On the parallelization of a search for counterexamples to a conjecture of Erdős

On the parallelization of a search for counterexamples to a conjecture of Erdős

By

ShengWei Shen, B.A.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2012) McMaster University
(Computing and Software) Hamilton, Ontario

TITLE:	On the parallelization of a search for counterexamples to a conjecture of Erdős
AUTHOR:	ShengWei Shen, B.A.Sc. (McMaster University)
SUPERVISOR:	Dr. Antoine Deza
NUMBER OF PAGES:	xii, 54.

Abstract

Denote by $k_t(G)$ the number of cliques of order t in a graph G having n vertices. Let $k_t(n) = \min\{k_t(G) + k_t(\overline{G})\}$ where $\overline{G}$ denotes the complement of G. Let $c_t(n) = k_t(n)/\binom{n}{t}$ and c_t be the limit of $c_t(n)$ for n going to infinity. A 1962 conjecture of Erdős stating that $c_t = 2^{1-\binom{t}{2}}$ was disproved by Thomason in 1989 for all $t \geq 4$. Tighter counterexamples have been constructed by Jagger, Štovíček and Thomason in 1996, by Thomason for $t \leq 6$ in 1997, and by Franek for $t = 6$ in 2002. Further tightenings $t = 6, 7$ and 8 was recently obtained by Deza, Franek, and Liu.

We investigate the computational framework used by Deza, Franek, and Liu. In particular, we present the benefits and limitations of different parallel computer memory architectures and parallel programming models. We propose a functional decomposition approach which is implemented in C++ with POSIX thread (Pthread) libraries for multi-threading. Computational benchmarking on the parallelized framework and a performance analysis including a comparison with the original computational framework are presented.

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor, Dr. Antoine Deza, who has supported me throughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way. My special thanks go to the members of the examination committee: Dr. Antoine Deza, Dr. Frantisek Franek, and Dr. Ned Nedialkov.

I sincerely thank Dr. Frantisek Franek for many helpful and useful suggestions and discussions of this research.

I gratefully acknowledge Feng Xie for suggesting me the road to graduate study.

Many thanks go in particular to Min Jing(Jessie) Liu and Hongfeng Liang. I am much indebted to Jessie for many helpful discussion and advising on the research and letting me use the code she developed for the original computational framework. I have also benefited by the help from Hongfeng who always kindly grants me his time for solving some of my programming problems.

My special thanks go to Alvin Hsieh for all the help and discussions on the study and projects of graduate courses.

I would also gratefully appreciate the help throughout these two years from all the Advanced Optimization Laboratory members, Feng Xie, Min Jing Liu, Hongfeng Liang, Alvin Hsieh, Mei Jiang and Robert Fuller.

Finally, my deepest gratitude and love goes to my family for their dedication, patience and the many years of support during my university studies that provided the foundation for this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Basic definition

Definition 1.1.1 *a clique is a subset S of vertices in a graph G, such that every two vertices in S are connected.*

Denote by $k_t(G)$ the number of cliques of order t in the graph G. Let $k_t(n) = \min\{k_t(G) + k_t(\overline{G}) : n = |G|\}$, where $\overline{G}$ denotes the complement of G, and n denotes the order of G. In addition, $k_t(\overline{G})$ refers to the number of cocliques.

A complete graph on n vertices is often denoted by K_n, and $k_t(n)$ is then the minimum number of monochromatic $K_t s$ in a random colouring of the edges of K_n with two colours. Here, $K_t s$ is referred to cliques of order t or t-cliques, since any subgraph induced by a clique is a complete subgraph. Let $c_t(n) = k_t(n)/\binom{n}{t}$, and $c_t = \lim_{n\to\infty} c_t(n)$. Thus $c_t(n)$ denotes the minimum proportion of monochromatic $K_t s$ in a random colouring of edges of K_n with two colours, and c_t is the value of the multiplicity constant.

1.2 Background and previous work

An old conjecture of Erdős [6], related to Ramsey's Theorem, states that $c_t = 2^{1-\binom{t}{2}}$. It is intuitively true for $t = 2$, and it is also true for $t = 3$ follows from a result of Goodman's work [15]. One of the motivations behind the conjecture is the fact that the conjecture is true for any t while random graph is assumed.

In [7], Erdős and Moon showed that a modified conjecture for complete bipartite subgraphs of bipartite graphs is true. Sidorenko [19] showed that a modified conjecture for cycles is true, but not true for certain incomplete subgraphs. Franek and Rödl [11] showed that the original conjecture for $t = 4$ is true for nearly quasirandom, and thus quasirandom graphs.

Thomason disproved the conjecture in general for $t \geq 4$ in his paper [20]. The counter example he constructed are infinite sequences of graphs, which generated from a single underlying graph, with limits smaller than what the conjecture stipulates. The upper bounds for c_t that Thomason obtained as shown below:

$$\begin{aligned} t = 4, \quad & c_4 \leq 0.976 \times 2^{1-\binom{4}{2}} = 0.976 \times 2^{-5} \\ t = 5, \quad & c_5 \leq 0.906 \times 2^{1-\binom{5}{2}} = 0.906 \times 2^{-9} \\ t \geq 6, \quad & c_t \leq 0.936 \times 2^{1-\binom{t}{2}} \end{aligned}$$

By using Thomason's method in producing an infinite sequence of graphs from a single underlying graph, it then suffices to find just a single graph that satisfies certain conditions to generate a counterexample to the conjecture. However, the underlying graphs of Thomason's counterexamples are quite abstract and complicated. Later on, Thamason first improved these upper bounds in [21] to $c_4 \leq 0.9693 \times 2^{1-\binom{4}{2}}$ and $c_5 \leq 0.8801 \times 2^{1-\binom{5}{2}}$.

Franek and Rödl illustrated an alternative proof in [10, 12] by using graphs generated by computer as a simpler counterexample, they achieved a same upper bound as the one Thomason obtained. These upper bounds were further improved to $c_6 \leq 0.7446 \times 2^{1-\binom{6}{2}}$ by Franek [13], and $c_t \leq 0.835 \times 2^{1-\binom{t}{2}}$ for $t \geq 7$ by Jagger, Šťovíček and P. Thomason [16] On the other hand, the lower bound of c_t is also improved by various people. see Giraud [14] who showed that $c_4 \geq 0.695 \times 2^{1-\binom{4}{2}}$, and Colon [4] for a recent improvement over Erdős's original application of Ramsey's Theorem.

The simple construction used in [13] for $t = 6$ is based on the approach conducted by Franek and Rödl [12] who tied the best upper bound for c_4. Based on this simple construction, Deza, Franek and Liu [5] investigate a computational framework to search for tighter upper bounds for small t. They verified that the construction used in [13] for $t = 6$ also improves the previously known best upper bound for $t = 7$ to $c_7 \leq 0.7156 \times 2^{1-\binom{7}{2}}$. Note that they have also achieved the best upper bounds c_6 and c_7 without searching for potential tighter constructions. The up to date upper bounds they obtained in [5] for c_t compare to the previous results are shown in Table 1.1

t	Previous Result	New Result	Family(X,F) for New Result
4	$0.9693 \times 2^{1-\binom{4}{2}}$	$0.97650 \times 2^{1-\binom{4}{2}}$	$(X, F) = (10, \{1, 3, 4, 7, 8, 10\})$
5	$0.8801 \times 2^{1-\binom{5}{2}}$	$0.88584 \times 2^{1-\binom{5}{2}}$	$(X, F) = (10, \{1, 3, 4, 7, 8, 10\})$
6	$0.7446 \times 2^{1-\binom{6}{2}}$	$0.74444 \times 2^{1-\binom{6}{2}}$	$(X, F) = (10, \{1, 3, 4, 7, 8\})$
7	$0.835 \times 2^{1-\binom{7}{2}}$	$0.68690 \times 2^{1-\binom{7}{2}}$	$(X, F) = (11, \{3, 4, 7, 8, 10, 11\})$
8	$0.835 \times 2^{1-\binom{8}{2}}$	$0.70014 \times 2^{1-\binom{8}{2}}$	$(X, F) = (12, \{1, 3, 4, 7, 8, 11, 12\})$
> 8	$0.835 \times 2^{1-\binom{t}{2}}$	...	...

Table 1.1: upper bounds up to date

In order to further improved the upper bounds for $t \leq 8$, Parallelization is needed to be introduced into the computational framework for better performance on searching the potential tighter constructions. In this thesis, we will address the way of how parallelization is applied, the performance comparison to the current computational framework, the difficulty of the application and the potential drawbacks of the new framework.

Chapter 2

Counter Example Construction

2.1 Graph generation

For the purpose of improving the upper bound for c_t of some small t, we need some graphs that the number of cliques and cocliques can be expressed in closed form. By following the approach used in [12, 13, 5], a so-called Boolean graphs were introduced to compute the Erdős's conjecture. they have exhibited constructions with low numbers of cliques and cocliques. In spite of the large order $(2^{10} - 2^{14})$of these graphs, they are rather simple and highly regular.

Definition 2.1.1 *Let X be a finite set of size n and $F \subseteq \{1, 2, ..., n\}$. Define the graph $G_{X,F}$ which vertices are all 2^n subsets of X. And two subset x_i and x_j of X are connected by an edge if and only if $|x_i \triangle x_j| \in F$, where $x_i \triangle x_j$ denotes the symmetric difference of the sets x_i and x_j.*

Denote $\overline{F}$ as the complement of F, such that $\overline{F} = \{1, 2, ..., n\} - F$, then we have $\overline{G}_{X,F} = G_{X,\overline{F}}$.

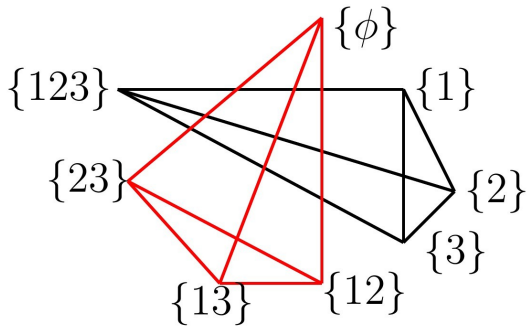


Figure 2.1: A simple $G_{X,F}$ with $|X| = 3$, and $F = \{2\}$

Definition 2.1.2 *For $m \geq 1$, An ordered m-tuple,$\langle x_0, x_1 \cdots, x_{m-1}\rangle$ is an (X, F)-m-*tuple*, if $x_i \subseteq X$ and $|x_i| \in F$ for each $i < m$, and $|x_i \triangle x_j| \in F$ for all $i \neq j < m$.*

(X, F)-1-tuple, (X, F)-2-tuple and (X, F)-3-tuple was named as X, F-singleton, X, F-pair, and X, F-triple respectively by Franek and Rödl in [10]. For the purpose of generalization, we will use (X, F)-m-tuple for $m \geq 1$.

Lemma 2.1.1 *For a given X and F, let S_m denote the number of (X, F)-m-tuples, and $k_{m+1}(G_{X,F})$ denote the number of cliques of size $m+1$ in the graph $G_{X,F}$. Then:*

$$k_{m+1}(G_{X,F}) = \frac{2^n}{(m+1)!} S_m(X, F) : n = |X|.$$

Proof 2.1.1 *Follow the proof in [10]*

(1) when $m = 2$,we want to proof: $k_3(G_{X,F}) = \frac{2^n}{(3)!} S_2(X, F)$.

Let $\{a, b, c\}$ be a 3-clique in $G_{X,F}$. It is easy to verify that $\langle a \triangle b, a \triangle c\rangle$ is (X, F)-2-tuple, and all elements in the 2-tuple are mutually distinct, But any permutation of the two elements in the 2-tuple forms an (X, F)-2-tuple.

