北京大学

北京大学博士学位论文

题目：最小顶点覆盖的局部搜索算法

Title：Local Search Algorithms for Minimum Vertex Cover

姓　　名：	蔡 少 伟
学　　号：	10848883
院　　系：	信息科学技术学院
专　　业：	计算机软件与理论
研 究 方 向：	算法设计与分析
导 师 姓 名：	苏 开 乐　教授

二〇一二年五月

版权声明

摘 要

最小顶点覆盖问题是经典的NP难组合优化问题，有着重要的理论意义和广泛的应用。给定一个无向图，顶点覆盖是指该图的一个顶点子集，使得图中每一条边都至少有一个点属于该集合。而最小顶点覆盖问题就是对一个图找出规模最小的顶点覆盖。最小顶点覆盖问题还有两个等价的问题，即最大团问题和最大独立集问题。这两个问题也是经典的NP难问题，并且也有着广泛应用。这三个问题可以看成是一个问题的不同形式化定义，由于它们的理论意义和应用意义，引起了广泛的兴趣和研究。

对于这三个问题的现实求解，目前主要有两类算法：精确算法和启发式算法。精确算法可以保证找到最优解，启发式算法不能保证得到最优解，但是能比较快速的给出较好的解。由于组合爆炸的存在，寻找最优解的精确算法所需要的计算时间会随着顶点和边密度的增加呈指数型地增长，无法用于大型问题实例的求解。所以，对于规模较大的问题实例，一般都诉诸于启发式算法，其中最主要是局部搜索算法。本文的工作集中于设计求解最小顶点覆盖问题的高效局部搜索算法。

本文提出了几个局部搜索策略，并基于这些策略设计了三个高效的局部搜索算法，包括基于部分顶点覆盖和边加权技术的EWLS算法，在EWLS算法基础上利用格局检测策略改进得到的EWCC算法，以及综合了格局检测，两阶段点对交换和带遗忘边加权策略的NuMVC算法。在两个权威的实例组DIMACS和BHOSLIB上的实验说明，这些算法都具有很好的性能。

本文在MVC局部搜索方面取得了一些不错的成果。目前我们保持着挑战实例frb100-40的最新求解纪录，而我们提出的NuMVC算法在大量权威数据集上比以前的其他最小顶点覆盖算法和最大团算法都有相当大幅度的提高，代表着当前最高效的最小顶点覆盖局部搜索算法，另外，本文提出的格局检测是一个一般化的局部搜索策略，可广泛应用于各种组合问题，已经被成功应用于命题逻辑可满足性(SAT)问题的局部搜索算法。

关键词：局部搜索，最小顶点覆盖，格局检测，边加权

ABSTRACT

Local Search Algorithms for Minimum Vertex Cover

Shaowei Cai
Directed by Kaile Su

The minimum vertex cover (MVC) problem is a well-known NP-hard combinatorial optimization problem of great importance in theory and applications. Given an undirected graph $G = (V, E)$, a vertex cover is a subset $S \subseteq V$, such that every edge in G has at least one endpoint in S. The MVC problem is to find the minimum sized vertex cover in a graph. MVC has two equivalent problem, i.e., the maximum clique (MC) problem and the maximum independent set (MIS) problem. Due to their significant importance in theory and applications, these three problems have been widely investigated.

Practical algorithms for these three problems mainly fall into two types: exact ones and heuristic ones. Exact algorithms guarantee the optimality of the solutions they find, but may fail to give a solution within reasonable time for large instances. As the size of the problem increases, the exact methods become futile. On the other hand, although heuristic algorithms cannot guarantee the optimality of their solutions, they can solve large and hard instances in the sense of giving optimal or near-optimal solutions within reasonable time. Therefore, for large and hard problems, one must resort to heuristic approaches, which are mainly local search algorithms. This thesis focuses on designing efficient local search algorithms for MVC.

This thesis proposes several local search strategies, and utilize them to design three efficient local search algorithms for MVC, including EWLS, EWCC and NuMVC. EWLS is based on the concept of partial vertex cover and utilizes an edge weighting technique. EWCC is improved from EWLS by using the configuration checking strategy. NuMVC employs a new search framework called two-stage exchange, and combines configuration checking and edge weighting with forgetting. The experimental results on two standard benchmarks DIMACS and BHOSLIB show that these algorithms are very efficient.

This thesis has some good achievements in this thesis. The new record set by our algorithms for the challenging instance frb100-40 remains the best solution for this

instance. The NuMVC algorithm dominates on most hard DIMACS instances and the whole BHOSLIB benchmark, dramatically improving the existing results, so it represents a significant progress in heuristic algorithms for MVC (MC, MIS). Moreover, the configuration checking strategy is a general local search strategy for combinatorial problems, and has been successfully applied in local search algorithms for the propositional satisfiability problem (SAT).

Key Words: Local Search, Minimum Vertex Cover, Configuration Checking, Edge Weighting

目 录

摘 要 I

ABSTRACT II

第一章 前言 1

1.1 最小顶点覆盖问题 1

1.2 局部搜索算法 3

1.3 研究动机 3

1.4 相关工作 5

1.4.1 启发式算法 5

1.4.2 精确算法 7

1.5 主要贡献及文章结构 8

1.5.1 主要贡献 8

1.5.2 文章结构 9

第二章 局部搜索算法及其实验分析 11

2.1 基本概念 11

2.2 局部搜索算法简介 13

2.3 关于局部搜索的一般性注解 15

2.3.1 局部搜索算法设计是科学与艺术的结合 15

2.3.2 局部搜索中的集中性和多样性 16

2.3.3 循环问题 17

2.4 局部搜索算法的实验方法论 17

2.5 MVC局部搜索算法的实验分析 18

2.5.1 MVC基准实例 18

2.5.2 基本统计量 19

2.5.3 其它实验分析 21

2.6 本章小结 21

第三章 EWLS算法 23
3.1 概念和理论基础 23
3.1.1 基本符号和概念 23
3.1.2 边加权概念和定理 23
3.2 部分顶点覆盖 26
3.3 边加权 27
3.4 EWLS算法描述 27
3.5 算法进一步说明 30
3.5.1 $ChooseSwapPair$函数 30
3.5.2 维护L和UL列表 31
3.5.3 调整PVC大小 31
3.5.4 集中性和多样性的平衡 32
3.6 实验分析 32
3.6.1 实验设置 32
3.6.2 实验结果 33
3.6.3 挑战实例的新纪录 36
3.7 讨论 37
3.7.1 边加权策略的有效性 37
3.7.2 $delta$参数 37
3.7.3 从老到新的边扫描策略 39
3.8 EWLS算法在社会网络研究中的应用 41
3.8.1 社会网络的“团叠加”演化 41
3.8.2 “团叠加”生成器的构建 42
3.8.3 实验分析 43
3.9 本章总结 44

第四章 格局检测及EWCC算法 47
4.1 格局检测 47
4.2 EWCC算法描述 50
4.3 实验分析 50
4.3.1 实验设置 50

4.3.2 比较EWLS和EWCC 52
4.3.3 与其他启发式算法比较 55
4.3.4 与精确算法比较 59
4.3.5 与SAT算法比较 61
4.4 讨论 62
4.4.1 运行时分析 62
4.4.2 关于格局检测的更多见解 63
4.5 格局检测在其他算法的应用 67
4.5.1 利用格局检测改进COVER算法 67
4.5.2 格局检测在SAT局部搜索的应用 72
4.6 本章总结 72
第五章 NuMVC算法 75
5.1 两阶段交换 75
5.2 带遗忘的边加权 76
5.3 NuMVC算法描述 77
5.4 实验分析 80
5.4.1 实验设置 80
5.4.2 NuMVC性能 81
5.4.3 比较实验结果 81
5.4.4 对两阶段交换策略的实验分析 87
5.4.5 对遗忘机制的实验分析 88
5.5 讨论 88
5.6 本章总结 89
第六章 总结与进一步工作 91
6.1 本文内容总结 91
6.2 进一步工作 91
参考文献 93
索引 98
个人简历、在学期间的研究成果 99

致谢 100

表格

3.1 EWLS和其他算法在DIMACS基准实例的比较 34
3.2 EWLS和其他算法在BHOSLIB基准实例的比较 35
3.3 EWLS和EWLS$_0$的性能比较 37
3.4 EWLS在不同$delta$下的性能 38
3.5 EWLS在不同的L扫描策略下的性能 40

4.1 EWLS和EWCC在DIMACS实例的比较，k^*列标注星号的表示k^*已经被证明是最优解的规模 54
4.2 EWLS和EWCC在表现较差的DIMACS实例上的性能 55
4.3 EWLS和EWCC在BHOSLIB 实例的比较 56
4.4 DIMACS实例比较结果 57
4.5 BHOSLIB实例比较结果 58
4.6 EWCC与精确算法MaxCLQdyn+EFL+SCR在DIMACS实例上的比较 60
4.7 EWCC与2004年SAT比赛结果比较 61
4.8 关于格局检测策略的统计 66
4.9 COVER与COVERcc算法在DIMACS实例上的比较 69
4.10 COVER与COVERcc算法在BHOSLIB实例上的比较 71

5.1 NuMVC在DIMACS实例上的性能，k^*列标注星号的表示k^*已经被证明是最优解的规模 82
5.2 NuMVC在BHOSLIB实例上的性能 83
5.3 NuMVC和其他算法在DIMACS实例的比较 84
5.4 NuMVC和其他算法在BHOSLIB实例的比较 85
5.5 NuMVC与其他算法在frb100-40上的比较 87
5.6 Complexity per step on selected instances 88
5.7 NuMVC和NuMVC-在BHOSLIB实例的比较,NuMVC-是NuMVC去掉遗忘机制的版本 89

插图

1.1 顶点覆盖，独立集和团 2
1.2 本文主要贡献及其关系 8

2.1 局部搜索 11
2.2 局部最优 13
2.3 集中性搜索和多样性搜索 17

3.1 一个带边权的简单图 27
3.2 数据结构L和UL 31

4.1 格局检测（红点(浅色点)表示在当前候选解） 49
4.2 EWLS和EWCC与其他算法在BHOSLIB实例上的平均成功率比较 . 59
4.3 EWLS和EWCC在C1000.9(上)和p_hat1500-1(下)的RLD(左)和RTD(右) 其中近似指数函数为$ed[m](x) = 1 - 2^{-x/m}$ 64
4.4 EWLS和EWCC在frb53-24-5(上)和frb59-26-5(下)的RLD(左)和RTD(右) 其中近似指数函数为$ed[m](x) = 1 - 2^{-x/m}$ 65
4.5 EWLS，COVER与EWCC，COVERcc算法在BHOSLIB实例上的平均成功率比较 70

