k 元 n 维冒泡排序网络的子网排除

杨玉星[1,2] 邱亚娜[1]

(河南师范大学数学与信息科学学院 新乡 453007)[1]

(河南师范大学大数据统计分析与优化控制河南省工程实验室 新乡 453007)[2]

摘 要 在并行计算机系统中,元器件和线路故障普遍存在,而系统的容错能力可以通过其底层基础网络的拓扑性质衡量。为了精确度量以 k 元 n 维冒泡排序网络为底层拓扑结构的并行计算机系统的容错能力,结合其层次结构和子网划分特征,分别提出了节点故障模型和线路故障模型下攻击该网络中所有 $k-m$ 元 $n-m$ 维冒泡排序子网络的算法,确定了需要攻击的最优节点集合和最优线路集合。根据算法可得:当 $2\leqslant k\leqslant n-2$,$m\leqslant k-1$ 时,攻击 k 元 n 维冒泡排序网络中所有的 $k-m$ 元 $n-m$ 维冒泡排序子网络,在节点故障模型下需要攻击至少 $C_n^m m!$ 个节点,在边故障模型下需要攻击至少 $C_n^m m!$ 条线路。

关键词 并行计算机,高性能互连网络,k 元 n 维冒泡排序网络,容错,子网排除

中图法分类号 TP393.02 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.039

Sub-network Preclusion in (n,k)-bubble-sort Networks

YANG Yu-xing[1,2] QIU Ya-na[1]

(School of Mathematics and Information Science, Henan Normal University, Xinxiang 453007, China)[1]

(Henan Engineering Laboratory for Big Data Statistical Analysis and Optimal Control, Henan Normal University, Xinxiang 453007, China)[2]

Abstract In the real parallel computer systems, faults of processors and links are inevitable, and the fault tolerance ability of a system can be evaluated by the performance of its interconnection network. In order to measure the fault tolerance abilities of the parallel computers which take (n,k)-bubble-sort networks as underlying topologies, combining the hierarchical structures and sub-network partitions' characters of (n,k)-bubble-sort networks, an algorithm of the problem that destroy all the $(n-m,k-m)$-bubble-sort networks in the (n,k)-bubble-sort network under the node fault model and the link fault model was proposed, and the optimal attacking nodes set and the optimal attacking links set were obtained. According to the algorithm, to destroy all the $(n-m,k-m)$-bubble-sort networks in the (n,k)-bubble-sort network, at least $C_n^m m!$ nodes or $C_n^m m!$ links ought to be faulty under the node fault model and the link fault model, respectively, where $2\leqslant k\leqslant n-2$ and $m\leqslant k-1$.

Keywords Parallel computer, High-performance interconnection network, (n,k)-bubble-sort network, Fault tolerance, Sub-network preclusion

1 引言

称并行计算机系统中处理器等硬件之间的连接模式为该系统的基础网络。基础网络的性质决定了以其作为底层拓扑而搭建的并行计算机系统的性能。

在实际应用中,并行计算机系统的处理器和通信线路难免因硬件老化、外力破坏、病毒入侵等而产生故障。这些故障反映在并行计算机系统的基础网络中,从而导致网络的节点和线路产生故障。故障发生时,若网络中依然存在既定规模的健康子网,则故障的系统依然可以正常执行相应子系统的功能。受此启发,Becker 和 Simon 在第 27 届国际计算机科学基础大会(FOCS 1986)上提出一个问题:当一定数目的节点或边发生故障时,该网络遭到破坏的最大维数能达到多少?他们确定了一个对手若要破坏 n 维超立方网络中所有的 $(n-m)$ 维子超立方所需要破坏的最少节点数和最少线路数。时隔十年,Latifi[1] 提出了一个类似的问题(子网排除问题):在一些元件发生故障时,网络中还将有多大规模的健康子网?随后,Latifi,Saberinia 和 Wu[2] 确定了 n 维 Star 网络中使得所有 $n-1$ 维子网络都遭到破坏的最少线路数,以及使得所有 $n-2$ 维子网络都遭到破坏的线路数上界。Walker 和 Latifi[3] 于 2010 年优化了文献[2]中的上界值。Wang 和 Yang[4] 于 2012 年探索了冒泡排序网络中的子网排除问题,采用构造子网节点排除割和子网边排除割的方法,得出了攻击 n 维冒泡排序网络中的所有 $n-1$ 维子网所需攻击的最少节点数和最少线路数。笔者[5] 于 2013 年研究了 k 元 n 立方网络中的长度为 k 的圈的排除问题,构造了当 k 为奇数时,k 元 n 立方网络的 k 圈排除的一种递归算法。Yang 等[6] 于 2015 年采用构造映射的方案,确定了攻击 n 维冒泡排序网络中的所有 $n-2$ 维子网所需攻击的最少节点数和最少线路数。自 2012 年起,诸如 k 元 n 立方网络[7]、Arrangement 网络[8-9]、Path-Star