Thus there are 2 distinct (X, F)*-2-tuples. We could have chosen any of the vertices of the cliques to determine the 2* (X, F)*-2-tuples, not just a. Therefore the clique* $\{a, b, c\}$ *determines* $3 \times 2 = 6$ *distinct* (X, F)*-2-tuples. On the other hand, it is easy to show that if* $\langle x_0, x_1 \rangle$ *is an* (X, F)*-2-tuple, and if* $a \subseteq X$*, then* $\{a, a\triangle x_0, a\triangle x_1\}$ *is a 3-clique in* $G_{X,F}$*. Thus there are exactly* $\frac{2^n}{(3)!}S_2(X, F)$ *3-cliques in* $G_{X,F}$*.*

(2) when $m = 3$*,we want to proof:* $k_4(G_{X,F}) = \frac{2^n}{(4)!}S_3(X, F)$*.*

Let $\{a, b, c, d\}$ *be a 4-clique in* $G_{X,F}$*. It is easy to verify that* $\langle a\triangle b, a\triangle c, a\triangle d\rangle$ *is* (X, F)*-3-tuple, and all elements in the triple are mutually distinct, But any permutation of the three elements in the* 3*-tuple forms an* (X, F)*-3-tuple. Thus there are 6 distinct* (X, F)*-3-tuples. We could have chosen any of the vertices of the cliques to determine the 6* (X, F)*-3-tuples, not just a. Therefore the clique* $\{a, b, c, d\}$ *determines* $4 \times 6 = 24$ *distinct* (X, F)*-3-tuples. On the other hand, it is easy to show that if* $\langle x_0, x_1, x_2\rangle$ *is an* (X, F)*-3-tuple, and if* $a \subseteq X$*, then* $\{a, a\triangle x_0, a\triangle x_1, a\triangle x_2\}$ *is a 4-clique in* $G_{X,F}$*. Thus there are exactly* $\frac{2^n}{(4)!}S_3(X, F)$ *4-cliques in* $G_{X,F}$*.*

(k) we want to proof: $k_{m+1}(G_{X,F}) = \frac{2^n}{(m+1)!}S_m(X, F)$*.*

Let $\{x_0, x_1, ...x_m\}$ *be a* $(m + 1)$*-clique in* $G_{X,F}$*. It is easy to verify that* $\langle x_0\triangle x_1, x_0\triangle x_2, ..., x_0\triangle x_m\rangle$ *is* (X, F)*-m-tuple, and all elements in the m-tuples are mutually distinct, But any permutation of the* m *elements in the* m*-tuple forms an* (X, F)*-m-tuple. Thus there are* $m!$ *distinct* (X, F)*-m-tuples. We could have chosen any of the vertices of the cliques to determine the* $m!$ (X, F)*-m-tuples, not just* x_0*. Therefore the clique* $\{x_0, x_1, ...x_m\}$ *determines* $(m + 1) \times m! = (m + 1)!$ *distinct* (X, F)*-m-tuples. On the other hand, it is easy to*

show that if $\langle x_0 \triangle x_1, x_0 \triangle x_2, ..., x_0 \triangle x_m \rangle$ *is an* (X, F)*-m-tuple, and if* $a \subseteq X$*, then* $\{a, a \triangle x_0, a \triangle x_1, ..., a \triangle x_m\}$ *is a m-clique in* $G_{X,F}$*. Thus there are exactly* $\frac{2^n}{(m+1)!} S_m(X, F)$ $(m+1)$*-cliques in* $G_{X,F}$*.*

Like Thomason did in [20], we are able to generate an infinite sequence of graphs by expanding a single graph that is mentioned above:

Definition 2.1.3 *Given a graph G of order n, and a positive integer d. The graph* G^d *is obtained by "blown up" the graph G: where each vertex of G is replaced by a d-cliques. Thus there are* $\underline{nd}$ *vertices in* G^d*. Besides the edges within the created d-cliques, there is an edge between two vertices* v_i *and* v_j *of* G^d *if and only if an edge existed in G between the original G-vertices* k_i *and* k_j *where* v_i *and* v_j *are contained in the corresponding "blown up" d-cliques formed from* k_i *and* k_j*,* $i \neq j$*.*

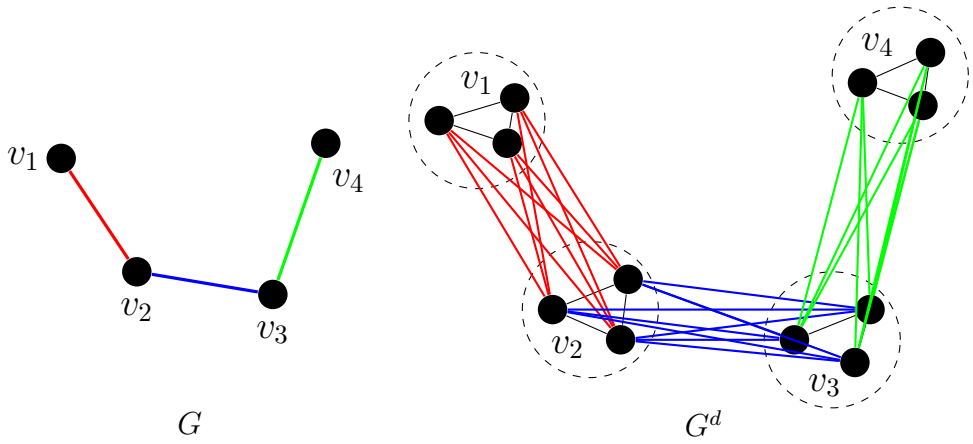


Figure 2.2: An example shows graph G and G^d, with $n = 4, d = 3$

Figure 2.2 shows a simple example of a graph G which has 4 vertices and 3 edges. If we set $d = 3$, then the new graph G^d have 12 vertices and 39 edges.

2.2 Evaluating Erdős number $\hat{c}_t$

This section is based on Franek and Rödl paper [10] and personal communication with Liu [17]. In order to be distinguished to the conjecture c_t, we denote the number of computer-generated graphs that previously described by $\hat{c}_t$.

Lemma 2.2.1 *If all graphs in an infinite sequence of graphs $\{G^d\}_{d=0}^{\infty}$ were obtained from a single graph G of size n as defined in Def. 2.1.1, then*

$$\hat{c_4} \leq \lim_{d\to\infty} \frac{k_4(G^d)+k_4(\overline{G^d})}{\binom{nd}{4}} = \frac{24(k_4(G)+k_4(\bar{G}))+36k_3(G)+14k_2(G)+k_1(G)}{n^4}$$

Proof 2.2.1 *Fix an d, Let's calculate the number of all 4-cliques in G^d; there are 5 possible cases:*

(1) Each of the four vertices are from distinct "blown up" G-vertices.

Since each vertex of such a 4-clique can be chosen independently, there are $\binom{d}{1}^4 k_4(G)$ such 4-cliques in G^d. Denote this number as $Q_1(d)$.

(2) Two of the four vertices of the 4-clique are from the same "blown up" G-vertex, while the remaining two vertices are from distinct "blown-up" G-vertices.

There are 3 ways to choose the "blown-up" G-vertex with two vertices, and $\binom{d}{2}$ ways to choose the vertices in it. The remaining two can be chosen independently, hence there are $3\binom{d}{2}\binom{d}{1}^2 k_3(G)$ such 4-cliques in G^d. Denote this number as $Q_2(d)$.

(3) Three of the four vertices of the 4-clique are from the same "blown up" G-vertex, while the remaining one vertex is from a different "blown up" G-vertex.

There are 2 ways to choose the "blown up" G-vertex with three vertices, and $\binom{d}{3}$ *ways to choose the vertices in it. The remaining two can be chosen independently, hence there are* $2\binom{d}{3}\binom{d}{1}k_2(G)$ *such 4-cliques in* G^d*. Denote this number as* $Q_3(d)$*.*

(4) Two of the four vertices of the 4-clique are from the same "blown up" G-vertex, while the remaining two vertex is from a different "blown up" G-vertex.

There are $\binom{d}{2}$ *ways to choose the two vertices in each of the "blown up" G-vertices. Hence there are* $\binom{d}{2}^2 k_2(G)$ *such 4-cliques in* G^d*. Denote this number as* $Q_4(d)$*.*

(5) Finally, the four vertices of the 4-clique are from a single "blown up" G-vertex.

There are $\binom{d}{4}$ *ways to choose the four vertices in the "blown up" G-vertex. Hence there are* $\binom{d}{4}k_1(G)$ *such 4 cliques in* G^d*. Here,* $k_1(G) = n$*. Denote this number as* $Q_5(d)$*.*

Thus we will have:

$$\begin{aligned}
k_4(G^d) &= Q_1(d) + Q_2(d) + Q_3(d) + Q_4(d) + Q_5(d) \\
&= \tbinom{d}{1}^4 k_4(G) + 3\tbinom{d}{2}\tbinom{d}{1}^2 k_3(G) + [2\tbinom{d}{3}\tbinom{d}{1} + \tbinom{d}{2}^2]k_2(G) + \tbinom{d}{4}k_1(G) \\
&= d^4 k_4(G) + 3\tbinom{d}{2}d^2 k_3(G) + [2\tbinom{d}{3}d + \tbinom{d}{2}^2]k_2(G) + \tbinom{d}{4}k_1(G) \\
&= d^4 k_4(G) + \frac{3}{2}d^4 k_3(G)O_1(d) + [\frac{2}{3!}d^4 O_2(d) + \frac{1}{4}d^4 O_3(d)]k_2(G) + \frac{1}{4!}d^4 k_1(G)O_4(d)
\end{aligned}$$

where $O_1(d) = \frac{d-1}{d}$*,* $O_2(d) = \frac{(d-1)(d-2)}{d^2}$*,* $O_3(d) = \frac{(d-1)^2}{d^2}$*,* $O_4(d) = \frac{(d-1)(d-2)(d-3)}{d^3}$*.*

$$\lim_{d\to\infty}\frac{k_4(G^d)}{\binom{nd}{4}} = \lim_{d\to\infty}\frac{\sum Q_i(d)}{\binom{nd}{4}}$$
$$= \lim_{d\to\infty}\frac{d^4k_4(G)+\frac{3}{2}d^4k_3(G)O_1(d)+[\frac{2}{3!}d^4O_2(d)+\frac{1}{4}d^4O_3(d)]k_2(G)+\frac{1}{4!}d^4k_1(G)O_4(d)}{\binom{nd}{4}}$$
$$= \lim_{d\to\infty}\frac{d^4[k_4(G)+\frac{3}{2}k_3(G)O_1(d)+[\frac{2}{3!}O_2(d)+\frac{1}{4}O_3(d)]k_2(G)+\frac{1}{4!}k_1(G)O_4(d)]}{\frac{(nd)^4}{4!}\frac{(nd-1)(nd-2)(nd-3)}{(nd)^3}}$$
$$= \frac{\lim_{d\to\infty} d^4[k_4(G)+\frac{3}{2}k_3(G)O_1(d)+[\frac{2}{3!}O_2(d)+\frac{1}{4}O_3(d)]k_2(G)+\frac{1}{4!}k_1(G)O_4(d)]}{\lim_{d\to\infty}\frac{(nd)^4}{4!}O_5(d)}$$

And we know that:

$O_1(d) = \frac{d-1}{d} = 1-\frac{1}{d}$,

$O_2(d) = \frac{(d-1)(d-2)}{d^2} = 1-\frac{3}{d}+\frac{2}{d^2}$,

$O_3(d) = \frac{(d-1)^2}{d^2} = 1-\frac{2}{d}+\frac{1}{d^2}$,

$O_4(d) = \frac{(d-1)(d-2)(d-3)}{d^3} = 1-\frac{6}{d}+\frac{11}{d^2}-\frac{6}{d^3}$,

$O_5(d) = \frac{(nd-1)(nd-2)(nd-3)}{(nd)^3} = 1-\frac{6}{nd}+\frac{11}{n^2d^2}-\frac{6}{n^3d^3}$

Thus, $\lim_{d\to\infty} O_i(d) = 1$, *and both* d^4 *in the numerator and denominator can be cancelled.*

$$= \frac{k_4(G)+\frac{3}{2}k_3(G)+[\frac{2}{3!}+\frac{1}{4}]k_2(G)+\frac{1}{4!}k_1(G)}{\frac{n^4}{4!}}$$
$$= \frac{4!*[k_4(G)+\frac{3}{2}k_3(G)+[\frac{2}{3!}+\frac{1}{4}]k_2(G)+\frac{1}{4!}k_1(G)]}{4!*\frac{n^4}{4!}}$$
$$= \frac{24k_4(G)+36k_3(G)+14k_2(G)+k_1(G)}{n^4}$$

Since for $\overline{G^d}$ *only case (1) can happen,* $\lim_{d\to\infty}\frac{k_4(\overline{G^d})}{\binom{nd}{4}} = \frac{24k_4(\overline{G^d})}{n^4}$

Therefore, $\lim_{d\to\infty}\frac{k_4(G^d)+k_4(\overline{G^d})}{\binom{nd}{4}} = \frac{24(k_4(G)+k_4(\bar{G}))+36k_3(G)+14k_2(G)+k_1(G)}{n^4}$ $\square$