5.1 NuMVC与其他算法在BHOSLIB实例上的平均运行时间比较 86
5.2 NuMVC与其他算法在BHOSLIB实例上的平均成功率比较 86

第一章　前言

1.1 最小顶点覆盖问题

最小顶点覆盖问题是一个典型的图论问题。在介绍问题的定义之前，我们首先介绍一些图论方面的基本概念。读者如果需要对此更详细的了解，请参考文献 [1]。一个无向图(undirected graph)是一个二元组(V, E),其中V为n 个结点构成的非空集合$\{v_1, v_2, ..., v_n\}$，称为顶点集，E是V中元素构成的无序二元组的集合，称为边集。我们用$V(G)$和$E(G)$分别表示图G的顶点集和边集。边$e = \{u, v\}$表示连接顶点u 和v的边，u和v是e的端点，并且$endpoint(e) = \{u, v\}$。对于一个边子集$M \subseteq E$，我们用$endpoint(M)$表示顶点集合$\bigcup_{e\in M} endpoint(e)$。如果两个顶点有边连接，则称这两点是邻接的，两点互为邻居顶点。$N(v) = \{u \in V|(u, v) \in E\}$是顶点$v$的邻域，并且我们记$N[v] = N(v) \cup \{v\}$。顶点$v$的度数为$d(v) = |N(v)|$。图$G$的最大度数为所有顶点中度数最大的点的度数，记为$\Delta(G)$。一个简单图$G = (V, E)$的补图为$\overline{G} = (V, \overline{E})$，其中$\overline{E} = \{(u, v)|u, v \in V, u \neq v \text{ and } (u, v) \notin E\}$。对于一个集合$S$, 我们用$random(S)$表示$S$中的一个随机元素。

下面给出最小顶点覆盖问题的形式化定义：

定义 1.1: (顶点覆盖) 给定一个无向图$G = (V, E)$，顶点覆盖是指图G顶点集的一个子集$C \subseteq V$，使得G的每一条边都有一个点属于C。

定义 1.2: (最小顶点覆盖问题) 给定一个无向图$G = (V, E)$，最小顶点覆盖（Minimum Vertex Cover，简记MVC）问题要求找出一个具有最小基数的顶点覆盖。

最小顶点覆盖问题有两个密切相关的问题：最大独立集问题和最大团问题。他们的形式化定义如下：

定义 1.3: (独立集) 给定一个无向图$G = (V, E)$，独立集是指图G顶点集的一个子集$C \subseteq V$，其中的任意两顶点都不邻接。

定义 1.4: (最大独立集问题) 给定一个无向图$G = (V, E)$，最大独立集（maximum independent set，简记MIS）问题要求找出一个具有最大基数的独立集。

定义 1.5: (团) 给定一个无向图$G = (V, E)$，团是指图G顶点集的一个子集$C \subseteq V$，其中的任意两顶点都邻接。

定义 1.6: (最大团问题) 给定一个无向图$G = (V, E)$，最大团（Maximum Clique，简记MC）问题要求找出一个具有最大基数的团。

顶点覆盖，独立集和团这三个概念联系非常紧密。图1.1显示了这三个概念的例子，图中加黑的点构成的顶点集合表示对应的概念。对于一个无向图$G = (V, E)$，一个顶点集合S是G的一个独立集当且仅当$V \backslash S$是G的一个顶点覆盖; 一个顶点集合K是G的一个团当且仅当$V \backslash K$是补图$\overline{G}$的一个顶点覆盖。因此，MIS问题和MC问题都可以和直接地规约到MVC问题。为了求得G的一个最大独立集, 我们可以找出G的一个最小顶点覆盖C，然后返回$V \backslash C$。类似地，为了找到G的一个最大团, 我们可以找出其补图$\overline{G}$的一个最小顶点覆盖C，然后返回$V \backslash C$.

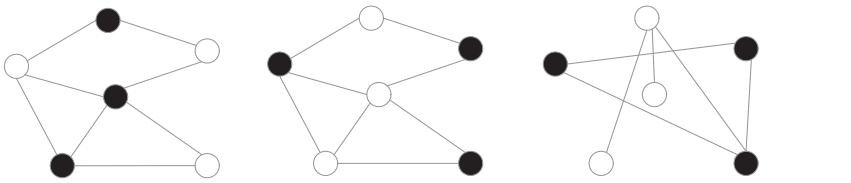

图 1.1　顶点覆盖，独立集和团

从顶点覆盖，独立集和团这三个概念的关系，不难看出最小顶点覆盖(MVC)，最大独立集（MIS）以及最大团（MC）这三个问题的等价关系。虽然三个问题在理论上存在不同的属性和难度刻画，不过从现实求解的角度看，也就是从实验算法的角度，这三个问题可以被认为是同一个问题的三种等价形式。MVC算法可以直接用来求解MIS和MC问题。用不同的形式化定义来描述问题，使我们对问题的复杂性有更多的理解，并且针对不同形式的定义，求解算法也有很大的不同。这里还有区分两个概念，在计算机算法研究中，一般而言，“问题”是指抽象的问题，即从问题的定义出发来讲的；而“问题实例”是指具体化的问题，即从问题的具体输入数据来讲的。计算机算法在求解问题的时候，需要输入问题实例。

以上三个问题是经典的NP难问题，它们的判定版本是早期的6个NP完全问题其中的三个 [2]。MVC问题在现实中多个领域，比如网络安全，调度问题和

大规模集成电路设计等，都有着广泛的应用 [3]。另外，MC问题也因其广泛应用而著名，其应用领域包括信息检索，实验设计，信号传输，计算机视觉 [4]，尤其是计算生物学，比如DNA序列和蛋白质的检测等 [5]。由于这三个问题的难度和它们在工业界的直接应用，任何对问题求解方面小的的优化都可以产生巨大的经济效益。

1.2 局部搜索算法

局部搜索（Local Search）是一种求解优化（尤其是组合优化）问题的方法。组合优化（Combinatorial Optimization）是优化问题的一个分支，研究的是组合的（即可行解为离散情形或可行解能简化为离散情形）优化问题，其目标为在有限个可供选择的方案中找出最优的方案，通常可描述为：令$S\{s_1, s_2, ..., s_n\}$为所有候选解构成的解空间，其中$S' \subseteq S$为问题的可行解集合，$f(s_i)$为状态s_i对应的目标函数值，要求寻找最优解s^*，使得对于所有的$s_i \in S'$，有$f(s^*) = minf(s_i)$（对最小化问题）或$f(s^*) = maxf(s_i)$（对最大化问题）。最小顶点覆盖问题是一个典型的组合优化问题。

局部搜索算法具有简单和高效两个特点。优化是非常广的一个问题模型，几乎所有问题都可以看成某种形式的优化问题。只要再定义解空间的邻域结构就可以用局部搜索算法来求解了。局部搜索算法的思路和实现也相当简单。局部搜索算法还是求解NP难问题的一个高效的方法，比如对于皇后问题，旅行商问题和SAT问题，局部搜索算法都是公认的最有效的一类算法，尤其对大规模问题实例。

为了应用局部搜索算法，首先要定义候选解的邻域结构，即什么样的两个候选解是邻居。定义好邻域结构之后，问题的解空间就可以看成一个图。局部搜索算法从解空间的一个点出发，每一步从当前候选解移动到一个邻居候选解，去寻找较好的候选解，候选解的质量是由一个评估函数来界定的。算法从当前候选解跳转到邻居解的时候，只根据有限的局部信息来做决定。

1.3 研究动机

算法领域是计算机科学的核心领域，该领域可以分为理论算法和实验算法两个主要方向。理论算法方向致力于推导出更好的理论上界，而实验算法则致力于设计求解难解问题更快的实用算法。MVC，MIS和MC这三个问题是算法领

域非常重要的难解问题，由于其基础性和重要性被写进在各种算法教科书。最大团问题也是美国离散数学学会（DIMACS）于1992年提出的3个挑战问题的其中之一（另外两个问题是SAT和图着色问题）。

MVC，MIS和MC这三个问题都是NP难的，它们的判定版本都是NP完全问题 [2]。而且,这三个问题都有很强的难近似性。对于MVC问题，要在多项式时间内找一个近似比为1.3606的解是NP难的 [6]; 目前最好的MVC近似算法只能达到$2-o(1)$的近似比 [7, 8]。对于其他两个问题，Håstad [9, 10]证明了MIS和MC不可能有近似比为$|V|^{1-\epsilon}(\epsilon>0)$的多项式时间算法除非NP=ZPP[1]。最近，Zuckerman对 [9]中的结果去随机之后得到更强大结论，即MC问题不可能有近似比为$|V|^{1-\epsilon}(\epsilon>0)$的多项式时间算法，除非NP=P [11]。当前，对于MC问题最好的多项式近似算法是由Feige提出的算法，其近似比为$O(n(loglogn)^2/(logn)^3)$ [12]。

对于MVC(MIS，MC)问题的现实求解，目前主要有两类算法：精确算法和启发式算法。精确算法基本都采用分支限界方法，如 [13–19]。启发式算法则主要是局部搜索算法，如MVC局部搜索算法 [20–23]，MIS局部搜索算法 [24–27]，和MC局部搜索算法 [4, 28–33]。精确算法可以保证找到最优解，但是随着问题规模的增长，其运行时间呈指数级别增长。所以精确算法不适合用于求解大型实例。另一方面，虽然启发式算法不能保证得到最优解，但是能够对大型难解实例比较快速的给出较好的解甚至是最优解。现实应用中的很多实例都是大规模的，对于这些实例，一般都诉诸于启发式算法。

虽然已经有了不少求解MVC (MIS, MC)问题的启发式算法，但是这些算法的性能仍然不能令人满意，特别是对难解实例的求解。比如，有一个困难的挑战实例frb100-40，该图有4000个点，图中隐藏了一个规模为3900 的最小顶点覆盖。在我们的工作 [34]之前，对该实例找到的最好的解只是规模为3903的顶点覆盖 [35]。可能有人会认为，最近针对可满足性问题(SAT)的算法已经取得了非常好的成果，很多SAT实例都可以被高效地求解。或许我们可以把MVC (MIS，MC)问题实例转为SAT实例，然后调用SAT算法求解。然而实际情况是，在这些问题上，SAT算法还比不上专门的MVC (MIS，MC)算法。比如对于RB模型产生的随机相变实例，SAT算法的性能比MVC算法的性能要弱很多 [36], 这即使对于最先进的SAT算法比如2011年比赛的得奖算法也是如此 [37]。因此，

[1]ZPP是一个问题类，其中的问题可以被一个概率算法以0错误的概率在多项式期望时间内求解。

一个重要的研究课题就是如何更高效地求解大的困难的MVC (MIS, MC)问题实例。基于以上考虑，本文主要关注于设计高效的局部搜索算法求解MVC问题。

1.4 相关工作

1.4.1 启发式算法

学者们已经提出很多启发式算法（多是局部搜索算法）用于求解MVC，MC和MIS这三个问题。本节对已有的启发式算法做一个简单的回顾。

早期的MVC启发式算法很多是拟物算法。Evans在 [20]一文提出了一个新的MVC进化算法，并回顾了以前的MVC进化算法。 [21]和 [22]提出了MVC的蚁群算法。不过这些拟物算法的性能都不是很好。2007年的MVC局部搜索算法COVER [23]通过迭代求解顶点覆盖判定问题来求解MVC问题，算法每个搜索步交换两个顶点（一个属于当前候选解，一个在当前候选解之外），保持候选解大小不变，争取覆盖最多的边。COVER算法基于迭代局部搜索算法的框架，使用了边加权技术来引导搜索。该算法提出一个$gain(u,v)$评估函数，衡量每对顶点交换之后能带来的好处。算法每一个搜索步都选取最好的一对顶点进行交换，然后对没覆盖的边的权值加1。COVER本身是一个顶点覆盖算法，也即要求输入要求的顶点覆盖的目标大小。作者在文中也提出通过迭代运行该算法就可以求解MVC问题。并且显示用于求解MVC的迭代版本COVER算法在时间性能上并不会比COVER算法差很多。在本文的工作之前，COVER是拥有最佳性能的MVC局部搜索算法。