到稿日期:2017-04-19 返修日期:2017-07-20 本文受国家自然科学基金(U1304601)资助。

杨玉星(1981-),男,博士,副教授,CCF 会员,主要研究方向为图与组合网络优化;邱亚娜(1990-),女,硕士生,主要研究方向为组合网络优化。

Cayley 网络[10]等二参数网络的子网排除问题引起了广泛关注。

k 元 n 维冒泡排序网络[11-12]是冒泡排序网络的重要推广，它保留了 n 维冒泡排序网络的层次性、正则性，比 n 维冒泡排序网络更易于控制节点的个数且更为灵活与实用。本文将研究 $2\leqslant k\leqslant n-2$ 时 k 元 n 维冒泡排序网络的子网排除问题，以确定在破坏 k 元 n 维冒泡排序网络的子网排除问题时节点和线路的最佳攻击方案。下文未定义而直接使用的图论记号与术语请参见文献[13]。

2　问题描述

2.1　k 元 n 维冒泡排序网络

对于任意的整数 $t\geqslant 1$，记 $I_t=\{1,2,\cdots,t\}$。

设 $n\geqslant 1$，$k\in I_n$，k 元 n 维冒泡排序网络 $B_{n,k}$ 是一个具有 $C_n^k k!$ 个形如 $x=x_1x_2\cdots x_k$ 的节点的无向图。其中，对于任意的 $i\in I_k$，均有 $x_i\in I_n$，且对于任意的 $j\neq i$，均有 $x_j\neq x_i$。

$B_{n,k}$ 中两个不同的节点 $x=x_1x_2\cdots x_k$ 与 $y=y_1y_2\cdots y_k$ 相邻当且仅当满足以下两个条件之一即可：

(1) $y_1\in I_n\backslash\{x_j\mid j\in I_k\}$，并且对任意的 $i\in I_k\backslash\{1\}$ 均有 $y_i=x_i$ 成立；

(2) 存在一个整数 $i\in I_k\backslash\{1\}$ 使得 $y_i=x_{i-1}$，$x_i=y_{i-1}$ 并且对任意的 $j\in I_k\backslash\{i-1,i\}$ 均有 $y_j=x_j$ 成立。

当节点 x 和 y 因满足条件(1)而相邻时，称连接它们的边为 1-边；因满足条件(2)而相邻时，称连接它们的边为 i-边。

由上述定义可知，$B_{n,k}$ 是 $n-1$ 正则且节点对称的。当 k 取 0 和 1 时，k 元 n 维冒泡排序网络分别约简为 n 个节点的完全图网络和传统的 n 维冒泡排序网络。在无特殊说明的情况下，下文 k 元 n 维冒泡排序网络 $B_{n,k}$ 均满足 $2\leqslant k\leqslant n-2$。

$B_{n,k}$ 具有层次结构。由 $B_{n,k}$ 的结构可以看出，将 $B_{n,k}$ 划分为 $B_{n-1,k-1}$ 的方式仅有一种：删除 $B_{n,k}$ 中的所有 k-边。此时，可将 $B_{n,k}$ 划分为 n 个互不相交的 $B_{n-1,k-1}$。例如，删除 $B_{5,3}$ 中所有 3-边可将 $B_{5,3}$ 划分为 5 个互不相交的 $B_{4,2}$；删除 $B_{4,2}$ 中所有 2-边可将 $B_{4,2}$ 划分为 4 个互不相交的 $B_{3,1}$。图 1 给出了网络 $B_{3,1}$，$B_{4,2}$ 和 $B_{5,3}$。