Proof. 2.2.1 has shown the way of partitioning the problem with $t = 4$ into 4 cases, whereas case (3) and (4) can be considered as a single case since both cases count the number of $k_2(G)$ type 4-cliques in G^d. Table 2.1 shows the possible partitioning number when $t = 4, 5, 6, 7$

t	cases	Q(d)
4	{1,1,1,1}	$\binom{d}{1}^4 k_4(G)$
	{1,1,2}	$3\binom{d}{2}\binom{d}{1}^2 k_3(G)$
	{1,3}&{2,2}	$[2\binom{d}{3}\binom{d}{1} + \binom{d}{2}^2]k_2(G)$
	{4}	$\binom{d}{4}k_1(G)$
5	{1,1,1,1,1}	$\binom{d}{1}^5 k_5(G)$
	{1,1,1,2}	$4\binom{d}{2}\binom{d}{1}^3 k_4(G)$
	{1,1,3}&{1,2,2}	$[3\binom{d}{3}\binom{d}{1}^2 + 3\binom{d}{2}^2\binom{d}{1}]k_3(G)$
	{1,4}&{2,3}	$[2\binom{d}{1}\binom{d}{4} + 2\binom{d}{3}\binom{d}{2}]k_2(G)$
	{5}	$\binom{d}{5}k_1(G)$
6	{1,1,1,1,1,1}	$\binom{d}{1}^6 k_6(G)$
	{1,1,1,1,2}	$5\binom{d}{2}\binom{d}{1}^4 k_5(G)$
	{1,1,1,3}&{1,1,2,2}	$[4\binom{d}{3}\binom{d}{1}^3 + \binom{4}{2}\binom{d}{2}^2\binom{d}{1}^2]k_4(G)$
	{1,1,4}&{1,2,3}&{2,2,2}	$[3\binom{d}{4}\binom{d}{1}^2 + 3 * 2\binom{d}{3}\binom{d}{2}\binom{d}{1} + \binom{d}{2}^3]k_3(G)$
	{1,5}&{2,4}&{3,3}	$[2\binom{d}{1}\binom{d}{5} + 2\binom{d}{2}\binom{d}{4} + \binom{d}{3}\binom{d}{3}]k_2(G)$
	{6}	$\binom{d}{6}k_1(G)$
7	{1,1,1,1,1,1,1}	$\binom{d}{1}^7 k_7(G)$
	{1,1,1,1,1,2}	$6\binom{d}{2}\binom{d}{1}^5 k_6(G)$
	{1,1,1,1,3}&{1,1,1,2,2}	$[5\binom{d}{3}\binom{d}{1}^4 + \binom{5}{2}\binom{d}{2}^2\binom{d}{1}^3]k_5(G)$
	{1,1,1,4}&{1,1,2,3}&{1,2,2,2}	$[4\binom{d}{4}\binom{d}{1}^3 + 4 * 3\binom{d}{3}\binom{d}{2}\binom{d}{1}^2 + 4\binom{d}{2}^3\binom{d}{1}]k_4(G)$
	{1,1,5}&{1,2,4}&{1,3,3}&{2,2,3}	$[3\binom{d}{1}^2\binom{d}{5} + 3 * 2\binom{d}{1}\binom{d}{2}\binom{d}{4} + 3\binom{d}{1}\binom{d}{3}\binom{d}{3} + 3\binom{d}{2}^2\binom{d}{3}]k_3(G)$
	{1,6}&{2,5}&{3,4}	$[2\binom{d}{1}\binom{d}{6} + 2\binom{d}{2}\binom{d}{5} + 2\binom{d}{3}\binom{d}{4}]k_2(G)$
	{7}	$\binom{d}{7}k_1(G)$

Table 2.1: Possible partitioning number

Lemma 2.2.2 *By following the partitioning method shown above, we could get the Erdős number of the graph G of size $t = 5, 6, 7$:*

$$\hat{c_5} \leq \lim_{d\to\infty} \frac{k_5(G^d) + k_5(\overline{G^d})}{\binom{nd}{5}} = \frac{120(k_5(G)+k_5(\bar{G}))+240k_4(G)+150k_3(G)+30k_2(G)+k_1(G)}{n^5}.$$

$$\hat{c_6} \leq \lim_{d\to\infty} \frac{k_6(G^d) + k_6(\overline{G^d})}{\binom{nd}{6}} = \frac{720(k_6(G)+k_6(\bar{G}))+1800k_5(G)+1560k_4(G)+540k_3(G)+62k_2(G)+k_1(G)}{n^6}.$$

$$\hat{c_7} \leq \lim_{d\to\infty} \frac{k_7(G^d) + k_7(\overline{G^d})}{\binom{nd}{7}} = \frac{5040(k_7(G)+k_7(\bar{G}))+15120k_6(G)+16800k_5(G)+8400k_4(G)+1806k_3(G)+126k_2(G)+k_1(G)}{n^7}.$$

Let $G = G_{X,F}$, and substitute $k_m(G_{X,F})$ by $S_{m-1}(X, F)$ using Lemma 2.1.1, then we can restate Lemma 2.2.2 as:

Lemma 2.2.3 *For a given pair (X, F), follows from Lemma 2.1.1 we know the size of the graph $G_{X,F}$ is 2^n, then we have:*

$$\hat{c_5} \leq \lim_{d\to\infty} \frac{k_5(G^d_{X,F}) + k_5(\overline{G^d_{X,F}})}{\binom{d2^n}{5}} = \frac{S_4(X,F)+S_4(X,\bar{F})+10S_3(X,F)+25S_2(X,F)+15S_1(X,F)+1}{2^{4n}}.$$

$$\hat{c_6} \leq \lim_{d\to\infty} \frac{k_6(G^d_{X,F}) + k_6(\overline{G^d_{X,F}})}{\binom{d2^n}{6}} = \frac{S_5(X,F)+S_5(X,\bar{F})+15S_4(X,F)+65S_3(X,F)+90S_2(X,F)+31S_1(X,F)+1}{2^{5n}}.$$

$$\hat{c_7} \leq \lim_{d\to\infty} \frac{k_7(G^d_{X,F}) + k_7(\overline{G^d_{X,F}})}{\binom{d2^n}{7}} = \frac{S_6(X,F)+S_6(X,\bar{F})+21S_5(X,F)+140S_4(X,F)+350S_3(X,F)+301S_2(X,F)+63S_1(X,F)+1}{2^{6n}}.$$

Note 2.2.1 *Recall that the Erdős's conjecture states that* $c_t = 2^{1-\binom{t}{2}}$. *In order to find the upper bound for the conjecture, we divide the number of computer-generated graphs* $\hat{c_t}$ *by the conjecture number* c_t. *For* $t = 5$, $c_t = 2^{1-\binom{5}{2}} = 2^{-9}$ *Thus we get:*

$$ratio_5 = \frac{\hat{c_5}}{c_5} = \frac{S_4(X,F)+S_4(X,\bar{F})+10S_3(X,F)+25S_2(X,F)+15S_1(X,F)+1}{2^{4n-9}}$$

Similarly,

$$ratio_6 = \frac{\hat{c_6}}{c_6} = \frac{S_5(X,F)+S_5(X,\bar{F})+15S_4(X,F)+65S_3(X,F)+90S_2(X,F)+31S_1(X,F)+1}{2^{5n-14}}$$

$$ratio_7 = \frac{\hat{c_7}}{c_7} = \frac{S_6(X,F)+S_6(X,\bar{F})+21S_5(X,F)+140S_4(X,F)+350S_3(X,F)+301S_2(X,F)+63S_1(X,F)+1}{2^{6n-20}}$$

2.3 Relationship of the coefficients of k_i and S_i

Liu [17] have noticed an important relationship between the coefficients of k_i and the coefficients of S_i when t changes. Table 2.2 and 2.3 show the coefficients of k_i and S_i respectively with different values of t.

t	S_1	S_2	S_3	S_4	S_5	S_6	S_7
4	7	6	1				
5	15	25	10	1			
6	31	90	65	15	1		
7	63	301	350	140	21	1	
8	127	966	1701	1050	266	28	1

Table 2.2: The coefficient of $S_i(X, F)$.

t	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
4	1	14	36	24				
5	1	30	150	240	120			
6	1	62	540	1560	1800	720		
7	1	126	1806	8400	16800	15120	5040	
8	1	254	5796	40824	126000	191520	141120	40320

Table 2.3: The coefficient of $k_i(X, F)$

All the values in the tables 2.2 and 2.3 were calculated by hand. From the two tables, it follows that, if we denote the coefficient of $S_i(X, F)$ with $t = j$ is $\alpha_{i,j}$, and the coefficient of $k_i(X, F)$ with $t = j$ is $\beta_{i,j}$, we get:

$$\alpha_{i,j} = \alpha_{i,j-1} \times (i + 1) + \alpha_{i-1,j-1};$$

$$\beta_{i,j} = (\beta_{i,j-1} + \beta_{i-1,j-1}) \times i;$$

Although Liu [17] hasn't come up with a formal proof of this relationship in the meantime, but she strongly believes that it is true.

Chapter 3

Computational Framework

In this chapter, we are going to describe more detail on the computational framework that Deza, Franek, and Liu have used in [5] to compute the upper bound for the conjecture. Lemma 2.2.3 presents a closed form for the upper bound of the c_i for a given pair of (X, F). In order to constitutes a counter example to the Erdős's conjecture, the next step is to find a set X and a family $F \subseteq \{1, 2, 3, ..., n\}$ so that the value of the $rate_i$ in Note 2.2.1 is less than one. The computational framework consists of a routine to compute all the required $S_i(X, F)$ and a routine to perform a search for the best (X, F).

3.1 Computation detail on $S_i(X, F)$

First, we'll discuss the method that was previously used in [12, 13] for computing $S_i(X, F)$. Liu [5] states that this method is rather slow and cannot be applied for $t > 7$, thus only a single pair $(10, \{1, 3, 4, 7, 8, 10\})$ was used in [13].

3.1.1 Compute $S_1(X, F)$

$S_1(X, F)$ is the basic case of computing $S_i(X, F)$, consider a 1-tuple $\langle x_0 \rangle$ where $x_0 \subseteq X$. We need to generate all possible $|x_0| \in F$. Therefore, the computation of $S_1(X, F)$ is quite straightforward, such that:

$$S_1(X, F) = \sum_{|x_0| \in F} \binom{n}{|x_0|} : \ n = |X|$$

3.1.2 Compute $S_2(X, F)$

Now consider an ordered 2-tuple $\langle x_0, x_1 \rangle$ where $x_0, x_1 \subseteq X$ and they are mutually distinct. By the definition of intuitive set theory, we know that $x_0 - x_1$, $x_1 - x_0$ and $x_0 \cap x_1$ are mutually disjoint. Let $m_0 = |x_0 - x_1|$, $m_0 = |x_1 - x_0|$ and $m_{01} = |x_0 \cap x_1|$.

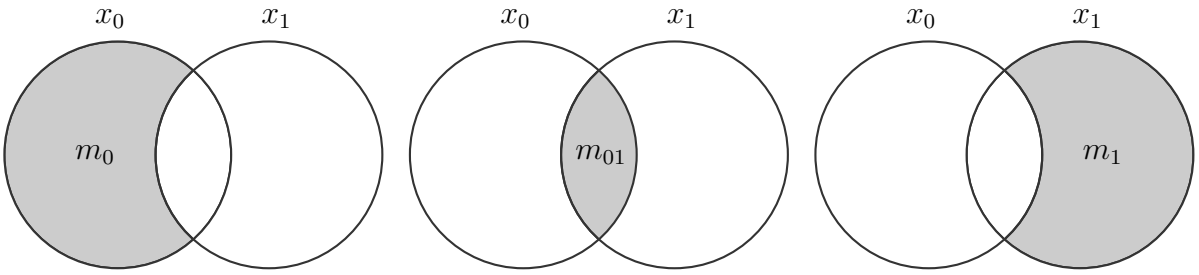


Figure 3.1: The representation of ms in S_2 in Venn diagram

Figure 3.1 shows the relationship of these ms, and then we'll have:

$$m_0 + m_{01} = |x_0|,$$

$$m_1 + m_{01} = |x_1|,$$

$$m_0 + m_1 = |x_0 \triangle x_1|;$$

According to Definition 2.1.2, we know that $|x_0|$, $|x_1|$ and $|x_0 \triangle x_1| \in F$. Hence we could compute the value of

$$S_2(X,F) = \sum_{m_0,m_1,m_{01} \in X} \binom{n}{m_0} \cdot \binom{n-m_0}{m_1} \cdot \binom{n-m_0-m_1}{m_{01}}$$

For all possible combinations of $\langle m_0, m_1, m_{01} \rangle$, we first list all the possible combinations of m_0 ,m_1 and m_{01}, and check the validation of those $|x_0|$, $|x_1|$ and $|x_0 \triangle x_1|$.