作为美国离散与应用数学学会(DIMACS)在第二届实现挑战“NP hard problems: The Second DIMACS Implementation Challenge”中提出的3个NP难问题之一，MC问题早期的比较成功的启发式算法收录在 [38] 一书。MC局部搜索算法一个重要的突破是2001年Batti提出的自适应局部搜索（Reactive Local Search，RLS)，该算法在DIMACS实例上优胜于以前的局部搜索算法 [39]。RLS 算法先随机生成一个团K，然后通过各种操作包括添加、替换、跃迁和重启等不断地对团K 进行改进。RLS的最重要的搜索策略是自适应禁忌策略。第二个里程碑式的工作是Pullan和Hoos于2006年提出的动态局部搜索最大团算法(Dynamic Local

Search Maximum Clique, DLS-MC) [4]。在 [4]一文中，Pullan回顾了当时的几个最好的五个算法，包括：Battiti和Protasi于2001年提出的自适应局部搜索（Reactive Local Search，RLS) [39]、Busygin于2002年提出的QUALEXMS算法 [28] 、Grosso, Locatelli和Croce于2004年提出的深度自适应贪心搜索(Adaptive Greedy Search，DAGS) [29]、Katayama, Hamamoto和Narihisa于2004年提出的k-opt算法 [30] 及Katayama, Sadamatsu和Narihisa于2007年提出的在k-opt算法基础上演化的迭代k-opt算法 [31]、Solnon和Fenet于2006年提出的Edge-AC+LS算法 [32]。在DIMACS实例上的实验表明，DLS算法在绝大部分DIMACS实例上都优于这些算法 [4]。DLS算法维护一个团，在迭代改善和平台搜索两种模式中切换。在迭代改进模式中，每一步都向当前团中加入一个新的点构成更大的团；当迭代改进模式无法进行下去的时候，算法就进行平台搜索。在平台搜索中，算法每一步会交换团中的一个点和团外的一个点，保持团大小不变，以此希望在某一步可以进行迭代改进模式。DLS算法引入了顶点惩罚策略，在每一个平台搜索的最后一步，对不在当前团的点进行惩罚，使得长期不被选中的点有机会被选中。DLS的不足之处在于，它有一个依赖于实例的参数pd用于决定什么时候对点的权值进行减少，这个参数的值随着问题的不同而不同，很难事先确定。随后，Pullan改进了DLS，提出了多阶段局部搜索算法(Phased Local Search, PLS)，从而避免了使用这个参数 [40]。PLS算法由三个不同的子算法构成，搜索的时候在子算法间来回切换。该算法算法没有依赖于实例的参数，并且性能上和DLS差不多或者更好。

相当于最大团和MVC，针对最大独立集（MIS）的启发式算法最近进展较少。早期的MIS启发式算法包括：优化交叉启发式（OCH） [24]，基于球体二次方程式的QSH [25]，还有进化算法WAO [26]。最近比较值得注意的进展是 [27]一文提出了快速实现点对交换的方法，并且在一个迭代局部搜索中应用了该方法。得到的新算法在一些实例上比著名的最大团算法DLS要好，不过在大多数问题实例上，还是显得较弱。

在本文的实验研究中，不是所有的算法都会被涉及。那些没有出现在我们的实验中的算法是因为它们已经（在已有文献中）被证明效率不佳。在对本文提出的算法进行实验评估时，我们只和该工作发表时性能最好的那些算法进行比较。

1.4.2 精确算法

虽然本文的工作主要关注于设计启发式算法，然而为了对最大团问题算法的现状有更全面的认识，我们在这里也简单介绍一下最大团精确算法的现状。

精确算法通常是通过隐式地枚举搜索空间中所有的局部最大团来得到结果 [13–19, 41]。若目标是找一个最大团或只考虑最大团的结点数，则可使用著名的也是常用的(分支限界(branch and bound) 方法。设计求解最大团的分支限界算法的关键在于：1. 如何找到一个好的下界，即最大团的结点数较大的下界? 2. 如何找到一个好的上界，即最大团的结点数较小的上界? 3. 如何分支，即如何将问题分解为较小的子问题? 其中最困难的是如何找到好的上界。目前求解最大团问题的分支限界算法多采用启发式分类算法将图分为若干个独立的集，从而得到一个上界。若图G可被分为k个独立的集，因最大团在每个独立集中至多选择一个结点，所以最大团包含的结点数不会超过k个。Konc和Janezic于2007年提出的算法MCQdyn [41]，以及Tomita和Kameda于2009年提出的算法 [17]是使用分支限界方法求解最大团问题的典型算法。此方法存在两点不足：(i)对图G分解的独立集的个数常常不是最小的，因为分解问题本身就是一个NP难度问题；(ii) 当图G 不是完美图(perfect graph)时，即使分解的独立集个数恰好等于最小数，也不能得到一个紧的上界。以上两点不足也许可以解释求解最大团精确算法的研究为什么总是停滞不前。

李初民和全喆于2010年提出，命题推理可用来明显地改进图分解所得出的上界，他们提出了一个新的算法MaxCLQ [18]。一方面，SAT命题推理是低层次的推理方法，通常不受图结构信息的引导。另一方面，求解最大团问题的经典算法在较高的层次上利用图的结构信息，但漏掉了某些图的细粒度信息。两者的结合将实现优势互补，得到好的计算结果。接着，李初民和全喆进一步利用推理技术“扩展失败文字检测”(Extended Failed Literal detection)策略和“软子句放松”(Soft Clause Relaxation)方法对MaxCLQ进行改进，得到了新的算法MaxCLQdyn+EFL+SCR [19]，该算法在DIMACS测试集上明显地优于之前其他最大团精确算法。不过，对于困难的随机实例集BHOSLIB，目前还没有精确算法能进行有效求解。

1.5 主要贡献及文章结构

1.5.1 主要贡献

本文针对MVC问题，提出了五个新的局部搜索策略及三个新的局部搜索算法。其中每个算法的性能在该算法发表的时候都比已有的MVC(MIS，MC)启发式算法更高效。图1.2展示了本文的主要贡献。在图1.2中，我们描述了这些策略和算法的关系，并标出了本文工作突出的亮点。具体研究内容如下：

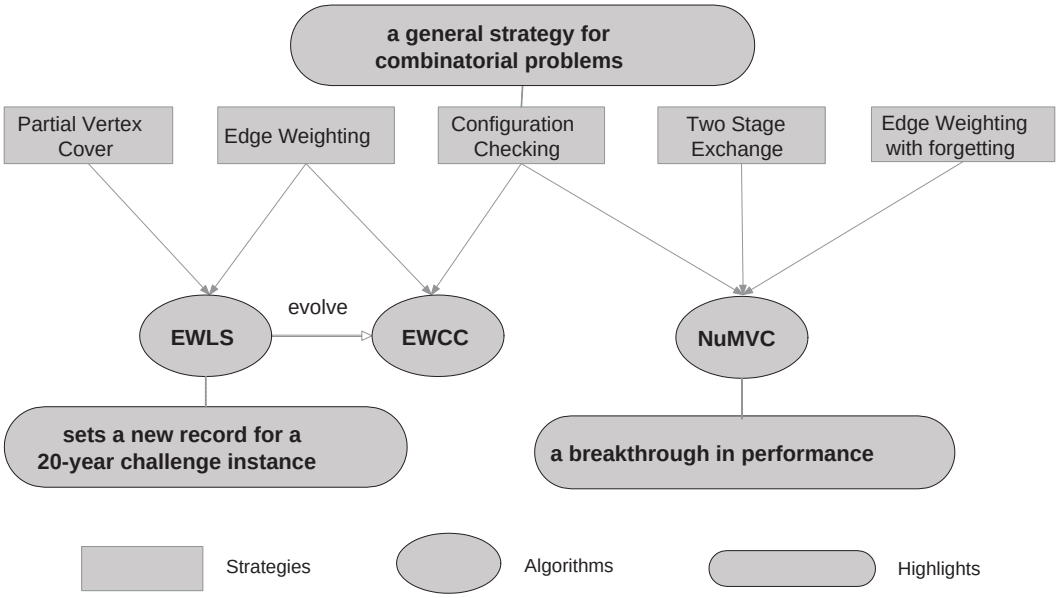

图 1.2　本文主要贡献及其关系

在策略上，我们首次提出了以下策略：

(1) 扩展部分顶点覆盖(Partial Vertex Cover) [34]: 当找到的部分顶点覆盖提供的解上界比保存的最优解更小时，算法通过覆盖剩余的边把它扩展为一个完整的顶点覆盖并且保存为当前找到的最优解。

(2) 格局检测(Configuration Checking) [36]: 是用于处理局部搜索中的循环问题的策略；当一个点被移除出当前候选解之后，如果它的所有邻居点的状态（是否在当前候选解中）都没有发生改变，则该点不允许重新加入当前候选解。格局检测策略是一个一般性的局部搜索策略，可应用于各种组合问题的局部搜索算法。

(3) 两阶段交换(Two Stage Exchange) [42]: 以前的算法在交换点对的时候都是先选出要交换的点对，然后进行交换，而两阶段交换则是先从当前候选解中选一个点移除，然后再选一个点加入，这使算法单步复杂度从平方降为线性。
(4) 边加权的遗忘机制(Forgetting Mechanism) [42]: 周期性地对边权进行减权，使得近期加权比早期加权更重要，更能引导搜索，以此使得边加权技术更有效。

在算法上，基于以上策略，我们设计了三个新的MVC局部搜索算法：

(1) EWLS算法 [34]：算法主体在于寻找一个更好的部分顶点覆盖。在引导搜索上，EWLS使用了一个新的边加权技术，在遇到局部最优的时候对未覆盖的边增加权重。EWLS算法在挑战实例frb100-40上刷新了求解记录，找到了规模为3902的顶点覆盖。

(2) EWCC算法 [36]：EWCC是在EWLS的基础上，利用格局检测(Configuration Checking)策略进行改进得到的新算法。格局检测策略使得EWCC算法在广泛的问题实例上比已有的MVC(MC,MIS)算法好很多。
(3) NuMVC算法 [42]：NuMVC采用了两阶段交换的框架，还使用了带遗忘机制的边加权技术。从性能上讲，NuMVC是MVC(MC,MIS)局部搜索算法的一个大突破，尤其在随机难解实例上，平均时间比之前最好的算法EWCC要好3倍左右。

在MVC之外，我们还基于格局检测策略，也设计了三个高效的SAT局部搜索算法，包括Swcc [43]，Swqcc [44]，和Swcca [45]，以及一个高效的求解weighted Max2SAT的局部搜索算法ANGScc [46]。

1.5.2 文章结构

本文的剩余部分组织如下：第2章对局部搜索算法及其实验分析方法进行介绍。第3章介绍EWLS算法及算法中的创新策略，包括部分顶点覆盖和边加权，并对它的性能进行实验分析。第4章介绍格局检测策略和EWCC算法，并对EWLS和EWCC进行详细实验分析。第5章介绍NuMVC算法及它的创新策略，