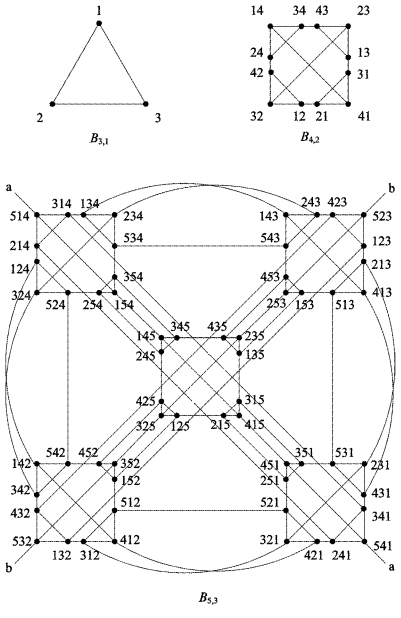

图 1　k 元 n 维冒泡排序网络实例

2.2　子网排除

设 G 是一个具有层次结构的网络，H 是其子网。

在仅有节点发生故障的“节点故障模型”下，定义 G 的 H 节点排除集、H 节点排除割及 H 节点排除数 $F_H(G)$ 如下。

定义 1　设 $P_V\subseteq V(G)$ 是 G 的故障节点集，若 $G-P_V$ 中不含与 H 同构的子网，则称 P_V 是 G 的一个 H 节点排除集。

定义 2　称网络 G 的元素个数最少的 H 节点排除集为 G 的 H 节点排除割。

定义 3　称网络 G 的 H 节点排除割的势为 G 的 H 节点排除数，记作 $F_H(G)$。

在仅有边发生故障的“边故障模型”下，可类似定义 G 的 H 边排除集、H 边排除割及 H 边排除数 $f_H(G)$。

定义 4　设 $P_L\subseteq E(G)$ 是 G 的故障边集，若 $G-P_L$ 中不含与 H 同构的子网，则称 P_V 是 G 的一个 H 边排除集。

定义 5　称网络 G 的元素个数最少的 H 边排除集为 G 的 H 边排除割。

定义 6　称网络 G 的 H 边排除割的势为 G 的 H 边排除数，记作 $f_H(G)$。

3　$B_{n,k}$ 的子网排除算法

3.1　$B_{n,k}$ 的子网表示

记 Λ_n 为符号集 $\{1,\cdots,n,X\}$。其中，X 表示 1 到 n 中的一个不确定符号。令 $2\leqslant k\leqslant n-2$，$m\in I_{k-1}$；$a_1,a_2,\cdots,a_m$ 为 I_n 中 m 个两两不同的整数，设 $M_m=\{x_1x_2\cdots x_{k-m}a_1a_2\cdots a_m\mid x_1,x_2,\cdots,x_{k-m}\in I_n\backslash\{a_1,a_2,\cdots,a_m\}\}$。其中，$x_1,x_2,\cdots,x_{k-m}$ 互不相同。

显然，$B_{n,k}$ 中 M_m 的导出子图同构于 $k-m$ 元 $n-m$ 维冒泡排序子网 $B_{n-m,k-m}$。为便于表达，对于任意的 $m\in I_{k-1}$ 采用长度为 k 的长字符串 $X^{k-m}a_1a_2\cdots a_m$ 来表示 $B_{n,k}$ 中 M_m 的导出子图。譬如，$B_{5,3}$ 中 $\{231,321,241,421,251,521,341,431,351,531,451,541\}$ 的导出子图表示为 X^21。显然，X^21 同构于 4 元 2 维冒泡排序子网 $B_{4,2}$。