3.1.3 Compute $S_3(X,F)$

Consider an ordered 3-tuple $\langle x_0, x_1, x_2 \rangle$ where $x_0, x_1, x_2 \subseteq X$ and they are mutually distinct. Similarly we will have the following conditions:

1. $|x_0| \in F$
2. $|x_1| \in F$
3. $|x_2| \in F$
4. $|x_0 \triangle x_1| \in F$
5. $|x_0 \triangle x_2| \in F$
6. $|x_1 \triangle x_2| \in F$
7. $|x_0 \cup x_1 \cup x_2| \leq |X|$

Let $m_{012} = |x_0 \cap x_1 \cap x_2|$, $m_{01} = |x_0 \cap x_1| - m_{012}$, $m_{02} = |x_0 \cap x_2| - m_{012}$, $m_{12} = |x_1 \cap x_2| - m_{012}$, $m_0 = |x_0 - (x_1 \cup x_2)|$, $m_1 = |x_1 - (x_0 \cup x_2)|$, $m_2 = |x_2 - (x_0 \cup x_1)|$. Then those ms are mutually disjoint. Rewrite the previous conditions in terms of ms, we would get:

1. $m_0 + m_{01} + m_{02} + m_{012} \in F$
2. $m_1 + m_{01} + m_{12} + m_{012} \in F$
3. $m_2 + m_{12} + m_{02} + m_{012} \in F$
4. $m_0 + m_{02} + m_1 + m_{12} \in F$
5. $m_0 + m_{01} + m_2 + m_{12} \in F$
6. $m_1 + m_{01} + m_2 + m_{02} \in F$
7. $m_0 + m_1 + m_2 + m_{01} + m_{02} + m_{12} + m_{012} \leq |X|$

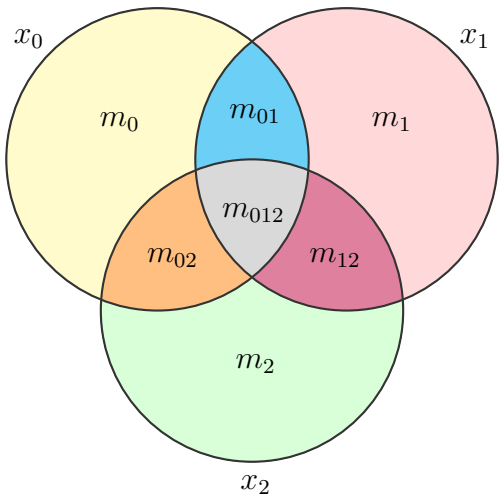


Figure 3.2: The representation of ms in S_3 in Venn diagram

Thus we could compute the value of S_3 by using the following formula:

$$S_3(X, F) = \sum_{valid\, m's} \binom{n}{m_0} \cdot \binom{n-m_0}{m_1} \cdot \binom{n-m_0-m_1}{m_{01}} \cdots$$

In order to find all possible combinations of ms by listing all the possible combinations of ms, and check the validation. Figure 3.2 also shows the relationship of these ms in S_3.

3.1.4 Compute $S_i(X, F)$ for $i > 3$

Similar computations, with increasing computational time, are performed to obtain the values of $S_i(X, F)$. We need to consider an ordered i-tuple $\langle x_0, x_1, x_2, ..., x_{i-1}\rangle$ of mutually distinct subsets of X, and find all the valid combination of ms. Then we can compute the sum of $\binom{n}{m_0} \cdot \binom{n-m_0}{m_1} \cdot \binom{n-m_0-m_1}{m_{01}} \cdots$. Intermediate computations of binomial coefficients are kept in a dynamic Pascal triangle.

Notice that the total number of the ms increases fairly quickly. For computing $S_6(X, F)$, we need to consider 63 ms, and for $S_7(X, F)$, then we will have to consider 127 ms. In general, we have to consider $(2^i - 1)$ ms to compute $S_i(X, F)$.

3.2 The m-approach to computing S_i

The counter example from previous section is based on a large but highly regular variant of Cayley graphs. The complexity of the algorithm that used in the previous section to compute S_i is $O((2^i)^n)$ at the worst case, which is quite intractable when i gets larger. In this section, we are going to discuss a more efficient algorithm that Liu has proposed in [5]. The following example illustrates how knowing and saving solutions for S_{i-1} can be used to accelerate the computation speed of solutions for S_i.

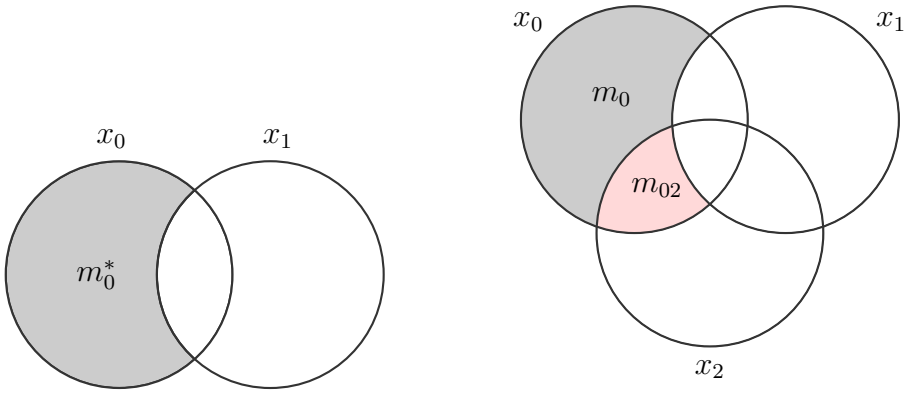
Let $m^* = \langle m_0^*, m_1^*, m_{01}^* \rangle$ be a valid solution for S_2, and we want to compute the solution for S_3. A valid solution $m = \langle m_0, m_1, m_2, m_{01}, m_{02}, m_{12}, m_{012} \rangle$ could be generated by reusing m^*, since

$$m_0{}^* = m_0 + m_{02}$$

$$m_1{}^* = m_1 + m_{12}$$

$$m_{01}{}^* = m_{01} + m_{012}$$

The figure below shows how the original m_0^* of S_2 is partitioned into the new m_0 and m_{02} of S_3, similar partitioning goes for $m_1{}^*$ and $m_{01}{}^*$.



(a) the original m_0^* of S_2 (b) the new m_0 and m_{02} of S_3

Figure 3.3: The representation of $m_0{}^* = m_0 + m_{02}$ in Venn diagram

In order to check the validity of the new ms, the following constraints can be used: $0 \leq m_0 \leq m_0{}^*$, $0 \leq m_1 \leq m_1{}^*$, and $0 \leq m_{01} \leq m_{01}{}^*$. At this point we have calculated $m_0, m_1, m_{01}, m_{02}, m_{12}, m_{012}$, the only thing left is m_2. Recall that $|x_2|$ should be in F, thus we can calculate m_2 directly: as $m_2 = z - m_{12} - m_{02} - m_{012}$ for some $z \in F$. We also need to check the symmetric difference relationships among the x_is. However, we only need to check $|x_0 \triangle x_2| \in F$ and $|x_1 \triangle x_2| \in F$.

Remark If m^* is a valid solution for S_i, and m is the corresponding valid solution for S_{i+1},

$$Y^* = \binom{X}{{m_0}^*}\binom{X-{m_0}^*}{{m_1}^*}\binom{X-{m_0}^*-{m_1}^*}{m_2^*}\binom{X-{m_0}^*-{m_1}^*-{m_2}^*}{{m_3}^*}\cdots$$

is the corresponding product of the binomial coefficients for m^*, and

$$Y = \binom{X}{m_0}\binom{X-m_0}{m_1}\binom{X-m_0-m_1}{m_2}\binom{X-m_0-m_1-m_2}{m_3}\cdots$$

is the corresponding product of the binomial coefficients for m, then

$$Y = Y^*\binom{{m_0}^*}{m_0}\binom{{m_1}^*}{m_1}\cdots\binom{{m_{01\cdots i}}^*}{m_{01\cdots i}}\binom{X-{m_0}^*-{m_1}^*-{m_{01}}^*-\cdots}{m_i}$$

Although we reuse the results from the computation of S_{i-1}, but the total number of m of computing the S_i is the same as the previous approach, which is $2^i - 1$.

Liu has showed us a simple example of how the approach works during a personal communication [17].

Example 3.2.1 *Find S_1, S_2, S_3, where $|X| = 5$, and $F = \{1, 2, 5\}$.*

1. Compute $S_1(X, F)$

The valid solution will be: $m_0 = 1$, $m_0 = 2$, or $m_0 = 5$.

Thus $S_1 = \binom{5}{1} + \binom{5}{2} + \binom{5}{5} = 16$

2. Compute $S_2(X, F)$ *We could reuse the solution from S_1.*

Step 1: In Table 3.1, we list all the possible ms without checking the symmetric difference.

$m_0{}^*$	m_0	m_{01}	choice for m_1
1	0	1	0, 1, 4
	1	0	1, 2, 5
2	0	2	0, 3
	1	1	0, 1, 4
	2	0	1, 2, 5
5	0	5	0
	1	4	1
	2	3	2
	3	2	0, 3
	4	1	0, 1, 4
	5	0	1, 2, 5

Table 3.1: compute ms before checking the symmetric difference. (m_1 = *some value in* $F - m_{01}$)

Step 2: We check the symmetric difference condition $|x_0 \triangle x_1| \in F$*, which is* $(m_0 + m_1) \in F$ *in terms of* m*. Table 3.2 shows the result.*

$m_0{}^*$	m_0	m_{01}	choice for m_1
1	0	1	1
	1	0	1
2	0	2	Nil
	1	1	0, 1, 4
	2	0	Nil
5	0	5	Nil
	1	4	1
	2	3	Nil
	3	2	Nil
	4	1	1
	5	0	Nil

Table 3.2: valid ms after checking $(m_0 + m_1) \in F$

Step 3: The last condition that we need to check is $m_0 + m_1 + m_{01} \leq |X|$*. Table*

3.3 shows the result.

${m_0}^*$	m_0	m_{01}	choice for m_1
1	0	1	1
	1	0	1
2	1	1	0, 1

Table 3.3: valid ms after checking $m_0 + m_1 + m_{01} \leq |X|$

Thus $S_2 = \binom{5}{1}\binom{4}{1} + \binom{5}{1}\binom{4}{1} + \binom{5}{1}\binom{4}{1} + \binom{5}{1}\binom{4}{1}\binom{3}{1} = 120.$

3. Compute $S_3(X, F)$

We could compute S_3 *by reuse the solution from* S_2*. The solution which we obtained from previous step are* $\{{m_0}^*, {m_1}^*, {m_{01}}^*\} = \{0, 1, 1\}, \{1, 0, 1\}, \{1, 1, 0\}, \{1, 1, 1\}$*.*

Step 1: In Table 3.4, we list all the possible m*s without checking the validation.*

$\{m_0{}^*, m_1{}^*, m_{01}{}^*\}$	$\{m_0, m_{02}\}$	$\{m_1, m_{12}\}$	$\{m_{01}, m_{012}\}$	choice for m_2
$\{0,1,1\}$	$\{0,0\}$	$\{0,1\}$	$\{0,1\}$	0, 3
			$\{1,0\}$	1, 4
		$\{1,0\}$	$\{0,1\}$	0, 1, 4
			$\{1,0\}$	1, 2, 5
$\{1,0,1\}$	$\{0,1\}$	$\{0,0\}$	$\{0,1\}$	0, 3
			$\{1,0\}$	1, 4
	$\{1,0\}$	$\{0,0\}$	$\{0,1\}$	0, 1, 4
			$\{1,0\}$	1, 2, 5
$\{1,1,0\}$	$\{0,1\}$	$\{0,1\}$	$\{0,0\}$	0, 3
		$\{1,0\}$	$\{0,0\}$	0, 1, 4
	$\{1,0\}$	$\{0,1\}$	$\{0,0\}$	0, 1, 4
		$\{1,0\}$	$\{0,0\}$	1, 2, 5
$\{1,1,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	2
			$\{1,0\}$	0, 3
		$\{1,0\}$	$\{0,1\}$	0, 3
			$\{1,0\}$	0, 1, 4
	$\{1,0\}$	$\{0,1\}$	$\{0,1\}$	0, 3
			$\{1,0\}$	0, 1, 4
		$\{1,0\}$	$\{0,1\}$	0, 1, 4
			$\{1,0\}$	1, 2, 5

Table 3.4: compute ms before checking the validation
($m_2 = some\ value\ in\ F - m_{02} - m_{12} - m_{012}$)

Step 2: After list all the possible ms. We could check the validation for each m. Notice that we do not need to check the symmetric difference between x_0 and x_1 whether it is in F, and we also do not need to check whether x_0 or x_1 is in F, since we have already checked them when we compute S_2. Therefore we only need to check $|x_0 \triangle x_2| \in F$, $|x_1 \triangle x_2| \in F$ and $|x_0 \cup x_1 \cup x_2| \leq X$ in the rest of the steps, details are omitted.