包括两阶段点对交换和带遗忘边加权，并对它进行实验分析。第6章对本文做了总结，并提出下一步要做的工作。

第二章　局部搜索算法及其实验分析

本章首先介绍局部搜索算法的基础知识，包括一些基础概念，简单的算法例子，优缺点，以及研究中比较重要的问题。然后简单介绍局部搜索算法的实验分析，最后结合本文工作介绍MVC(MC,MIS) 局部搜索算法的实验分析方法。

2.1 基本概念

局部搜索（Local Search）是一种求解优化问题的方法。优化是一种非常广的模型，几乎所有问题都可以看成是某种形式的优化问题，即在某个可行域中寻求目标函数的最优值。定义好候选解的领域结构之后，问题的解空间就可以看成一个图。局部搜索算法从解空间的一个点出发，每一步从当前候选解移动到一个邻居候选解，去寻找较好的候选解，候选解的质量是由一个评估函数来界定的。局部搜索算法从当前候选解跳转到邻居解的时候，只根据有限的局部信息来做决定。图2.1的左边是搜索空间的一个区域的鸟瞰图，其中c表示当前候选解，s表示问题的最优解；图的右边表示局部搜索每一步只根据当前的局部信息，邻居候选解的质量好坏由评估函数定义。

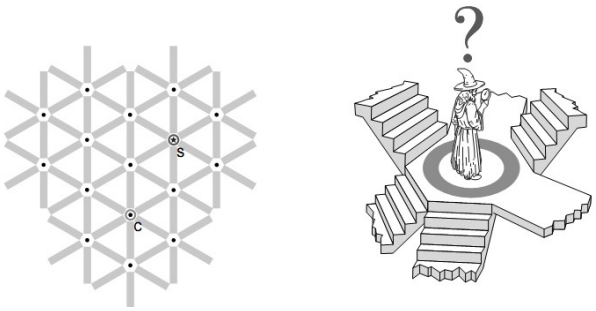

图 2.1　局部搜索

给定一个（组合）问题，一个用于求解该问题任意实例的局部搜索算法可以由以下几个部分形式化定义 [47]：

(1) 搜索空间S，包含所有候选解的有限集合。这里候选解也常被称为（搜索）位置，点，状态。

(2) 可行解集合$S' \subseteq S$，包含问题所有的可行解，可行解不一定是最有解，但最优解必须是可行解。
(3) 候选解的邻域关系$R \subseteq S \times S$，定义了什么样的两个候选解是邻居。
(4) 初始化函数$init : \emptyset \longmapsto \mathcal{D}(S)$。初始化函数确定了候选解集合的一个概率分布，算法根据初始化函数选择访问的第一个的候选解。
(5) 转移函数$step : S \longmapsto \mathcal{D}(S)$。转移函数把每一个候选解映射到候选解集合的一个概率分布上，算法根据转移函数选择下一步即将访问的候选解。一次转移也成为一个搜索步。

以上定义是对于基于单个候选解的局部搜索算法而言，也即每个搜索步都只访问候选解。然而我们要说明，局部搜索算法并不局限于单个候选解，也可以每一个步都维护多个候选解，实际上进化算法和蚁群算法就是这样做的。上面的定义也很容易扩展到基于多个候选解的局部搜索算法，只要把初始化函数和转移函数相应改为$init : \emptyset \longmapsto \mathcal{D}(\mathcal{P}(S) \setminus \emptyset)$和$step : S \longmapsto \mathcal{D}(\mathcal{P}(S) \setminus \emptyset)$即可，其中$\mathcal{P}(S)$是$S$的幂集，也即包括了$S$的所有子集的集合。

一般而言，局部搜索算法都会用到一个评估函数$g : S \longmapsto R^+$，用于衡量候选解质量的好坏。评估函数是针对算法而言，而目标函数是针对问题而言，这两个函数不一定一样。但是设计局部搜索算法时，选取的评估函数一般要满足评估函数值比较小的候选解其目标函数值也比较小。这样在求解对最小（大）化问题才能把搜索引导到目标函数值小（大）的候选解。

在讨论局部搜索算法的时候，我们还需要区分局部最优解(Local Optimum)和全局最优解(Global Optimum)的概念。这里我们以最小化问题为例，见图2.2。假设问题的目标函数为f, s是一个候选解，如果对于s的所有邻居候选解s'，都有$f(s) \leq f(s')$，则称s为局部最优（解），也叫局部最小（解）；如果对于所有候选解s'，都有$f(s) \leq f(s')$，则称s为全局最优（解），也叫全局最小（解）。

和大多数文献一样 [47]，本文认为，一切符合以上定义的算法都属于局部搜索算法，包括各种拟人拟物方法，如拟退火算法(Simulated Annealing) [48]和进化算法(Evolutionary Algorithms) [49]等。局部搜索相当于一个算法框架，在这个框架下，可以运用各种启发式策略来设计局部搜索算法的各个部分，主要包括对邻域，评估函数的设计，还有特别是转移函数的设计。进化算法借鉴了生物进化的思想来设计转移函数，而模拟退火则是借鉴了物理退火过程的现象设

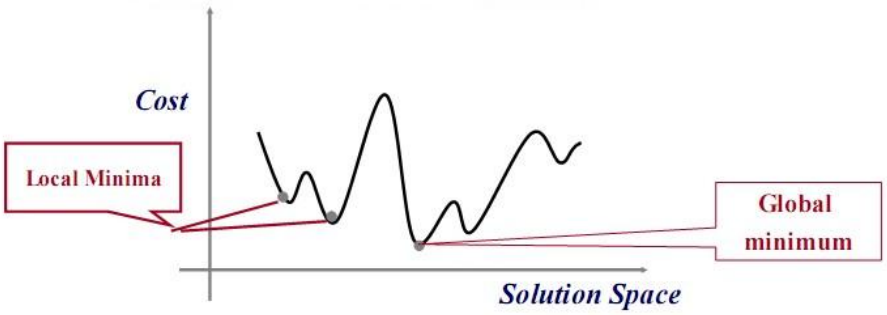

图 2.2　局部最优

计转移函数。这些拟物的算法因为其易于理解和实现，被各个行业的人所了解而著名。然而从对文献的调研中可以发现，对于特定问题，最优的算法往往是针对该问题设计的局部搜索算法。但这些拟物算法还是凭借其易于理解和门槛较低而保持着一定活力。

另外，本文讨论的局部搜索算法都是基于单个候选解的。从文献得知，比如最著名的NP难问题，包括SAT（命题逻辑可满足性问题），TSP（旅行商问题），最大团和图着色等，目前最好的局部搜索算法都是基于单个候选解的，基于多个候选解的局部搜索丰富了算法设计的思路，然而目前没有明显优势。

2.2 局部搜索算法简介

各种局部搜索算法的差异主要在于转移函数的设计，即如何从当前候选解转移到下一个候选解。所以在下面的介绍，我们主要描述他们的转移函数，也即一个搜索步是怎么进行的，这也是局部搜索算法最本质的部分。

按照转移函数的构成情况，局部搜索算法可分为简单局部搜索算法和复合局部搜索算法。

简单局部搜索算法主要有：迭代改进，随机游动，随机迭代改进，简单禁忌搜索等。

(1) 迭代改进（Iterative Improvement）：也称为爬山法，每次从当前候选解的邻居候选解中选择一个最优解进行转移，直到达到一个局部最优解。
(2) 随机游动（Random Walk）：每次从当前候选解的邻居中随机地选取一个进行转移。
(3) 随机迭代改进（Random Iterative Improvement）:每个搜索步首先产生一个

随机数$r \in [0,1]$，然后以r的概率进行随机游动，以$1-r$的概率进行迭代改进。

(4) 简单禁忌搜索（Simple Tabu Search）: 每个搜索步都从没被禁忌的邻居中选择一个最优的候选解进行转移。

从简单的局部搜索算法出发，可以构造很多复合局部搜索算法，这里举两个典型的例子：迭代局部搜索，贪心随机自适应搜索。

(1) 迭代局部搜索（Iterative Local Search）：在没有达到局部最优时，一直执行迭代改进算法；在达到局部最优的时候，执行随机游动算法。迭代局部搜索其实相当于搜索一系列的局部最优，然后返回最好的那个。

(2) 贪心随机自适应搜索（Greedy Random Adaptive Search Procedures）: 简称GRASP，可以看出一系列的大的迭代，每一次迭代首先用贪心的方法构造一个初始解，然后执行一个局部搜索算法来找到局部最优，之后进入下一个大迭代。

Hoos和Stützle提出用自动机来描述复合局部搜索算法 [47]，其中每个状态是一个简单的局部搜索算法，算法从初始化状态开始，在各个简单局部搜索算法之间切换，每个切换都依赖于某些条件。当复合局部搜索算法比较复杂时，这不失为理解算法的好途径。

局部搜索算法有几个明显的优点：

(1) 简单，通用。优化是非常广的一种模型，几乎所有的问题都可以看出某种形式的优化问题。只要再定义解空间的邻域结构，就可以应用局部搜索算法进行求解了。

(2) 高效。已经有很多研究表明局部搜索算法的高效，特别是对NP难问题的大实例，局部搜索算法似乎是最有希望的一种方法。

(3) 容易并行。因为很多局部搜索算法都是SLS，不同随机种子的运行直接并行，而不像精确算法的并行需要很大的工作量。

局部搜索算法具备的优点足以使得它有资格成为算法设计的一种基本方法。实际上，局部搜索算法已经被成功地应用于求解很多问题，尤其是难解的组合问题，比如SAT问题 [45, 50–56]，以及本文研究的最小顶点覆盖和它的两个等价问题 [4, 23, 34, 36, 39, 40, 42]。

当然，局部搜索算法也有它的缺点。最主要的一个缺点就是，局部搜索算法不能保证找到的解的质量。另外，对局部搜索算法的研究目前主要是基于算法的实验分析，对局部搜索缺乏一个统一的理论。

2.3 关于局部搜索的一般性注解

2.3.1 局部搜索算法设计是科学与艺术的结合

局部搜索算法设计是一门科学，但更是一门艺术。虽然局部搜索的算法框架简单，设计一个局部搜索算法也很容易，但要得到一个高效的局部搜索算法却是非常具有挑战性的任务，它不仅需要有科学的方法，也需要类似于艺术领域中的灵感。为了得到高效的局部搜索算法，算法设计者需要设计巧妙的启发式策略，以及有效地组合这些策略。这些是局部搜索算法设计的难点，而且目前并无成熟的章法可循。如果想通过尝试的方法来找到高效的策略，那几乎是不可能的，因为这类似于在无穷的策略空间中用盲目搜索去找令人满意的策略。我想这不管是人类的大脑还是计算机都是无法胜任的。目前对局部搜索算法的设计虽然有许多文章，然而，这些文章并未提及如何才能想到这些策略。这也是算法设计的魅力所在，它不存在工程化的方法，只有勇于探索才能找到出路。

本文作者通过亲身科研体验，认为要提出高效的局部搜索算法，首先要对问题有深刻的理解，只有对问题有足够深的理解了，才能利用问题自带的信息设计算法。比如对于邻域和评估函数的设计，对问题的理解越深，越是感到熟悉而自然，就越能提出更好的邻域函数和评估函数。第二点，局部搜索算法的想法也常常来源于其他学科，甚至是人类的经验知识。多涉猎各种学科的知识不仅能开阔思路，更甚者，能直接从中得到启发，把其他学科的方法应用到局部搜索中来，比如模拟退火算法就是受到物理退火过程的启发而设计的。第三点则是需要持之以恒的思考，并用科学的方法对发展中的策略和算法进行改进。