3.2　算法描述

目前，在 n 个元素中取 m 个进行全排的算法已经成熟，解决该问题的时间复杂度可控制在 $O(C_n^m m!)$ 内。在不增加时间复杂度的情况下，可修改在 n 个数中取 m 个进行全排的已有算法，从而得到算法 Permutation(n,m) 并使其输出两个指针数组 permu[] 和 unselect[]。其中，对于任意的 $i\in I_n$，permu[i] 指向第 i 个排列，unselect[i] 指向第 i 个排列对应的 I_n 中 $n-m$ 个未被选中的元素构成的升序数组。将每个排列及其对应的未被选中的元素序列均定义成字符数组类型。

算法 1　求解网络 $B_{n,k}$ 的一个 $B_{n-m,k-m}$ 节点排除割和边排除割的算法 Subnet_Preclusion(n,k,m)

输入：正整数 n≥4，2≤k≤n−2，m≤k−1

输出：$B_{n,k}$ 的 $B_{n-m,k-m}$ 节点排除割 P_V，边排除割 P_L

1. begin
2. $\varnothing\Rightarrow P_V$，$\varnothing\Rightarrow P_L$
3. NULL⇒unselect，NULL⇒permu
4. Permutation(n,m)
5. 1⇒i
6. while i≤C_n^mm!
7. substr(unselect[i],1,k−m)⇒p
8. concat(p,permu[i])⇒u

9. replace(v,1,unselect[i][k−m+1])⇒v
10. $P_V \cup \{u\} \Rightarrow P_V$, $P_L \cup \{(u,v)\} \Rightarrow P_L$
11. i+1⇒i
12. end while
13. end

其中，substr(str,i,j)函数返回 str 从第 i 位到第 j 位的子串；concat(str1,str2)返回 str2 链接到 str1 后的结果；replace(str,i,ch)将 str 的第 i 位用 ch 替换，并返回替换后得到的串。

3.3 算法分析

首先，证明上述算法的正确性。

定理 1 算法 1 的输出 P_V 和 P_L 分别为 k 元 n 维冒泡排序网络 $B_{n,k}$ 的 $k-m$ 元 $n-m$ 维冒泡排序子网节点排除割和边排除割；$B_{n,k}$ 的 $B_{n-m,k-m}$ 节点排除数和 $B_{n-m,k-m}$ 边排除数相等，且 $F_{B_{n-m,k-m}}(B_{n,k})=f_{B_{n-m,k-m}}(B_{n,k})=C_n^m m!$。

证明：当 $2\leqslant k\leqslant n-2$，$m\leqslant k-1$ 时，由 $B_{n,k}$ 中子网划分的唯一性可知，$B_{n,k}$ 中每个 $n-m$ 元 $k-m$ 维子网 $B_{n-m,k-m}$ 均可唯一地表示为 $X^{k-m}a_1a_2\cdots a_m$，其中，$a_1,a_2,\cdots,a_m\in I_n$ 且均不相同。因此，$B_{n,k}$ 中不同的 $B_{n-m,k-m}$ 的个数为 $|\{a_1a_2\cdots a_m \mid a_1,a_2,a_3,\cdots,a_m\in I_n\}|=C_n^m m!$。

由算法 1 知，其输出子网中选取一个节点(边)构成 P_V(P_2)恰为对 $B_{n,k}$ 的 $C_n^m m!$ 个不同的 $B_{n-m,k-m}$，每个节点集(边集)。对于 $B_{n,k}$ 中 $C_n^m m!$ 个不同的 $B_{n-m,k-m}$，每个 $B_{n-m,k-m}$ 破坏一个节点(或边)便可攻击掉这些 $B_{n-m,k-m}$。因此，P_V 是 $B_{n,k}$ 的一个 $B_{n-m,k-m}$ 节点排除集，P_L 是 $B_{n,k}$ 的一个 $B_{n-m,k-m}$ 边排除集。

接下来证明 P_V 和 P_L 的最小性。若 $B_{n,k}$ 中 $C_n^m m!$ 个互不相同的 $B_{n-m,k-m}$ 也互不相交，则对于这 $C_n^m m!$ 个不相交的 $B_{n-m,k-m}$，每个 $B_{n-m,k-m}$ 中至少需破坏一个节点(或边)才可以损坏上述 $B_{n-m,k-m}$。因此，只需证明上述 $C_n^m m!$ 个不同的 $B_{n-m,k-m}$ 也不相交即可。