Chapter 4

Parallelizing the Computational Framework

As mentioned in the previous chapters, in order to find a better upper bound of c_t, we need to find a combination of (X, F) in which the so called Boolean graphs are generated for computing the Erdős's conjecture can get a better upper bound. Here, one combination of (X, F) is called a seed. There are several ways to improve the computational framework that is used in [5], which would allow us to explore larger value of ts or to enlarge the searching space for some relatively smaller value of t, so that we can quickly find a good seed that leads to a better upper bound. In [5], they suggest that a straightforward parallelization could be an efficient approach. Therefore, in this chapter, we will introduce different types of parallel computing and which one is more appropriate to apply to their computational framework.

4.1 Parallel Computing

A very basic and general interpretation of parallel computing is that it is a process of computation in which the problem is split into several sub-problems,

which are then solved concurrently, the solution of the sub-problems may or may not be combined depends on the requirements to the original problem.

There are many different ways to classify parallel and sequential computers and programs. However, the most widely used scheme is created by Michael J. Flynn in 1966, called Flynn's taxonomy. Flynn classified programs and computers according to two independent dimensions of ***Instruction*** and ***Data***. [8] The chart below shows the 4 possible classification defined by Flynn:

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

Table 4.1: Flynn's taxonomy

Single Instruction, Single Data (SISD):

This classification typically means those computers (with a single CPU or a single core Processing Unit) which can only run sequential programs. Only one instruction stream is passing to the CPU at any given clock cycle, and the data is stored in a single memory pool.

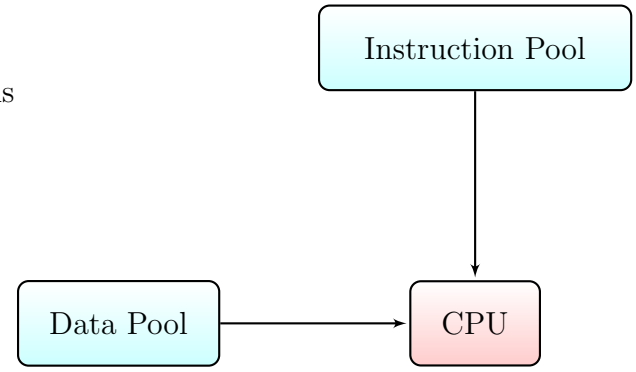


Figure 4.1: SISD

Single Instruction, Multiple Data (SIMD):

SIMD can be considered as a type of parallel computer. The same instruction is executed by all the CPUs or Processing Units during any given clock cycle, and each CPU can then operate on a different data sets. The execution is synchronous and deterministic.

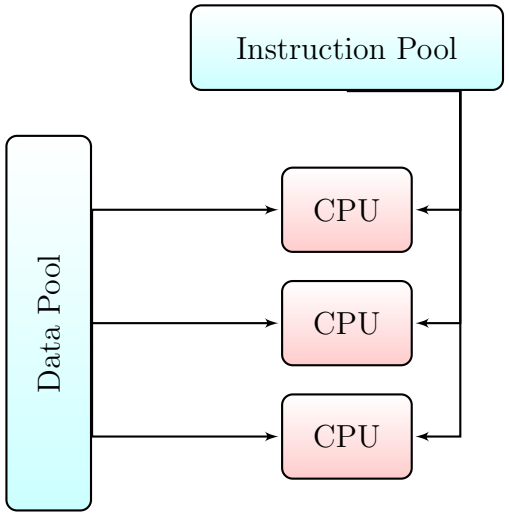


Figure 4.2: SIMD

Multiple Instruction, Single Data (MISD):

Only a few actual examples of this type of parallel computer have ever existed. Each CPU executes the data independently through different instruction streams, and only a single data stream is shared by these CPUs.

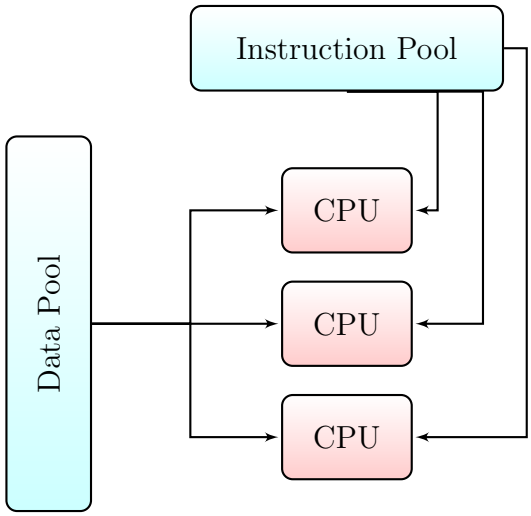


Figure 4.3: MISD

Multiple Instruction, Multiple Data (MIMD):

The most common type parallel computing we can see nowadays, most of the current supercomputers, workstation clusters, and multi-core PCs falls into this class. Every CPU can execute its own instruction stream, and each of them may have a distinct data set allocated for processing independently. Execution can be synchronous or asynchronous, deterministic or non-deterministic.

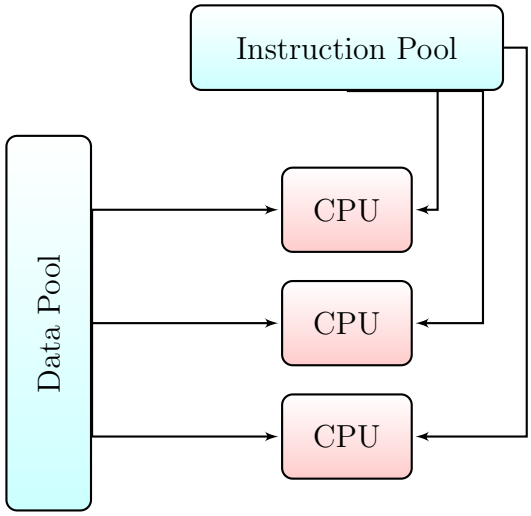


Figure 4.4: MIMD

To be more specific on **MIMD**, it can be further classified into two categories: **Shared Memory Model** and **Distributed Memory Model**. These two different memory architectures are based on the way how processors communicate with each other.

4.1.1 Parallel Computer Memory Architectures

Shared Memory Model

There are couple general characteristics for shared memory parallel computers to have in common. The most important one is that a single address space is shared by all processors, and these processors have the same ability to access all memory. However, processors can operate independently while sharing the same memory spaces, and any changes in a memory location by one processor can be seen to all other processors.

If we put memory access time into account, then we can further divide the model into two main classes: Uniform Memory Access (UMA) and Non-Uniform Memory Access (NUMA).

- UMA: This class architecture are commonly represented by Symmetric Multi-Processing (SMP) systems nowadays, a single copy of the operating system is in charge of all the processors, and these processors are connected to a shared memory. More importantly as all processors are identical and each of them has equal access time to the shared memory. We can consider a multi-core processor as a SMP system, since all the cores can be considered as separate processing units.

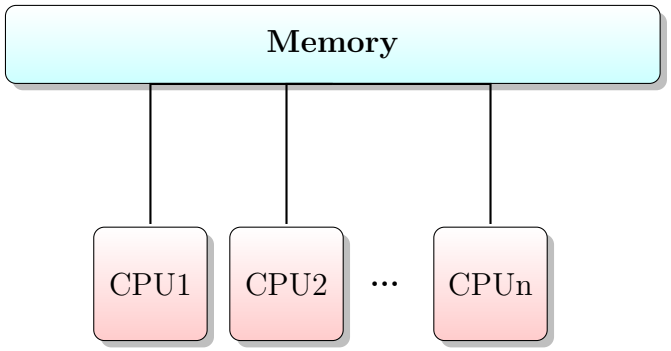


Figure 4.5: Shared Memory Model (**UMA**)

- NUMA: As opposite to UMA, the memory access time are not equal, and it depends on the memory location relative to a processor. Each processor has its own local memory and it is shared to other processors through a interconnection network. In other words, the memory is logically shared but physically distributed, also known as Distributed-Shared Memory

Model. The access time from a processor to its local memory is always faster than to non-local memory.

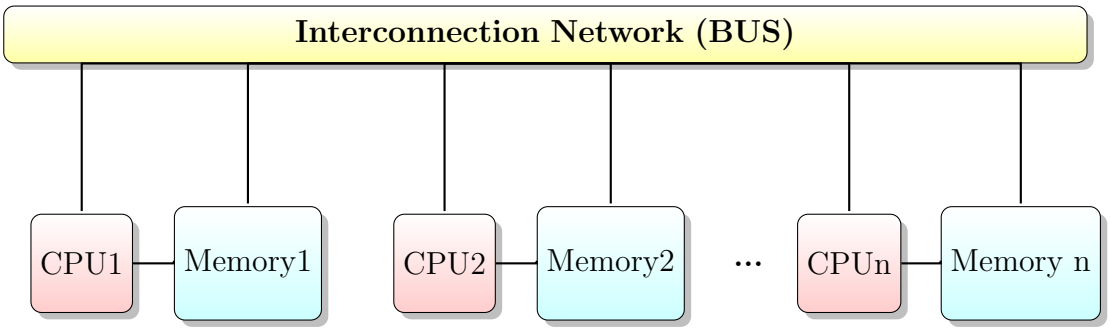


Figure 4.6: Shared Memory Model (**NUMA**)

It is relatively easy and user-friendly to program on shared memory model machines since processors are all connected to a large shared memory which allows the programmer to handle the memory as a global address space, and the communication between processors can be both fast and uniform due to the proximity of memory to processors. However, shared memory model does have some disadvantages need to be aware of. The main disadvantage is the scalability issue, because of the system benefits from the fast communication between memory to CPUs, if we add more CPUs to the system, then the traffic on the memory-CPU path increases geometrically, therefore the memory-CPU connection would eventually become a bottleneck. Another disadvantage is the programmer needs to take care of the synchronization of the access to the memory in order to prevent the occurrence of race condition. Third, overheads may incurred with maintaining cache coherency, here cache coherency is the

discipline that ensures the changes in the values of shared data are propagated throughout the whole system.

Distributed Memory Model

The memory architecture in this model looks quite similar to **NUMA** except each processor has its own private memory space, the memory addresses in one processor do not map to another, therefore the memory is not only physically distributed, but also logically distributed. In a distributed system, a processor with its own memory can be viewed as a complete computer system or a node, so that we also call it multi-computer model compare to the previous mentioned **UMA** model which usually referred to as multi-processor model, and these nodes require a communication network (e.g. Ethernet or Internet) to interact with each other by message passing.

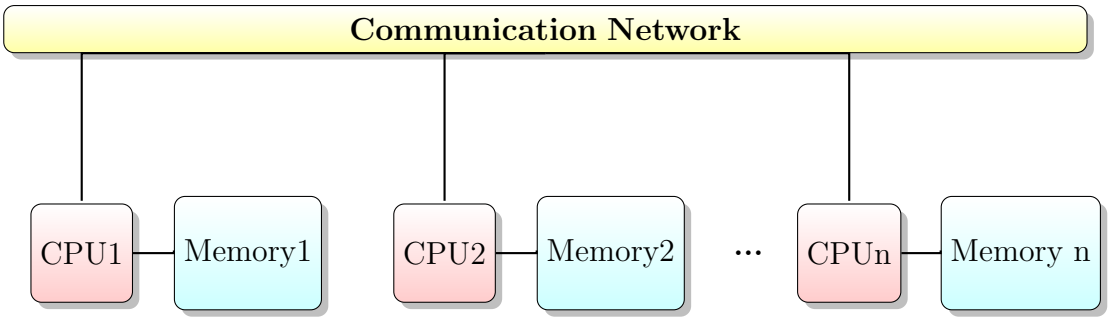


Figure 4.7: Distributed Memory Model

There are several benefits of this model. First, each processor is allowed to utilize its own memory without interference from other processors which excludes race conditions and bus contentions. Second, the memory scalability

is excellent in Distributed Memory Model, the size of memory increases with the number of processors proportionally, the scalability is now constrained by the capacity of communication network. Third, because of each processor is in charge of its own memory, so that cache coherency is no longer a problem. The major drawback in the Distributed Memory Model is that the data communication between processors is more difficult compare to the data communication in Shared Memory Model, processors must exchange messages with each other if one requires data from another's memory, then two potential overheads may occur: the time of constructing and sending a message, and the time of receiving and dealing with the message (in order to do so, a processor needs to interrupt its current computation if it's busy).