现在以本文第四章中的格局检测策略为例谈谈上面的三点体会。格局检测这个想法起初来源于本文作者对局部搜索算法的一个思考：局部搜索算法没有剪枝能力，如何使得它具备剪枝或者类似的能力？如果能做到这一点，可以避免算法的一些无用搜索，从而显著提高局部搜索算法的性能。另一方面，本文

作者由于对社会学和博弈论的兴趣阅读了这些方面的书籍，理解了全局利益的最优化经常是要考虑其他参与者的情况，完全理性（自私）的参与者组成的群体只能达到局部最优的全局利益，这方面最典型的例子就是博弈论中的各种均衡。总的来说，格局检测就是来源于这两个思考的结合，经过一段时期的思考，也从与老师和朋友的讨论中得到启发，模糊的想法一步步清晰。为了实现格局检测策略，需要好的数据结构和算法，也需要对之精益求精，在这些方面，就要用到科学理性的思维和方法了。比如对格局检测采用近似实现是考虑了算法的单步复杂度的原因，另外还需要安排实验比较全面地认识该策略，比如算法加入策略前后的单步时间和步数收敛速度，以及该策略使得哪些量发生了变化，这些都有助于我们看清策略的本质。

本小节最后以算法大师Edsger Dijkstra在“On the Nature of Computing Science”一文中谈论计算科学时曾经引用的一句话结尾：“优雅，就是简单但非常有效”。这应该成为算法设计者在设计算法时的一个信条。

2.3.2 局部搜索中的集中性和多样性

当我们在研究局部搜索算法时，有两个经常要讨论到的概念，即集中性和多样性。从某种程度讲，性能较好的局部搜索算法都有集中性搜索和多样性搜索，并且在搜索过程中切换得当。这就提供了一种研究局部搜索算法的思路，即从集中性搜索和多样性搜索的分配和切换是否合理来分析和改进算法。集中性和多样性是搜索的两个方面，见图2.3。集中性搜索试图改进解，而多样性搜索试图寻找不同的解空间以避免落入局部最优。集中性搜索可以改进解，但容易使搜索陷入解空间的一个不包含最优解的小区域。多样性搜索可以使搜索多样化，但有可能错过搜索空间中某个区域中的最优解。好的切换方案将根据搜索的特点和解空间的结构特性，使搜索过程能较容易地避开局部最优并能快速地找到最优解。

实际上，局部搜索算法提出的很多策略都是集中性搜索或多样性搜索的某种体现。当局部搜索算法研究者们意识到可以从集中性和多样性搜索更好地理解局部搜索算法之后，在对局部搜索策略和算法的有效性的直观解释时更是频繁提起这两个概念。

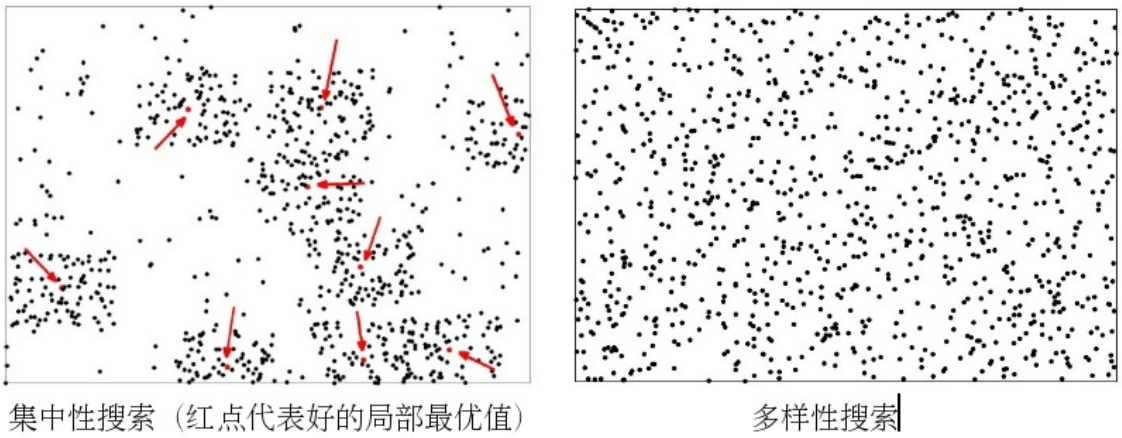

图 2.3　集中性搜索和多样性搜索

2.3.3 循环问题

局部搜索算法在搜索的时候，因为其只能保留当前局部信息，经常会重复访问最近已经访问过的候选解，这个现象叫做循环现象，也称为局部搜索的循环问题 [57]。循环问题不仅浪费了算法大部分时间，也使得算法经常陷入局部最优，从而导致算法性能下降。另一方面，循环问题又是局部搜索的固有问题，因为局部搜索算法并不记录访问过的候选解。如果我们增加一个机制来记录访问过的候选解，则需要指数空间才能做到。所以，这个问题本质上是不可消除的。所以局部搜索算法的一个重要问题就是如何尽量避免陷入循环，其实之前很多算法的从局部最优跳出来的策略也是减少循环的一种体现。由于局部搜索的循环问题在实用算法中不可能彻底消除，如何有效地减少循环就是一个重要的研究课题。本文提出的格局检测策略可以较好地解决该问题。

2.4 局部搜索算法的实验方法论

本节给出局部搜索算法进行实验分析的必要性和实验分析方法的简单说明，关于具体如何进行实验分析我们在下一节结合MVC问题进行说明。

局部搜索算法作为一种实用算法而得到广大认可，它可以有效求解诸如NP难问题等难解问题。作为实用算法的局部搜索算法，实验分析是一个直接并且能实际反映局部搜索算法性能的评估方法。在评估实用算法的时候，即使一个算法有再好的理论和解释，如果不能通过实验表示其在实际应用中具有

良好性能，仍然不是一个能在实际运行中表现良好的算法。虽然目前也有一些关于局部搜索算法的理论分析，但这些成果目前仍然是少而零散，也不是本文关注的方向。本文主要是通过实验分析和比较来评估一个局部搜索算法的性能的。实验分析是基于算法在基准实例上的性能表现来比较算法性能。基准实例是指大家认可的权威的问题实例或者通过大家认可的生成模型生成的问题实例。

局部搜索算法的一个重要特点是灵活性。局部搜索算法是逐步改进一个完整解，因此可以随时给出当前最好的解作为近似解。同时，随着时间的增加，局部搜索算法可以逼近全局最优解。局部搜索算法第二个特点是随机性。大多数局部搜索算法都使用了随机数，给定不同的随机种子，算法运行结果就不一样。因此，算法的性能必须在足够多的独立重复实验的基础上，在统计的意义上进行评估才是科学的。

在实验分析中，算法的性能是指算法在求解问题时需要的计算资源，主要指时间，与所给出的解的质量的关系。局部搜索算法一般要求在相同时间内比较解的质量和达到该解需要的时间。如果为了体现算法求解难解问题的能力，时间限制应该足够长，使得算法能够发挥其性能，在这样的条件下比较算法才有意义。如果时间限制太小，就有可能出现一些较弱的算法反而有好的结果。事实上，有些算法在短时间内能较快达到较好结果，但是之后算法就无法进一步改进其质量。当然，在其他应用场景下，可能有另外的时间限制和评价标准，但不管如何，我们认为算法研究必须采用客观的，统一的性能标准。

2.5 MVC局部搜索算法的实验分析

2.5.1 MVC基准实例

在MVC(MC,MIS)问题上，公认的测试实例有两组：DIMACS基准实例和BHOSLIB基准实例。其中DIMACS基准实例类型比较丰富，有结构化实例（图中带有某些特定结构）也有随机实例；而BHOSLIB基准实例则是一组以难解著称的随机实例。

DIMACS组织是由美国政府成立的离散数学及理论计算机科学中心DIMACS (discrete mathematics and theoretical computer science)，该中心是当前离散数学和理论计算机科学领域起到领导作用的研究中心。DIMACS于1993年

提出最大团基准图，以方便不同最大团算法之间进行比较，促进最大团算法设计。这些DIMACS基准图，提供了统一的最大团算法测试标准图 [38]。DIMACS基准图可以从网络下载 [58]，这些最大团测试数据集是迄今为止使用最多的最大团问题基准图。这些DIMACS基准图有从实际问题中编码成最大团问题实例的，比如来自编码理论，错误诊断和Keller猜想等；有从各种随机模型中产生的，如brock图。这些问题实例的规模从50个点和1000条边到大于5500个点和5百万条边。DIMACS组织也从这些实例里面选取了37个最具代表性的实例作为第二届DIMACS挑战测试问题(Second DIMACS Challenge Test Problems)，用于评估最大团算法性能。

BHOSLIB(Benchmarks with Hidden Optimum Solutions Library)基准图可以从网上下载到 [35]。这里共有40个例子,是目前国际公认最难的一组MVC，MIS和MC测试问题。这些实例是由著名的随机CS模型—RB模型 [59]在其相变参数设置产生得到。RB模型在相变区产生的实例已经从理论上 [60]和实验上 [61, 62]被证明是困难的。这些BHOSLIB基准图也因为其难度而闻名，被研究MVC和MC算法的学者们强烈推荐 [36, 63]。最近，这组基准图经常被用于评测新的MVC(MC,MIS)局部搜索算法的性能 [64]。这些基准图的规模从450个点到1534个点，它们的规模虽然不是很大，但是由于其特殊结构，目前仍然是最难的MVC(MC,MIS)实例。另外，除了40个普通的基准实例，还有一个4000个点的挑战实例，隐藏了一个3900个点的顶点覆盖（或等价地，100个点的独立集，或补图中100个点的团）。RB模型的作者猜测，这个实例在20年内不会有一个算法能在一个普通计算机上在一天时间内求出最优解。在我们的工作之前，对该实例找到的最优解是一个3903个点的顶点覆盖，而我们的算法找到了一个3902个点的顶点覆盖，刷新了该实例的求解纪录 [34]。

2.5.2 基本统计量

上文提到了，测量局部搜索算法的一个受到认可并且已经广泛使用的方法就是对每个实例上，算法运行多次独立重复实验，每次运行都给定一个时间限制，最后通过实验结果评估算法性能。从文献中看来，考虑到常用的基准实例中存在一些比较难的实例，对MVC(MC,MIS)问题实例比较合理的限制时间应该不小于1000秒。对于MVC(MC,MIS)问题的局部搜索算法实验分析，文献中比较常用的关于算法在一个实例上的基本统计量主要有以下几个。这里需要先介绍一下解质量，对于优化问题而言，一个解的解质量一般是指它对应的目标函数

值。

- 最优解质量：就是一个算法所有运行中找到的最好的解质量k^+。在比较若干个算法的时候，我们一般记最优的k^+为k^*。

- 平均解质量：算法每次运行就返回一个解，平均解质量就是一个算法所有运行找到的解质量的平均值。

- 最优解成功率：指的是一个算法找到k^*解的运行次数除以总的运行次数得到的比率。

- 平均成功时间：指的是一个算法找到k^*解的那些运行的平均运行时间，并不考虑失败运行的时间。

如果一个算法的k^+比其他算法都好，一般而言认为这个算法是最好的，当然运行的限制时间必须足够长，对于一些困难实例，每当一个算法能找到更优化的解时，都认为是该实例上更好的算法。