设 $X^{k-m}a_1a_2\cdots a_m$ 和 $X^{k-m}b_1b_2\cdots b_m$ 为 $B_{n,k}$ 中任意两个不同的 $B_{n-m,k-m}$。必定存在某个整数 $i\in I_m$ 使 $a_i\neq b_i$ 成立，否则 $X^{k-m}a_1a_2\cdots a_m$ 和 $X^{k-m}b_1b_2\cdots b_m$ 表示同一个子网 $B_{n-m,k-m}$。令 $u\in V(X^{k-m}a_1a_2\cdots a_m)$，$v\in V(X^{k-m}b_1b_2\cdots b_m)$，则 u 的标号形如 $x_1x_2\cdots x_{k-m}a_1a_2\cdots a_m$，$v$ 的标号形如 $y_1y_2\cdots y_{k-m}b_1b_2\cdots b_m$。因为对于某个 $i\in I_m$ 有 $a_i\neq b_i$，所以 $u\neq v$。进而可知，$X^{k-m}a_1a_2\cdots a_m$ 与 $X^{k-m}b_1b_2\cdots b_m$ 无公共节点，它们不相交。

综上，P_V 是 $B_{n,k}$ 的一个 $B_{n-m,k-m}$ 节点排除割，P_L 是 $B_{n,k}$ 的一个 $B_{n-m,k-m}$ 边排除割。由此可得，$B_{n,k}$ 的 $B_{n-m,k-m}$ 节点排除数 $F_{B_{n-m,k-m}}(B_{n,k})=|P_V|=C_n^m m!$，$B_{n,k}$ 的 $B_{n-m,k-m}$ 边排除数 $F_{B_{n-m,k-m}}(B_{n,k})=|P_L|=C_n^m m!$。

证毕。

算法 1 的时间复杂度分析如下。

算法 1 中，函数 Permutation(n,m)的时间复杂度为 $O(C_n^m m!)$，循环体 while 内各个被调函数的执行步骤数均为 $C_n^m m!$。其中，concat(p,permu[i])函数的自身复杂度为 $O(k)$。因此，算法 1 的时间复杂度均为 $O(k\cdot C_n^m m!)$。

算法 1 构造了 $B_{n,k}$ 的 $B_{n-m,k-m}$ 节点排除割和 $B_{n-m,k-m}$ 边排除割。通过计算相应排除割的元素个数，便可得出 $B_{n,k}$ 的 $B_{n-m,k-m}$ 节点排除数和 $B_{n-m,k-m}$ 边排除数。

4 实例

本节以 3 元 5 维冒泡排序网络 $B_{5,3}$ 的 2 元 4 维冒泡排序子网及 1 元 3 维冒泡排序子网排除问题为例，来说明子网节点排除割和子网边排除割的构造，并计算相应的子网节点排除数和子网边排除数。

算法使用 C 语言实现。硬件环境为 Lenovo Thinkpad X201，Intel i5 四核、主频 2.67GHz CPU，6GB 内存，SanDisk 240GB 固态硬盘；软件环境为 Windows 10 专业版 OS，Visual Studio 2013 程序开发软件。

例 1 构造 $B_{5,3}$ 的 $B_{4,2}$ 节点排除割、$B_{4,2}$ 边排除割、$B_{3,1}$ 节点排除割、$B_{3,1}$ 边排除割，并计算 $B_{5,3}$ 的 $B_{4,2}$ 节点排除数、$B_{4,2}$ 边排除数、$B_{3,1}$ 节点排除数和 $B_{3,1}$ 边排除数。

解：将 $B_{5,3}$ 中所有的 3-边删除，即将 $B_{5,3}$ 划分为 5 个互不相交的子网 $B_{4,2}$，即 $XX1$,$XX2$,$XX3$,$XX4$,$XX5$。再将每个 $B_{4,2}$ 的所有 2-边删除，可得到 20 个互不相交的子网 $B_{3,1}$。

调用第 3 节所述算法 Subnet_Preclusion(5,3,1)求出 $B_{5,3}$ 的 $B_{4,2}$ 节点排除割 P_V 和 $B_{4,2}$ 边排除割 P_L，程序实际运行时间为 0.003s。算法执行结束后，数组 unselect 的元素、permu 的元素、被 P_V(或 P_L)破坏掉的 $B_{4,2}$、故障节点和故障边如表 1 所列。