4.1.2 Parallel Programming Models

In this section, we will briefly describe some of the commonly used parallel programming models [1, 9], and mainly focus on the model that we use to parallelize the computational framework in the next section. Most of these models can be divided into two classes: **Problem Decomposition** and **Process Interaction**.

Problem Decomposition

Parallel program runs the processes simultaneously, and Problem Decomposition is the way how these processes are constructed.

- Task parallelism: This model focuses on processes or threads execution, these processes are usually functional distinct, each processing unit executes a different thread or process, it refers to one or more independent

tasks running simultaneously. The benefit it delivers efficient and scalable system resource usage.

- Data parallelism: This model is usually executing under SIMD (single instruction, multiple data) memory architecture which we just mentioned in previous section, it focuses on data distribution through different processing units, each processing unit executes the same task on different set of data.

Process Interaction

- Message passing: Message passing is widely used in Distributed Memory systems where parallel tasks exchange data by passing messages from one to another. Communications can be either synchronous or asynchronous.

- Shared Memory Model: In the aspect of programming model, the tasks in this model share a common address space, which they read and write asynchronously. An advantage of this model from the programmer's point of view is that the data is commonly shared, so that no communication of data needs to be explicitly specified between tasks. However, control the access to the shared memory is left to the programmer to take care of, various mechanisms such as locks or semaphores may be applied.

We decide to use Shared Memory Model (for both memory architecture and programming models) to parallelize the computational framework for several reasons. First, in terms of scalability requirement, the original serial program still can be handled by a single multi-processor machine, so distributed computational power is not necessary at this stage. Second, in order to make

the comparison to be fair, running both serial and parallel version on the same machine is relatively more convincing. Moreover, from the programming perspective, it is user-friendly to program on Shared Memory Model. To be more specific, Multi-Threading is going to be the parallel programming model.

4.1.3 Multi-threading

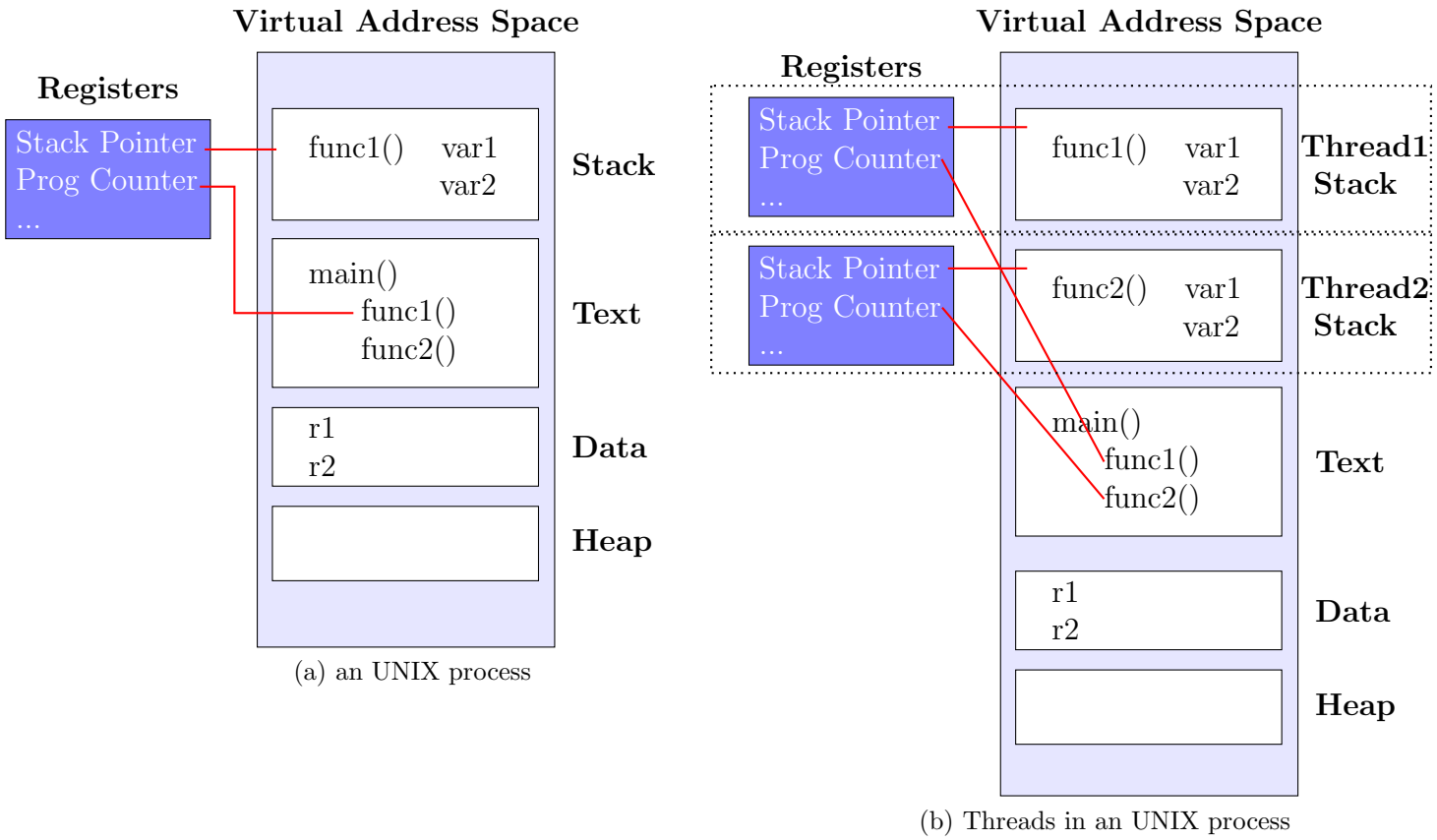
Thread

Before thread had been developed, process was the only way to deliver multiple tasks to the processor. A thread is an independent stream of instructions that can be scheduled to run by the operating system, it allows multiple subtasks exist as individual streams of control within the same process. In some operating system, thread can also be considered as light weight process.

The threads model takes a process and divides it into two parts [18, 2]:

- One includes resources used across the whole program, such as program instructions and global data. This part is still referred to as the process.
- The other contains information about the execution state, such as a program counter and a stack, so this part can be considered as a thread.

Multiple threads can exist within the same process and share resources such as memory, however different processes do not share these resources. Also, the virtual address space of a program is usually partitioned into 4 segments: stack, text(Instructions), data, and heap. Threads share all segments except the stack, so each thread has its own instruction pointer, set of registers and stack of memory. The following figure shows a process with and without threads.



(a) an UNIX process

(b) Threads in an UNIX process

Figure 4.8: Example of a process and threads within the process

Multi-threading vs Multi-processing

Multi-threading usually refers to a program with multiple threads running within a process, while multi-processing refers to program managed across multiple Operating System level processes. In most operating system, threads involve less overhead compare to processes, as a result the context switching and synchronization costs are lower, so using multi-threading is more efficient than multi-processing. However, multi-processing is more reliable in the long

run because they can be invoked and killed independently, and errors in one process cannot bring down another since they are insulated by the operating system [3]. Therefore, whether to use mutli-threading or multi-processing depends on the amount of shared data and the amount of context switching once the program is parallelized. Although the original serial version framework will not have many context switching if we parallelize it, we still want to use the multi-threading model due its efficiency. In fact, the computation is quite independent between parallelized tasks where the detail will be described later. The issue of reliability can be overcome with good programming style.

Pthreads

We are going to use Pthread to parallelize the computational framework. Pthreads is a standardized model for dividing a program into subtasks whose execution can be interleaved or run in parallel. The "P" in Pthreads comes from **POSIX**(Portable Operating System Interface), the family of **IEEE** operating system interface standards where Pthreads is defined. One reason we use Pthreads instead of other API is: Pthreads are defined as a set of C language programming types and procedure calls, and its libraries are a standards based thread API for C/C++. The program of the original framework was implemented in C++, so that re-writing the entire program for parallelization is not necessary if we use Pthreads. [18, 2]

4.2 Parallelizing Application

In this section we are going to apply the parallel model onto the original computational framework, and the comparison of the testing result between serial framework and parallel framework. First of all, we need to modify the original computational framework, so that the problem can be broken into discrete sub-problems, and these sub-problems can then be distributed into each individual threads for computing. There are two basic ways to do so: **Domain decomposition** and **Functional decomposition**. These two method can refer to two previously mentioned parallel programming model **Data parallelism** and **Task parallelism** respectively. From programming perspective, we decide to use functional decomposition because the problem is decomposed into smaller components, testing and debugging would be easier to handle.

4.2.1 m-approach decomposition

Recall the **m-approach** from Section 3.2. Denote w as the sum of all ms of $S_i(X, F)$, so that $w \leq |X|$. We can decompose the problem by fixing the value of w, and we know that $w \in [0, |X|]$, for each value of w, we check the validity of ms for S_i by using the corresponding conditions that mentioned in m-approach, then we can simply combine all the solutions by adding them up. In other words, we are decomposing the summation in the following formula:

$$S_i(X, F) = \sum_{valid\, m's} Y^* \cdot \binom{m_0{}^*}{m_0}\binom{m_1{}^*}{m_1}\cdots\binom{|X|-m_0{}^*-m_1{}^*-\cdots}{m_{i-1}}, i \geq 2$$

Where Y^* refers to Section 3.2. Note that, when $w = 0$, there will never be a

valid solution for ms, so we can ignore this case and the range of w would be $[1, |X|]$. The following example shows how the decomposition works in detail.

Example 4.2.1 *Find S_1, S_2, where $|X| = 3$, and $F = \{1, 2\}$.*

1. Compute $S_1(X, F)$

The valid solution will be: $m_0 = 1$ or $m_0 = 2$.

Thus $S_1 = \binom{5}{1} + \binom{5}{2} = 15$

2. Compute $S_2(X, F)$ *Reuse the solution from S_1.*

Step 1: In Table 4.2, we list all the possible ms base on the value of w without validating them.

$m_0{}^*$	m_0	m_{01}	w	m_1	thread #
1	0	1	1	0	thread 1
			2	1	thread 2
			3	2	thread 3
	1	0	1	0	thread 1
			2	1	thread 2
			3	2	thread 3
2	0	2	1	N/A	thread 1
			2	0	thread 2
			3	1	thread 3
	1	1	1	N/A	thread 1
			2	0	thread 2
			3	1	thread 3
	2	0	1	N/A	thread 1
			2	0	thread 2
			3	1	thread 3

Table 4.2: Compute ms without validating them. ($m_1 = w - m_0 - m_{01}$)

Note that each task of computing m_1 is assigned to the corresponding thread according to the value of w.

Step 2: We then check the condition $|x_1| \in F$, which is $m_{01} + m_1 \in F$ in terms of m. Table 4.3 shows the result.

$m_0{}^*$	m_0	m_{01}	w	m_1	thread #
1	0	1	1	0	thread 1
			2	1	thread 2
			3	~~2~~	thread 3
	1	0	1	~~$\emptyset$~~	thread 1
			2	1	thread 2
			3	2	thread 3
2	0	2	1	N/A	thread 1
			2	0	thread 2
			3	~~1~~	thread 3
	1	1	1	N/A	thread 1
			2	0	thread 2
			3	1	thread 3
	2	0	1	N/A	thread 1
			2	~~$\emptyset$~~	thread 2
			3	1	thread 3

Table 4.3: Eliminating invalid ms. ($m_{01} + m_1 \in F$)

Step 3: Now we check the symmetric difference condition $|x_0 \triangle x_1| \in F$, which is $m_0 + m_1 \in F$ in terms of m. Result shown in Table 4.4. Notice that we don't need to check condition $m_0 + m_1 + m_{01} \leq |X|$ in the end because it is the same as our assumption that $w \leq |X|$. The final result is shown in Table 4.5

$m_0{}^*$	m_0	m_{01}	w	m_1	thread #
1	0	1	1	$\emptyset$	thread 1
			2	1	thread 2
			3	N/A	thread 3
	1	0	1	N/A	thread 1
			2	1	thread 2
			3	~~2~~	thread 3
2	0	2	1	N/A	thread 1
			2	$\emptyset$	thread 2
			3	N/A	thread 3
	1	1	1	N/A	thread 1
			2	0	thread 2
			3	1	thread 3
	2	0	1	N/A	thread 1
			2	N/A	thread 2
			3	~~1~~	thread 3

Table 4.4: Eliminating invalid ms. $(m_0 + m_1 \in F)$

$m_0{}^*$	m_0	m_{01}	w	m_1	thread #
1	0	1	2	1	thread 2
	1	0	2	1	thread 2
2	1	1	2	0	thread 2
			3	1	thread 3

Table 4.5: Result after eliminating invalid ms.