绝大多数情况下，参与评估的几个算法找到的最好的解都具备同样的质量，那么就考虑其他统计量，比如平均解质量，最优解成功率和平均成功运行时间。当算法在一个实例上的不同运行找到的解存在几种以上不同质量的时候，统计所有运行找到的解的平均质量也是个很好的衡量指标。不过，对很多MVC(MC,MIS)问题实例而言，算法运行结果往往只有最优解k^*和次优解k^*+1，这个时候就不适合用平均解质量比较算法性能，因为这个时候算法的平均解质量的差别比1还小，有时候会造成误解，认为算法性能差不多。

出现以上情况的时候，一般就考虑最优解成功率和平均成功运行时间。最优解成功率成功率明显较高的算法为较好的一个。在运行时间上，很多文献都使用了平均成功运行时间。Pullan等人在 [4]中比较正式地提出用平均成功运行时间来评估MC局部搜索算法。然而，这个统计量并不考虑失败运行的时间，因此很有可能出现一个算法成功率比较高，而同时它的平均成功运行时间也就较长。Pullan等人认为这种情况下两个算法是不可比的。但由于这种情况经常会发生的，因此这种评估方法不是很实用。问题就出现在平均成功时间这个统计量上。这里我们考虑一个情况：对于某个特定实例，算法A的成功率为50%，平均成功时间为100秒；算法B的成功率仅为1%，其平均成功运行时间为50秒。很明

显，应该是第一个算法占优。但是根据Pullan提出的方法却认为两个算法不可比。

我们认为平均成功运行时间不是一个能客观反映算法性能的统计量。当考虑算法的时间性能时，应该考虑所有运行的平均时间，其中对于失败的运行，其运行时间就是算法运行的时间限制，也是最长的，因此会使得平均时间变长，也就不会出现以上情况。更进一步，在计算平均时间的时候，对于失败的运行，应该对其运行时间进行处罚加长然后才进行统计。实际上，在著名的算法比赛SAT比赛里，对于失败的运行其运行时间在统计的时候要乘以10倍表示惩罚，这样使得成功概率较大的算法排名更靠前。

针对平均成功运行时间统计量的不足，除了通过比较平均运行时间来得到相对公正的评估，我们提出可以考虑算法的期望成功时间，也就是先算出算法成功需要的期望次数，算法成功需要的次数是一个几何分布，其期望L等于1除以成功概率。然后，算法的期望成功时间就等于$(L-1)*cutoff+ave_suc_time$。这就衡量了一个算法在实际应用中需要多长时间能找到一个最优解。

2.5.3 其它实验分析

Hoos等人提出了运行长度分布(RLD)和运行时间分布(RTD) [47]用于衡量算法在一个实例上性能，其优点是能全面的展示一个算法随着运行步数和运行时间的增长，找到的解的质量和成功率的变化。但是缺点是每张图只能反应一个实例，因此只适合对一些实例做分析。若对所有实例这样分析，不仅太耗时，论文空间也不允许。RLD和RTD已经被广泛应用于局部搜索算法随时间增加的行为分析上 [4, 65–67]。

对应地，Hoos等人还提出了对于特定算法而言一组实例的难度分布，以此衡量该算法在该组实例上的总体性能。难度分布主要是刻画随着允许的运行步数和运行时间的增长，能找到最优解的实例个数占该组实例个数比率的变化。

2.6 本章小结

本章对局部搜索算法做了一个相对全面但是很简单的介绍。我们首先介绍了局部搜索算法的基本概念，包括局部搜索算法的定义以及定义中的各个部分。接着区分了全局最优和局部最优，目标函数和评估函数，局部搜索算法与拟人拟物算法，单候选解局部搜索算法与多候选解局部搜索算法等多组概念。

然后介绍局部搜索算法的现状，常见的简单局部搜索算法框架，以及局部搜索算法的优缺点。最后介绍了局部搜索算法研究中的一些重要的观点，问题和方法，包括：局部搜索算法设计的要点，局部搜索中的集中性与多样性，循环问题，以及局部搜索算法的实验方法论，尤其是MVC局部搜索算法实验所采用的基准实例和实验分析方法。

第三章 EWLS算法

本章介绍求解MVC问题的边加权局部搜索(Edge Weighting Local Search，简记EWLS)算法 [34]。EWLS算法基于部分顶点覆盖的概念，并使用了一个边加权技术。其性能在大部分DIMACS难解实例上较之前已有算法更优，更在大部分BHOSLIB基准实例上占优，尤其是对一个挑战实例求出了更小的顶点覆盖，刷新了在该实例上自2007年创下的求解纪录。

3.1 概念和理论基础

本节首先介绍EWLS算法中基本的符号和概念定义，EWCC和NuMVC算法也采取了这套符号和定义；然后描述一个基本定理，EWLS和EWCC算法用到该定理来简化计算。

3.1.1 基本符号和概念

给定一个无向图$G = (V, E)$，一个候选解就是一个顶点子集$X \subseteq V$。我们说一条边$e \in E$被一个候选解X覆盖指的是边e至少有一个点属于点集合X。算法在搜索过程中维护一个当前候选解。为了方便起见，在下文中，我们用C来表示当前候选解。另外，算法还维护了一个边集合L来存储未覆盖的边。算法的每一个搜索步就是交换两个顶点：把一个点$u \in C$从C中移除，然后把另一个点$v \notin C$加入到C中。一个点v的状态$(state)$是一个布尔变量$s_v \in \{1, 0\}$，其中$s_v = 1$表示$v \in C$, 而$s_v = 0$则表示$v \notin C$。一个点的年龄(age)指的是从该点状态上一次改变之后发生的搜索步数。类似的，一条边的状态也是个布尔变量，其值1和0分别表示被C覆盖和不被C覆盖。一条边的年龄指的是从该边状态上一次改变之后发生的搜索步数。

3.1.2 边加权概念和定理

本文介绍的三个算法都用到了边加权技术，算法的评估函数也是基于这些边加权的概念。为了下文讨论算法的方便，这里介绍一些关于边加权技术的概念。一个边加权无向图由一个无向图$G = (V, E)$结合一个加权函数w构成，使得每条边$e \in E$都有一个正整数$w(e)$ 作为它的权值。我们用一个三元组(V, E, w)来表示一个边加权无向图，其中(V, E)是一个无向图。设w是图G的加权函数，则

一个候选解X的代价由公式

$$cost(G,X) = \sum_{e\text{不被}x\text{覆盖}} w(e)$$

计算，这个代价就是没有被X覆盖的点的边的总权值。我们把$cost(G,X)$定为算法的评估函数，代价越低的候选解越好。

对于一个顶点$v \in V$，我们定义

$$dscore(v) = cost(G,C) - cost(G,C')$$

其中当$v \in C$时$C' = C\backslash\{v\}$，否则$C' = C \cup \{v\}$。我们用$dscore(v)$来衡量改变顶点v状态的益处。很明显，当$v \in C$时$dscore(v) \leq 0$，当$v \notin C$时$dscore(v) \geq 0$。对两个顶点$u,v \in V$, 其中$u \in C$而$v \notin C$，我们定义

$$score(u,v) = cost(G,C) - cost(G,[C\backslash\{u\}] \cup \{v\})$$

衡量交换u和v的益处。

下面我们介绍一个定理，该定理阐述了$dscore$和$score$两者的关系。EWLS和EWCC就是根据该定理来计算交换一个顶点对的$score$值，以此简化计算。

定理 3.1: 对于一个无向图$G = (V,E)$，C是当前候选解,w 是边加权函数，对于一个顶点对$u,v \in V$, 其中$u \in C$且$v \notin C$，则有：当$e(u,v) \in E$时$score(u,v) = dscore(u) + dscore(v) + w(e(u,v))$；否则$score(u,v) = dscore(u) + dscore(v)$。

证明: 在这个证明中，全集为$E(G)$，$u \in C$且$v \notin C$。令

$E_0 = \{e|e\text{不被}C\text{覆盖}\}$,

$E_1 = \{e|e\text{不被}C\backslash\{u\}\text{覆盖}\}$,

$E_2 = \{e|e\text{被}u\text{覆盖但不被}C\backslash\{u\}\text{覆盖}\}$。

注意到$C = \{u\} \cup [C\backslash\{u\}]$，我们可以把$E_0$写为

$E_0 = \{e|e\text{不被}u\text{覆盖且不被}C\backslash\{u\}\text{覆盖}\}$。

则有$E_1 = E_0 \cup E_2$且$E_0 \cap E_2 = \emptyset$，所以

$$\begin{aligned} dscore(u) &= cost(G,C) - cost(G, C\backslash\{u\}) \\ &= \sum_{e\in E_0} w(e) - \sum_{e\in E_1} w(e) \\ &= \sum_{e\in E_0} w(e) - (\sum_{e\in E_0} w(e) + \sum_{e\in E_2} w(e)) \\ &= -\sum_{e\in E_2} w(e) \end{aligned}$$

令$E_3 = \{e|e$不被$C \cup \{v\}$覆盖$\}$,

$E_4 = \{e|e$被v覆盖且不被C覆盖$\}$。

而E_3可写为

$E_3 = \{e|e$不被v覆盖且不被C覆盖$\}$,

明显地，$E_0 = E_3 \cup E_4$且$E_3 \cap E_4 = \emptyset$。所以

$$\begin{aligned} dscore(v) &= cost(G,C) - cost(G, C \cup \{v\}) \\ &= \sum_{e\in E_0} w(e) - \sum_{e\in E_3} w(e) \\ &= \sum_{e\in E_0} w(e) - (\sum_{e\in E_0} w(e) - \sum_{e\in E_4} w(e)) \\ &= \sum_{e\in E_4} w(e) \end{aligned}$$

令$E_5 = \{e|e$不被$[C\backslash\{u\}] \cup \{v\}$覆盖$\}$, $E_6 = \{e|e$被v覆盖且不被$C\backslash\{u\}$覆盖$\}$。

类似地，我们把E_5展开为

$E_5 = \{e|e$不被v覆盖且不被$C\backslash\{u\}$覆盖$\}$,

则我们有$E_1 = E_5 \cup E_6$ and $E_5 \cap E_6 = \emptyset$,

$$\begin{aligned} score(u,v) &= cost(G,C) - cost(G, [C\backslash\{u\}] \cup \{v\}) \\ &= \sum_{e\in E_0} w(e) - \sum_{e\in E_5} w(e) \\ &= \sum_{e\in E_0} w(e) - (\sum_{e\in E_1} w(e) - \sum_{e\in E_6} w(e)) \\ &= \sum_{e\in E_0} w(e) - (\sum_{e\in E_0} w(e) + \sum_{e\in E_2} w(e) - \sum_{e\in E_6} w(e)) \\ &= \sum_{e\in E_6} w(e) - \sum_{e\in E_2} w(e) \end{aligned}$$