表 1 $B_{5,3}$ 的 $B_{4,2}$ 节点排除割和 $B_{4,2}$ 边排除割

unselect[i]	permu[i]	受损 $B_{4,2}$	P_V 的节点	P_L 的边
2345	1	$XX1$	231	(231,431)
1345	2	$XX2$	132	(132,432)
1245	3	$XX3$	123	(123,423)
1235	4	$XX4$	124	(124,324)
1234	5	$XX5$	125	(125,325)

由表 1 可知，若需攻击 $B_{5,3}$ 中所有的 $B_{4,2}$，则在节点故障模型下需要攻击至少 5 个节点；在边故障模型下，则需要攻击至少 5 条边。因此，$F_{B_{4,2}}(B_{5,3})=f_{B_{4,2}}(B_{5,3})=5$。

调用算法 Subnet_Preclusion(5,3,2)求出 $B_{5,3}$ 的 $B_{3,1}$ 节点排除割 P_V 和 $B_{3,1}$ 边排除割 P_L，程序实际运行时间为 0.015s。算法执行结束后，数组 unselect 的元素、permu 的元素、被 P_V(或 P_L)破坏掉的 $B_{4,2}$、故障节点和故障边如表 2 所列。

表 2 $B_{5,3}$ 的 $B_{3,1}$ 节点排除割和 $B_{3,1}$ 边排除割

unselect[i]	permu[i]	受损 $B_{3,1}$	P_V 的节点	P_L 的边
345	12	$X12$	312	(312,412)
245	13	$X13$	213	(213,413)
235	14	$X14$	214	(214,314)
234	15	$X15$	215	(215,315)
345	21	$X21$	321	(321,421)
145	23	$X23$	123	(123,423)
135	24	$X24$	124	(124,324)
134	25	$X25$	125	(125,325)
245	31	$X31$	231	(231,431)
145	32	$X32$	132	(132,432)
125	34	$X34$	134	(134,234)
124	35	$X35$	135	(135,235)
235	41	$X41$	241	(241,341)
135	42	$X42$	142	(142,342)
125	43	$X43$	143	(143,243)
123	45	$X45$	145	(145,245)
234	51	$X51$	215	(251,351)
134	52	$X52$	152	(152,352)
124	53	$X53$	153	(153,253)
123	54	$X54$	154	(154,254)

由表 2 可知，若需攻击 $B_{5,3}$ 中所有的 $B_{3,1}$，则在节点故障模型下需要攻击至少 20 个节点；在边故障模型下，则需要攻击至少 20 条边。因此，$F_{3,1}(B_{5,3})=f_{B_{3,1}}(B_{5,3})=20$。

结束语　随着硬件制造工艺瓶颈的趋近，并行计算机系统特别是超级并行计算机系统的性能越来越依赖于其基础网络。当系统遭受攻击而发生节点故障或线路(边)故障时，被攻击方希望知道保护至少多少节点和线路才能保证还存在既定规模的子系统可以正常运行。另一方面，当攻击对手的系统时，希望能够用尽可能小的攻击代价，来使对手系统中既定规模的子系统损失殆尽。上述问题反映到系统的基础网络上，可以归结为子网排除问题。目前，子网排除问题得到完全解决的网络仅限于超立方网络。超立方网络的子网划分方式相对简洁，较易于构造子网节点排除割和边排除割，从而可计算出任意规模子网的子网节点排除数和子网边排除数。

本文通过构造子网节点排除割和边排除割的方案完全解决了 k 元 n 维冒泡排序网络。对于大多数典型的、具有潜在应用价值的著名网络，诸如 k 元 n 立方网络、Star 网络、Bubble-sort 网络、Arrangement 网络等，它们的子网划分方式不唯一。以不同的方式划分时，相同规模的若干子网往往具有公共节点或公共边，从而导致这些网络的子网节点排除割和边排除割的构造异常困难。这些网络的子网排除问题主要在于具体规模的子网节点排除割和边排除割的构造，以及任意规模子网的节点排除数和边排除数的上界优化问题。上述经典网络中任意规模子网的节点排除割和边排除割的构造是子网排除问题有待攻克的难点之一。

参 考 文 献

[1] LATIFI S. A study of fault tolerance in star graph [J]. Information Processing Letters, 2007, 102(5): 196-200.