Thus $S_2 = \binom{5}{1}\binom{4}{1} + \binom{5}{1}\binom{4}{1} + \binom{5}{1}\binom{4}{1} + \binom{5}{1}\binom{4}{1}\binom{3}{1} = 120.$

The following figure graphically shows an example of how the parallelization works with $|X| = 7$ and $t = 9$, so there will be seven threads in each task of computing the S_i for $i \geq 2$, computing S_1 is straightforward, therefore parallelization is ignored in S_1.

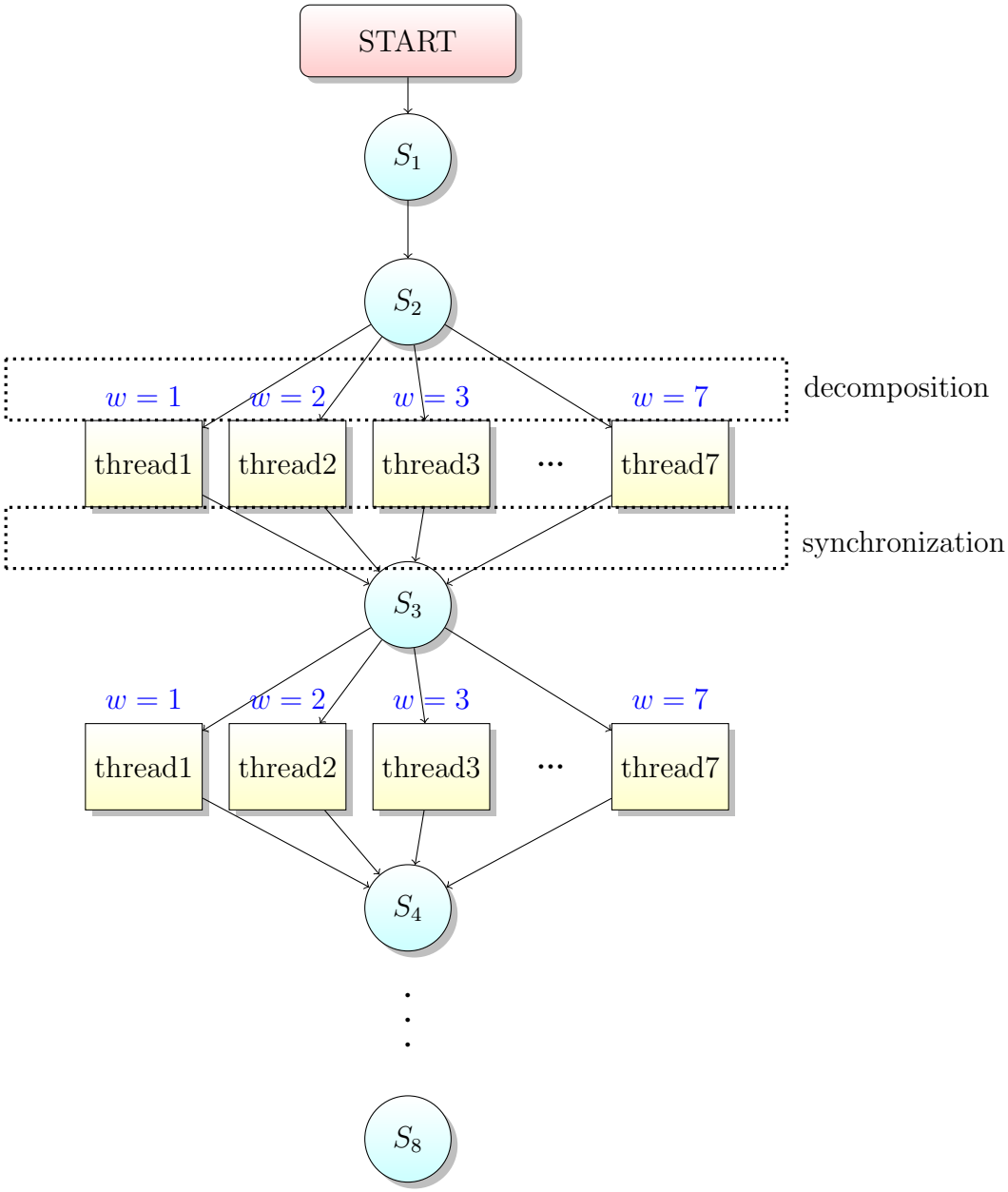


Figure 4.9: Graphical interpretation of m-approach decomposition ($|X| = 7, t = 9$)

4.3 Performance Analysis

In this section, we present the testing results for both the serial and parallel computational framework, and we discuss and compare them in terms of performance.

4.3.1 Testing Environment

Both the serial and the parallel testing program are written in C++, and compiled by the GNU C++ compiler (g++ 4.2.1). The testing machine we used was the department server **AdvOL3** which consists of 16 Dual Core AMD Opteron(tm) Processor 885 and 64 GB of memory.

4.3.2 Testing Results

We test both versions of the program by performing an brute-force search on (X, F) for different combination of X and t. The following table shows the total elapsed time of running two versions and the percentage of performance changing from serial to parallel.

	Serial(ms)	Parallel(ms)	Performance gain/loss
X = 5, t = 7	7.287E+03	7.942E+03	-9%
X = 6, t = 7	1.424E+05	1.544E+05	-8%
X = 6, t = 8	3.842E+06	3.378E+06	**12%**
X = 7, t = 6	6.317E+04	5.476E+04	**13%**
X = 7, t = 7	2.329E+06	2.464E+06	-6%
X = 8, t = 7	3.285E+07	3.369E+07	-3%
X = 9, t = 7	4.035E+08	4.136E+08	-2%

Table 4.6: Performance comparison between serial and parallel computational framework

When we first look at the results, we notice an unexpected phenomenon, almost all the parallel version programs runs slightly slower than the serial version. One possible problem could be the data communication overheads within the computation of the parallel version. Here, we need to explain a little more detail on the implementation of the parallel program. As mentioned in m-approach, for computing S_i, we need to reuse the computational result from S_{i-1}, and in order to do that, we save the result from S_{i-1} by writing it to a file, then this file can be used to compute S_i. For example, refer back to figure 4.9, at the synchronization stage after computing S_2, each thread saves the intermediate result to a temporary file, and then merge these files together (to avoid race conditions). The write file time occupies more than 85% of the total computing time for each S_i as shown in table 4.7, so if all the threads write its own file at the same time, the data transfer bandwidth would increase thread-number times, this could significantly slow down the performance once the bandwidth reaches its limit.

	Write file time proportion
X = 5, t = 7	85.75%
X = 6, t = 7	87.73%
X = 7, t = 7	86.33%

Table 4.7: The proportion of the write file time in parallel framework

However this is not the case because if communication overheads is the problem, then all the parallel results should be slower than the serial version, but we still have two results which show the parallel version is faster. Therefore, further testing is required to explore what the real problem is.

4.3.3 Further testing

To further test the performance, we are going to investigate more detail on thread-wise performance, recall that a combination of (X, F) is called a seed, and it can be represented as a characteristic vector of F as a subset of $\{1, 2, ..., X\}$, e.g. a seed $(X, F) = (10, \{1, 3, 5, 7, 9\})$ can be represented as $(X, F) = [1010101010]$ where the size of the vector is set to be X. We test the thread-wise performance by measuring the solution file size that is created from each thread for different seeds, so that we will know the computational load on each thread. The heavier the load is, the more computing time the thread requires. The following diagrams have shown examples of seeds that result in different situation of computational load on the threads.

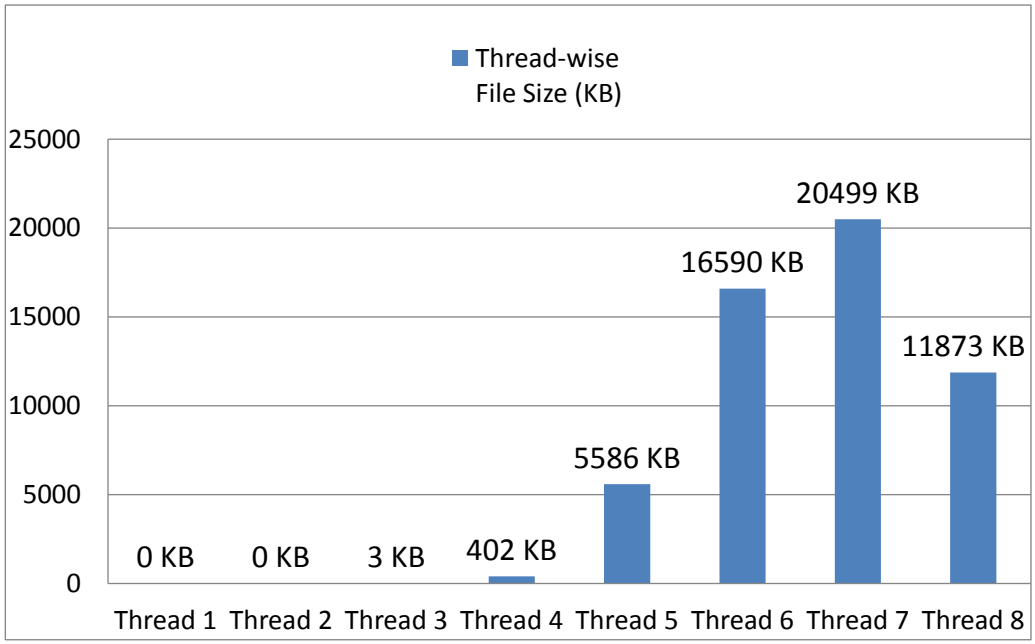


Figure 4.10: thread-wise file size of computing S_6 for seed $(X, F) = 11110011$

Figure 4.10 shows the size of the file generated by each thread when computing S_6 of seed $(X, F) = 11110011$, the seed typically reveals a situation where the computational load on each thread is relatively evenly distributed, in other words, the seed is parallelization friendly. Obviously, the more this type of seeds the parallel program has, the faster it runs. We also find that if the computation on one S_i within a parallelization friendly seed has this characteristic, it's very likely that the other S_is would perform similarly.