令$E_7 = \{e|e$被u覆盖，被v覆盖，且不被$C\backslash\{u\}$覆盖$\}$,

而$E_4 = \{e|e$不被u覆盖, 被v覆盖, 且不被$C\backslash\{u\}$覆盖$\}$,

那么我们有E_6=$E_4 \cup E_7$ 且$E_4 \cap E_7 = \emptyset$。所以

$$score(u,v) = \sum_{e\in E_4} w(e) + \sum_{e\in E_7} w(e) - \sum_{e\in E_2} w(e)$$

当$e(u,v) \in E$时，$E_7 = \{e(u,v)\}$,

$$\begin{aligned} score(u,v) &= \sum_{e\in E_4} w(e) + w(e(u,v)) - \sum_{e\in E_2} w(e) \\ &= dscore(u) + dscore(v) + w(e(u,v)) \end{aligned}$$

否则，$E_7 = \emptyset$，

$$score(u,v) = \sum_{e\in E_4} w(e) - \sum_{e\in E_2} w(e) = dscore(u) + dscore(v) \quad \square$$

3.2 部分顶点覆盖

部分顶点覆盖(Partial Vertex Cover)是EWLS算法一个很重要的概念。EWLS算法的思想就是找到一个比较优化的部分顶点覆盖，然后把它扩展为一个完整的顶点覆盖。它的形式化定义如下：

定义 3.1: 对于一个无向图$G = (V, E)$, 我们说一个大小为k的顶点子集$P \subseteq V$是一个(k,t)-部分顶点覆盖（$partial\ vertex\ cover$），简称(k,t)-PVC $(0 \leq t \leq |E|)$，如果P覆盖了G的$|E| - t$条边。

一个(k,t)-PVC可以用一个大小为k的顶点子集$P \subseteq V$和一个仅包含不被P覆盖的边的大小为t的边子集$L \subseteq E$来表示。显而易见，一个$(k,0)$-PVC就是一个大小为k的顶点覆盖，因为根据定义3.1，它覆盖了图的所有边。一般地，一个部分顶点覆盖与该图的最小顶点覆盖的规模有以下关系。

命题 3.2: 对于一个无向图$G = (V, E)$，G的一个(k,t)-PVC对该图的最小顶点覆盖的规模提供了一个$k + t$的上界。

证明: 根据部分顶点覆盖的定义，一个(k,t)-PVC 可以被扩展为一个最多含$k + t$的顶点覆盖，因为我们最多需要t个顶点来覆盖该PVC未覆盖的t条边。□

接下来我们举例说明PVC的概念以及$cost$和$dscore$的含义。

例: 在图3.1中，实心点表示在当前候选解C中的点，即被选为覆盖的顶点。图的每条边的边权标注在边上。当前候选解$C = \{v_3, v_5, v_6\}$就是一个(3,2)-PVC，因为$|C| = 3$且精确地有2条边($e(v_1, v_2)$ and $e(v_1, v_4)$)未被覆盖；C的代价为$cost(G, C) = w(e(v_1, v_2)) + w(e(v_1, v_4)) = 1 + 3 = 4$;$v_1$和$v_3$的$dscore$值分别为$dscore(v_1) = w(e(v_1, v_2)) + w(e(v_1, v_4)) = 1 + 3 = 4$和$dscore(v_3) = -w(e(v_2, v_3)) - w(e(v_3, v_4)) = -6 - 3 = -9$。

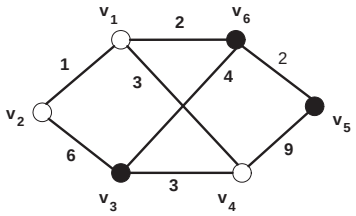

图 3.1　一个带边权的简单图

3.3 边加权

边加权策略在EWLS算法中扮演着很重要的角色。EWLS的边加权策略大意是这样的：每条边有一个正整数作为它的权值；在搜索过程中，每当陷入局部最优，算法就增加当时未被覆盖的边的权值。其实，EWLS的边加权策略和一个更一般的称为约束加权的策略很类似。约束加权技术作为一种多样性搜索策略 [68]，在SAT和CSP问题的局部搜索算法中被广泛应用，比如SAT局部搜索中的子句加权策略 [69–74]。从这个意义上讲，我们这个工作也为约束加权策略的有效性和一般性提供了更进一步的证据。

EWLS通过边加权策略增加了局部最优的代价，使得算法可以进一步找到隐藏在局部最优附近的更好的解。为了达到这个目的，EWLS首先将所有边权初始化为1；在搜索过程中，每次遇到一个局部最优的时候，EWLS对每条未覆盖的边，其边权加1，然后进行一个随机步跳到另一个起始点重新开始另一个搜索历程。

虽然COVER算法也用到了边权，但是EWLS和COVER的边加权技术是有区别的。COVER是一个迭代最优改进算法，且每一个搜索步都会更新边权，而EWLS是一个迭代局部搜索算法，且仅在遇到局部最优的时候才更新边权。

3.4 EWLS算法描述

EWLS的基本思想就是找到一个优化的部分顶点覆盖，然后扩展为完整的顶点覆盖。我们采用以下框架：每当找到一个提供更小上界的PVC时，EWLS就把它扩展为顶点覆盖并保存为算法已经找到的最优解。然后EWLS从当前的PVC里面移除掉一些顶点，接着以一个更小的PVC去继续寻找更好的上界。通过这种方法，MVC问题就转化为一系列这样的新问题：给定

一个图$G = (V, E)$和一个正整数k，要求寻找一个(k, t)-PVC，使得t最小化，也就是使得未覆盖的边最少。EWLS使用迭代局部搜索求解这个新问题：首先对初始候选解进行一个局部搜索阶段直到达到局部最优；然后扰动得到的局部最优解继而执行下一个局部搜索阶段。

基于以上考虑，我们给出EWLS算法的伪代码，如算法1所示。下面给出算法的详细描述。

算法开始时，EWLS创建两个集合L和UL。L是未覆盖的边的集合，而$UL \subseteq L$是L中在当前局部搜索阶段还没被函数$ChooseSwapPair$检查过的边的集合。这两个集合都初始化为边集E。其次，所有边权都初始化为1，并且计算每个顶点的$dscore$ 另外，为了构造当前候选解C，算法执行一个循环直至C成为一个顶点覆盖。该循环的每一步把$dscore$最大的顶点加入到C中（有多个这样的顶点时随机挑一个）。最后，最小顶点覆盖的上界ub初始化为$|C|$ 且已找到的最优解C^*初始化为C。每当找到一个新的上界，EWLS就从C中移除一些$dscore$最大的顶点直到$|C| = ub - delta$，其中$delta$是一个算法参数。这里我们要说明，在C中最大的$dscore$的绝对值最小，因为这里所有的$dscore$都是负的。

初始化之后，算法执行一个循环(第8-24行)直到搜索步数达到$maxSteps$。在循环中每一步，如果$ChooseSwapPair$ 函数能成功找到一对符合要求的顶点，则交换该顶点对来进行一个贪心步(第10行)。如果$ChooseSwapPair$失败了，意味着EWLS陷入局部最优，那么就对L中所有边的权值加1 (第12行)。更新完边权之后，算法通过交换C和$endpoint(L)$的各自一个随机点进行一个随机步(第13行)。算法具体是这样来选取$endpoint(L)$集合中的一个随机点的：首先随机选取L中一条边，然后随机选取该边的一个端点。EWLS在搜索过程记录了上一步加入C的点和上一步从C中移除的点，防止下一步马上对这两点进行反操作（第14-15行）。在每一步的最后，如果算法找到一个新的上界，则做一些更新(第16-23行)。首先，更新上界ub(第17行)和最优解C^*(第18-22行)。C^*的更新分两种情况：如果$L = \emptyset$，也就是说C已经是一个顶点覆盖了，则C^*直接更新为C；否则，EWLS构造一个顶点集合C^+来覆盖剩下的未覆盖的边，然后把C^*更新为$C \cup C^+$。在构造C^+的时候EWLS使用了一个简单的贪心策略，每一次就选能覆盖最多未覆盖边的点加入。最后，EWLS还更新C，把$dscore$最大的点移除掉直至$|C| = ub - delta$，以减小规模后的C去继续搜索更好地上界(第23行)。

Algorithm 1: EWLS

EWLS($G, delta, maxSteps$)
Input: graph $G = (V, E)$,delta(adjust size of C according to $|C| = ub - delta$), $maxSteps$
Output: vertex cover of G
begin
  $step := 0$; $L := E$; $UL := E$;
  initialize all edge weights as 1 and compute $dscores$ of vertices;
  construct C greedily until it's a vertex cover;
  $ub := |C|$; $C^* := C$;
  remove vertices with the highest $dscore$ from C until $|C| = ub - delta$;
  while $step < maxSteps$ **do**
    if $((u, v) := ChooseSwapPair(C, L, UL)) \neq (0, 0)$ **then**
      $C := [C \backslash \{u\}] \cup \{v\}$;
    else
      $w(e) := w(e) + 1$ for each $e \in L$;
      $C := [C \backslash \{u\}] \cup \{v\}$ where $u := random(C)$ and $v := random(endpoint(L))$;
    $tabuAdd := u$;
    $tabuRemove := v$;
    if $|C| + |L| < ub$ **then**
      $ub := |C| + |L|$;
      if $L = \emptyset$ **then**
        $C^* := C$;
      else
        construct C^+ greedily that covers L;
        $C^* := C \cup C^+$;
      remove vertices with the highest $dscore$ from C until $|C| = ub - delta$;
    $step := step + 1$;
  return C^*;
end

3.5 算法进一步说明

本节给出EWLS算法更多的详细说明。我们首先介绍$ChooseSwapPair$函数，EWLS就是利用该函数挑选交换点对的。我们还介绍了数据结构L和UL，以及它们的维护细节。另外我们还给出了调整当前PVC大小时移除顶点的策略的直观解释。最后我们总结了EWLS的集中性搜索和多样性搜索之间的平衡。

3.5.1 $ChooseSwapPair$函数

EWLS算法最重要的部分就是$ChooseSwapPair$函数。它的伪代码如算法2所示。

Algorithm 2: function ChooseSwapPair

```
1  ChooseSwapPair(C,L,UL)
   Input: current candidate solution C, uncovered edge set L, edge set UL of
          uncovered edges unchecked in the current local search stage
   Output: a pair of vertices
2  begin
3      if S := {(u,v)|u ∈ C, v ∈ endpoint(e_oldest),
       u ≠ tabuRemove, v ≠ tabuAdd and score(u,v) > 0} ≠ ∅ then
4          return random(S) ;
5      else
6          foreach e ∈ UL, from old to young do
7              if S := {(u,v)|u ∈ C, v ∈ endpoint(e),
               u ≠ tabuRemove, v ≠ tabuAdd and score(u,v) > 0} ≠ ∅ then
8                  return random(S);
9      return (0,0);
10 end
```

$ChooseSwapPair$函数选择符合下面条件的一对顶点：$u \neq tabuRemove$，$v \neq tabuAdd$且$score(u,v) > 0$，其中$u \in C$ and $v \in endpoint(L)$。当选择加入C的顶点v时，它优先选择更老的边的端点。具体地，它首先检查L中最老边e_{oldest} (图3.2中$e_{i|L|}$)。如果存在满足条件的点对(u,v)(第3行)，函数随机地返回一个这样的点对。否则，它按照从老到新的顺序检查UL中的边。如果检查到边$e \in UL$时，存在满足条件的点对(u,v)(第7行)，则随机地返回一个这样的点对。最后，如果函数不能找到符合条件的点对，则返回(0,0)。