[2] LATIFI S, SABERINIA E, WU X. Robustness of star graph network under link failure [J]. Information Sciences, 2008, 178(3): 802-806.

[3] WALKER D, LATIFI S. Improving bounds on link failure tolerance of the star graph [J]. Information Sciences, 2010, 180(13): 2571-2575.

[4] WANG S, YANG Y. Fault tolerance in bubble-sort graph networks [J]. Theoretical Computer Science, 2012, 421: 62-69.

[5] YANG Y X, WANG S Y. Recursive algorithm for k-cycle preclusion problem in k-ary n-cubes [J]. Journal of Computer Applications, 2013, 33(9): 2401-2403. (in Chinese)
杨玉星，王世英. k 元 n 立方网络的 k 圈排除问题的递归算法 [J]. 计算机应用，2013，33(9)：2401-2403.

[6] YANG Y, WANG S, LI J. Subnetwork preclusion for bubble-sort networks [J]. Information Processing Letters, 2015, 115(1): 817-821.

[7] WANG S, ZHANG G, FENG K. Fault tolerance in k-ary n-cube networks [J]. Theoretical Computer Science, 2012, 460: 34-41.

[8] WANG S, FENG K. Fault tolerance in the arrangement graphs [J]. Theoretical Computer Science, 2014, 533: 64-71.

[9] FENG K, WANG S, ZHANG G. Link failure tolerance in the arrangement graphs [J]. International Journal of Foundations of Computer Science, 2015, 26(2): 241-254.

[10] WANG M, YANG W, GUO Y, et al. Conditional fault tolerance in a class of Cayley graphs [J]. International Journal of Computer Mathematics, 2016, 93(1): 678-682.

[11] SHAWASH N. Relationships among popular interconnection networks and their common generalization [D]. Oakland, USA: Oakland University, 2008.

[12] CHENG E, LIPTÁK L, SHERMAN D. Matching preclusion for the (n,k)-bubble-sort graphs [J]. International Journal of Computer Mathematics, 2010, 87(11): 2408-2418.

[13] BONDY J A, MURTY U S R. Graph Theory [M]. New York: Springer Press, 2008: 624-626.

(上接第 252 页)

[16] Wine Develop Guide[EB/OL]. http://www.winehq.org/docs/winedev-guide/index.

[17] SHACHAM H, PAGE M, PFAFF B, et al. On the effectiveness of address-space randomization [C]// Proceedings of the 11th ACM Conference on Computer and Communications Security. 2004: 298-307.

[18] LI L, JUST J E, SEKAR R. Address-Space Randomization for Windows Systems [C]// Proceedings of Computer Security Applications Conference (ACSAC'06). 2006: 329-338.

[19] WHITEHOUSE O. An Analysis of Address Space Layout Randomization on Windows Vista [M]. Symantec Advanced Threat Research, 2007.

[20] Inside Windows Vista User Access Control [EB/OL]. https://technet.microsoft.com/zh-cn/magazine/2007.06.uac(en-us).aspx.

[21] UAC Windows7 Tutorial[EB/OL]. http://sourcedaddy.com/windows-7/users-accounts-and-uac.html.

[22] SAMI A, YADEGARI B, RAHIMI H, et al. Malware detection based on mining API calls [C]// Proceedings of the 2010 ACM Symposium on Applied Computing. 2010: 1020-1025.

[23] QIAO Y, et al. Analyzing Malware by Abstracting the Frequent Itemsets in API Call Sequences [C]// Proceedings of 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. 2013: 265-270.

[24] FARUKI P, LAXMI V, VINOD P, et al. Behavioural detection with API call-grams to identify malicious PE files [C]// Proceedings of the First International Conference on Security of Internet of Things. 2012: 85-91.

[25] VirusTotal[EB/OL]. https://www.virustotal.com/en.