31.12	43.95	48.39	74.24	84.37	86.00	113.69	127.27	137.03
총 시간 (초)	662.99	63.52	47.15	49.97	76.07	86.30	87.86	115.30	127.35	137.08



결론

- 추가다항식을 통한 **F4**의 개선
  - 수학적으로 계산된 d_{reg} 값의 감소
  - 실험적으로 얻어진 d_{max}값의 감소 (d_{reg} 와 비슷, ± 1)
  - F4 알고리즘의 수행속도 개선 가능
- 실험 결과로는 추가다항식**+ F4**의 경우가 더 빠름
  - 추가다항식의 개수는 d_{reg} 값이 1, 2정도 감소되는 정도면 충분
  - 그 이상의 d_{reg} 감소를 위한 추가다항식은 너무 많이 필요함
- 복잡도
  - d차 추가다항식 1개의 생성에 필요한 복잡도는
    - ✓ $O(2^{n-d})$ (알고리즘 1)
    - ✓ $O(2^{2n-d})$ (알고리즘 2, 3)
  - F4의 복잡도 감소는 추정만 가능

Q&A

?