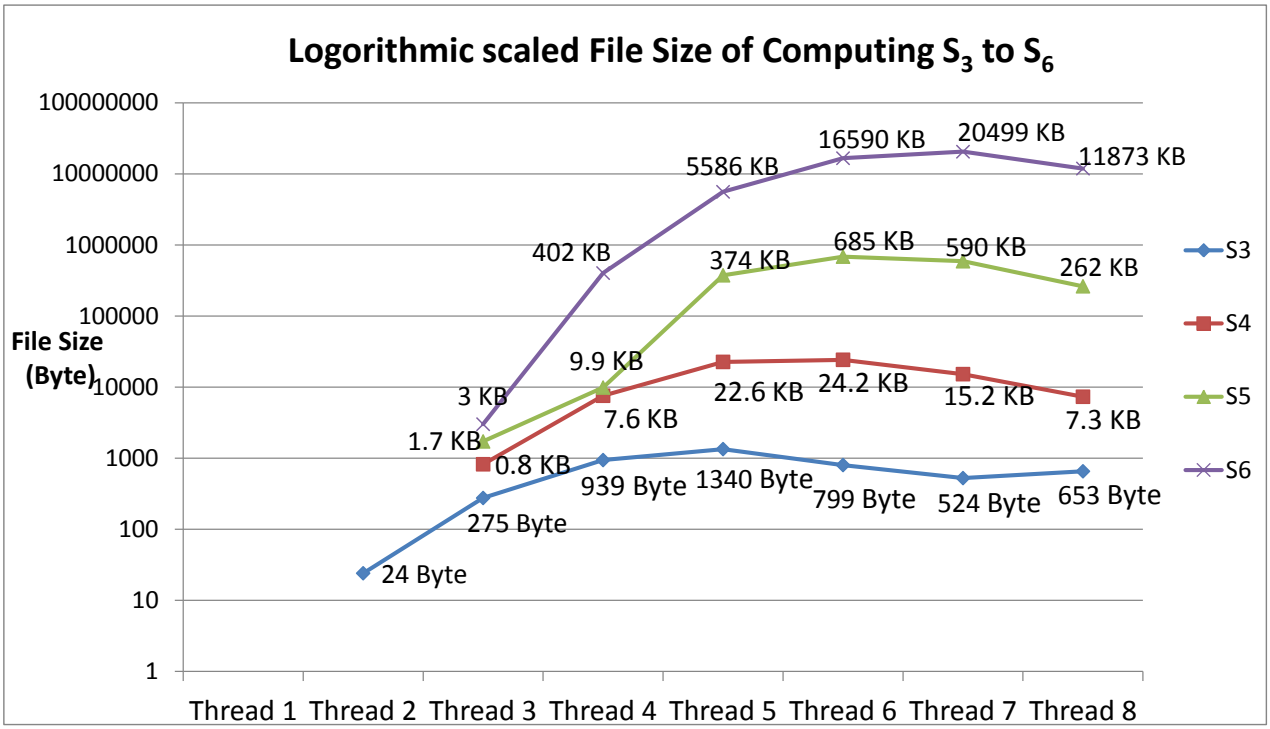


Figure 4.11: thread-wise file size of computing S_3 to S_6 for seed $(X, F) = 11110011$

As shown in Figure 4.11, although the file size increases very quickly from

computing S_3 to S_6, but the results still follow a flat smooth pattern which reveals the consistency of the relatively evenly distributed computational load on each thread throughout all the S_is.

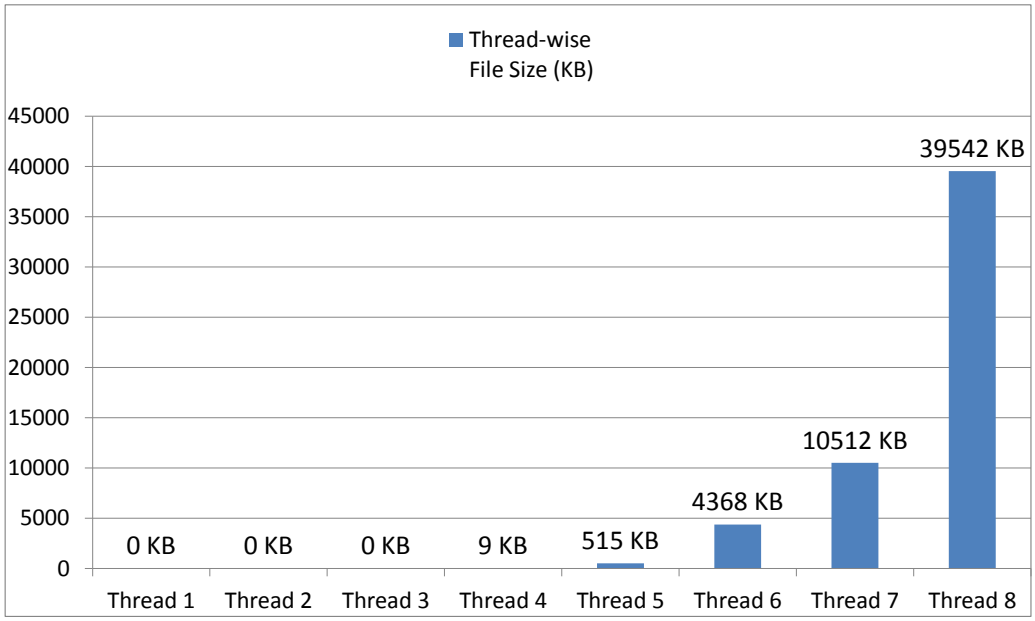


Figure 4.12: thread-wise file size of computing S_6 for seed $(X, F) = 11011011$

The figure above shows another situation where the computational load is slightly biased, which means fewer threads share the total computational load. We call this situation as parallelization unfriendly. In this particular example, the file size of last three threads are dominating the computational load, and the benefits obtained from multi-threading would probably be nullified by the communication overheads we just mentioned before. Therefore, the parallel program will not gain much performance from this type of seeds.

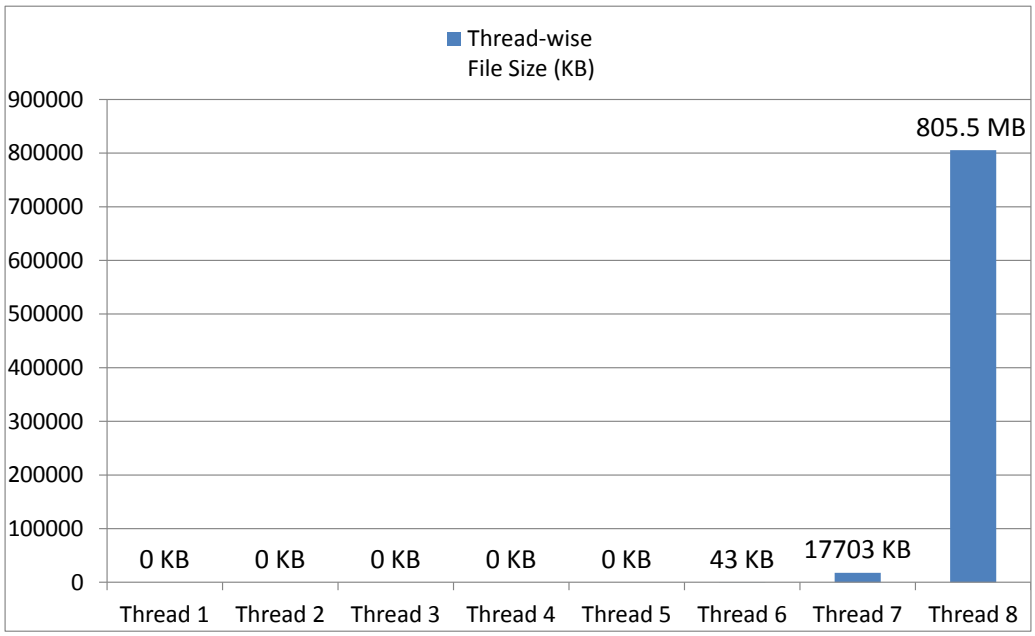


Figure 4.13: thread-wise file size of computing S_6 for seed $(X, F) = 10111110$

The last situation is where the computational load is extremely biased as shown in Figure 4.13, almost all the computational load is relied on a single thread (thread 8 in this case). This situation could be a disaster to parallelization, there's no performance gain at all in this type of seed, and the comparison result in the table below is consistent with what we just analyzed for these three types of seeds.

Seed	Type	Serial(ms)	Parallel(ms)	Performance gain/loss
11110011	relatively evenly distributed	7791	3613	53.63%
11011011	slightly biased	7970	6761	15.17%
10111110	extremely biased	116359	130006	-11.73%

Table 4.8: Performance comparison between serial and parallel version of different types of seed on S_6

Last but not least, recall the comparison results in Figure 4.6, we notice that if we fix the value of $t = 7$, the performance slowly increases (-9% to -2%) as X increases. Filtered results are shown below. This indicates that if we keep increasing the search space X for our brute force search, the performance of parallel framework would eventually beats the serial version. However, as we testing to $X = 8, t = 8$, the temporary solution file created by a thread can be as large as 1 TB which exceed our server storage limit. Therefore, further testing is unavailable at this stage for X is larger than 8.

	Serial(ms)	Parallel(ms)	Performance gain/loss
X = 5, t = 7	7.287E+03	7.942E+03	-9%
X = 6, t = 7	1.424E+05	1.544E+05	-8%
X = 7, t = 7	2.329E+06	2.464E+06	-6%
X = 8, t = 7	3.285E+07	3.369E+07	-3%
X = 9, t = 7	4.035E+08	4.136E+08	-2%

Table 4.9: Performance comparison between serial and parallel computational framework with t fixed

Chapter 5

Conclusion

We have suggested and presented a functional decomposition method to parallelize the original computational framework [5] for computing the ratio of monochromatic t-cliques and the number of all t-subsets for a specific Cayley graph determined by a pair (X, F), and we also numerically tested the performance of this parallelized framework. These numerical tests were conducted by brute force search on several combinations of X and t. Although the performance of the parallelized framework showed some undesirable results, but we did address the problem throughout the performance analysis.

The numerical experiments demonstrated that not only the communication overheads can slow down the parallelization but also the performance is seed dependent. A relatively evenly distributed seed is parallelization friendly, but slightly and extremely biased seeds can really hurt the overall performance if the number of these parallelization unfriendly seeds dominates the searching space.

In the future work, one way to improve the performance is to parallelize the original computational framework by domain decomposition (data paral-

lelism). In other words, parallelizing the seeds, each thread would obtain a seed for a complete serial processing, so that seed dependency problem can be avoided. In addition to the domain decomposition, a pre-processing procedure could be seed filtering – to eliminate parallel unfriendly seeds. Although we can't figure out what type a seed is without testing it at this time, but one could further investigate the method to determine the seed type in the future. Another way is to refine the original computational framework by using pipelining, passing the intermediate result directly instead of saving them into a file, which could significantly increase the performance since the write file time occupies more than 85% of the total computing time in our numerical experiments.

Bibliography

[1] B. Barney: Introduction to Parallel Computing. https://computing.llnl.gov/tutorials/parallel_comp/.

[2] B. Barney: POSIX Threads Programming. https://computing.llnl.gov/tutorials/pthreads/ (2012)

[3] P. Bridger: C++ Multithreading Tutorial. http://www.paulbridger.com/node/17/ (2005)

[4] D. Colon: On the Ramsey multiplicity of complete graphs. *Combinatorica*, Vol. 32 (2012), 171-186

[5] A. Deza, F. Franek, M.J. Liu: On Erdős' conjecture for multiplicities of complete sub-graphs. *Journal of Discrete Algorithms*, (Revised March 2012).

[6] P. Erdős: On the number of complete subgraphs contained in certain graphs, *Publ. Math. Inst. Hung. Acad. Sci., VII, ser. A* 3 (1962), 459-464

[7] P. Erdős and J. W. Moon: On subgraphs on the complete bipartite graph, *Canad. Math. Bull.*, 7 (1964), 35-39

[8] M.J. Flynn: Very high-speed computing system. *Proc. IEEE.* 54(1966) 1901-9

[9] I. Foster: Designing and Building Parallel Programs. `http://www.mcs.anl.gov/~itf/dbpp/text/node9.html#SECTION02230000000000000000` (1995)

[10] F. Franek, V. Rödl: Disproving Erdős's conjecture on multiplicities of complete subgraphs using computer. *Technical Report*, No. 88-11 (1988)

[11] F. Franek, V. Rödl: Ramsey problem on Multiplicities of Complete Subgraphs in Nearly Quasirandom Graphs. *Graphs and Combinatorics*, Vol. 8 (1992), 299-308

[12] F. Franek, V. Rödl: 2-Colorings of complete graphs with a small number of monochromatic K_4 subgraphs. *Discrete Mathmatics*, Vol. 114 (1993), 199-203

[13] F. Franek, V. Rödl: A note on Erdős's conjecture on multiplicities of complete subgraphs - lower upper bounds for cliques of size 6 *Combinatorica*, No. 3, Vol. 22, (2002), 451-454

[14] G. Giraud: Sur le problème de Goodman pour les quadrangles et la majoration des nombres de Ramsey *J. Combin. Theory Ser.* B 30 (1979), 237-253

[15] A. W. Goodman: On sets of acquaintances and strangers at any party, *Amer. Math. Monthly*, Vol. 66 (1959), 778-783

[16] C. Jagger, P. Šťovíček, A. Thomason: Multiplicities of subgraphs. *Combinatorica*, Vol. 16, (1996), 123-141

[17] M.J. Liu: Personal communication. 2012

[18] B. Nichols, D. Buttlar, J.P. Farell: Pthreads Programming. *O'Reilly & Associates*, 1 (1996), 1-26

[19] A. F. Sidorenko: Cycles in Graphs and Functional inequalities, *Mathematical Notes*, Vol. 46 (1989), 877-882

[20] A. Thomason: A disproof of a conjecture of Erdős in Ramsey theory, *J. London Math.*, Vol. 39, No. 2 (1989), 246-255

[21] A. Thomason: Graph products and monochromatic multiplicities. *Combinatorica*, 17 (1997), 125-134