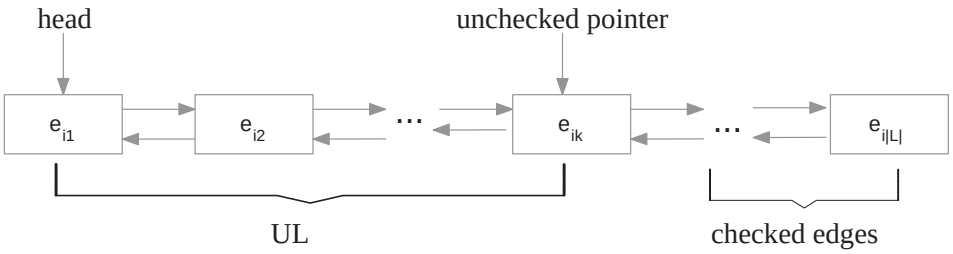

图 3.2　数据结构L和UL

3.5.2 维护L和UL列表

为了维护未覆盖边集合L，我们使用了一个双向链表。在搜索过程中，当一条边变成未覆盖状态时，它被插入到表头。因此，L中的边从表头到表尾其年龄是递增的。数据结构$UL \subseteq L$保存那些在当前局部搜索阶段还没被函数$ChooseSwapPair$检查过的未覆盖边。在每个局部搜索阶段起始时，UL被重新设为L，然后随着搜索的进行，我们用一个指针来把UL从L中区分出来，如图3.2所示。注意到当一条边变成未覆盖并插入链表时，即使它之前被检查过，它因为会插入在表头而属于UL。所以在本实现中，UL实际上是包含那些在最后一次插入L之后未被覆盖过的边。

3.5.3 调整PVC大小

当找到最小顶点覆盖规模的一个新上界时，EWLS会通过移除C中那些$dscore$最大的顶点来减小当前PVC的大小直至$|C| = ub - delta$。

有人可能注意到一个更贪心调整C的办法是从C^*中移除$dscore$最大的顶点直至$|C| = ub - delta$，剩下的顶点则构成新的C。其实，从C中移除点也足够贪心了。我们在下面说其理由。当找到一个新上界($|C| + |L| < ub$)时，因为在搜索过程我们始终保持$|C| + delta = ub$，所以有$|L| < delta$。在后面的章节我们会通过实验显示$delta$一般是一个很小的整数(经常不超过4)。所以，当找到一个新上界时，L只包含很少的边。因此，这些未覆盖边里面基本不可能有公共顶点，也就是说，C^+包含$|L|$个顶点，每个顶点覆盖一条未覆盖边。这样的话，从C^*中移除一个属于C^+的点仅使得一条边变成未覆盖，这表明C^+中的点具有较高的$dscore$(绝对值较小)。所以，为了方便，我们直接从C中移除$dscore$最高

的顶点直至$|C| = ub - delta$，这可以看成是以下两个过程的合并：(1) 从C^*中移除所有属于C^+的点；(2)从C^*剩下的点(其实就是C)移除$dscore$最大的点，直到剩下$(ub - delta)$个点，也就是新的当前PVC。

3.5.4 集中性和多样性的平衡

EWLS通过各种技术在搜索的集中性和多样性直接取得了巧妙的平衡。在贪心步上，候选交换点对都是根据贪心策略选出来的，但从这些候选点对中选取一个点对真正进行交换则是随机的。另外，由老到新的搜索策略使得EWLS优先覆盖较老的未覆盖边，这就保持L更新的频繁，由此每个搜索阶段的搜索范围都足够广使得算法可以找到高质量的局部最优。EWLS还加入了随机步来增加搜索的多样性。最后，边加权技术动态改变搜索空间的代价分布，使得算法不容易陷在一个小区域，这也是一种多样性的体现。

3.6 实验分析

本节通过实验评估EWLS算法在DIMACS和BHOSLIB这两个标准的MVC(MC,MIS)基准实例上的性能。在每一组基准实例上，我们把EWLS的性能与当时最好的算法做比较。

3.6.1 实验设置

在介绍实验结果之前，我们先介绍实验的相关设置。

- **实现：** EWLS由C++语言实现；其他用于比较的算法也是由C++语言实现。所有的算法的源程序都用g++编译器带O2选项编译。
- **实验平台：** 所有实验都在一台个人计算机上运行，其配置是3 GHz Intel Core 2 Duo CPU E8400和4GB RAM，操作系统是Linux。为了得到该机器的性能，我们在该机器上运行了DIMACS机器基准实例 [75]。这台机器求解r300.5实例需要0.19秒，求解r400.5实例需要1.12秒，求解r500.5实例需要4.24秒。
- **停止标准：**对于DIMACS实例，每次运行允许最多步数$maxSteps$为10^8，而对于BHOSLIB实例，由于其难解性，最多步数$maxSteps$为4×10^8。关于给定相同时间限制下的实验结果，在下一章我们将把EWLS和EWCC与其他算法做比较并给出性能结果。这里的实验以步数限制为停止条件，实

验结果和算法的实现优劣无关，有助于看清算法的步数复杂度。另外，因为实验中的几个算法的单步复杂度在同一个数量级，所以一些性能相差很明显的实验结果还是能说明算法的在那些实例上的优劣的。

- **实验报告方法：** 对于每个实例，每个算法都执行100次不同随机种子的独立运行。对每个实例，我们报告以下信息：最小（或已知最小）顶点覆盖的大小(k^*)；每个算法成功找到k^*解的次数("suc")；每个算法的平均成功时间("time")，表格中的时间以秒为单位；以及EWLS的$delta$参数的值(d)。每个实例上最优算法的结果的字体加黑表示。

3.6.2 实验结果

本节分别阐述算法在DIMACS和BHOSLIB两组基准实例上的实验结果。

DIMACS基准实例实验结果

在DIMACS基准实例上，我们把EWLS算法和PLS以及COVER算法做比较。其中，COVER算法的代码是从网上(http://www.informatik.uni-freiburg.de/~srichter/)下载的。COVER本身是个求解顶点覆盖判定问题的算法，我们运行的是它的迭代版本，也就是可以求解MVC的版本。而PLS是由它的作者Pullan提供给我们的。

表3.1显示了在DIMACS基准实例上的实验结果。大部分DIMACS实例对时下的求解器都太容易了。那些没有在表格出现的实例都被几个算法在2秒之内以100%的成功率求出最优解。为了得到有意义的比较，$C2000.9$ 的k^*设置为1921而不是已经知道的最优解1920。实际上，目前已有的最优解1920个点的顶点覆盖是在10^9为步数限制下求得的 [63]，文献中并没有任何算法能在10^8内求出1920的解。

实验结果显示EWLS和PLS在这些DIMACS实例上平分秋色，而COVER在这些实例上则显得较弱。在评测的37个DIMACS实例上，EWLS找到k^*解的有35个，比其他两个算法多（PLS是34个，COVER是32个）。PLS主要是在$brock$实例占优，而EWLS则在多组不同实例占优。EWLS占优的实例里面有三个出名的困难实例($C2000.9$，$MANN_a45$和$MANN_a81$)，在这些难解实例上，PLS不能找到一个k^*解。特别地，在公认的最难解实例$MANN_a81$上，EWLS有7次运行找到了k^*解，但是PLS只找到了大小为k^*+2的解，而COVER只有5次能找到k^*+1的解。这说明EWLS在$MANN$实例上的性能比

Graph		EWLS			PLS		COVER	
Instance	k^*	d	suc	time	suc	time	suc	time
brock400_2	371	1	2	40.902	**100**	0.111	0	n/a
brock400_4	367	1	96	56.816	**100**	0.031	2	242.025
brock800_2	776	1	0	n/a	**100**	7.157	0	n/a
brock800_4	774	1	0	n/a	**100**	1.918	0	n/a
C2000.9	1921	1	**18**	327.875	0	n/a	0	n/a
C4000.5	3982	1	100	686.472	**100**	43.886	100	658.33
keller6	3302	1	**100**	4.934	36	161.568	100	68.214
MANN_a45	690	1	**56**	68.069	0	n/a	40	171.823
MANN_a81	2221	3	**7**	115.751	0	n/a	0	n/a
p_hat1500-1	1488	1	100	13.587	**100**	0.961	100	18.095

表 3.1　EWLS和其他算法在DIMACS基准实例的比较

其他算法明显好很多。

不过，EWLS和COVER都在$brock$实例上表现很弱。$brock$实例是人工构造的实例，这些图的最小顶点覆盖（或者补图的最大团）都是由度数较低的顶点构成的，目的就是为了使得贪心启发式失效。实际上，绝大部分算法比如GRASP，RLS，和k-opt采用的策略都会使得度数较高的点被选择的概率较大，他们在$brock$实例上也都表现的很弱。PLS之所以在这些实例上面表现那么好，是因为PLS有三个子算法，其中一个就是优先考虑度数低的顶点的。

BHOSLIB基准实例实验结果

在DIMACS基准实例上，我们把EWLS算法和PLS-MVC以及COVER算法做比较。因为我们没有得到PLS-MVC的求解器，所以PLS-MVC的实验结果是从其文献 [33]中的结果按机器的速度比规约到我们实验机器上的结果。

表3.2显示了在BHOSLIB基准实例上的实验结果。两组小实例$frb30$和$frb35$上的每个实例都被几个算法在2秒之内以100%的成功率求出最优解，因此没有在表格中报告。根据使用BHOSLIB测试集的算法文献 [64]，EWLS是当时在该组基准实例上以100%成功率求出最优解的实例个数最多的(40个实例中有29个)，并且在大部分实例上都花费的时间更少。

实验结果表明，不管从解的质量，成功率，还是运行时间，EWLS在大部分BHOSLIB实例上都是性能最好的算法，尤其在大实例上更明显。$frb40$之后的实例基本全部都是EWLS占优，除$frb50$-23-3，$frb53$-24-4和$frb56$-25-1三个

Graph		EWLS			PLS-MVC		COVER	
Instance	k^*	d	suc	time	suc	time	suc	time
frb40-19-1	720	2	**100**	0.701	100	1.629	100	1.575
frb40-19-2	720	2	**100**	16.855	100	26.129	100	17.180
frb40-19-3	720	3	100	3.526	**100**	2.929	100	5.061
frb40-19-4	720	3	100	14.029	100	15.306	**100**	11.794
frb40-19-5	720	2	100	84.878	**100**	68.840	100	124.153
frb45-21-1	900	4	**100**	13.963	100	19.656	100	14.427
frb45-21-2	900	2	**100**	25.842	100	47.148	100	37.692
frb45-21-3	900	2	**100**	93.172	100	204.937	100	109.831
frb45-21-4	900	2	**100**	18.078	100	28.246	100	21.888
frb45-21-5	900	4	**100**	52.913	100	73.689	100	105.138
frb50-23-1	1100	2	**100**	262.313	87	479.845	100	267.752
frb50-23-2	1100	2	**61**	535.889	59	622.067	53	730.832
frb50-23-3	1100	1	42	692.21	22	786.944	47	967.853
frb50-23-4	1100	1	**100**	31.573	100	37.682	100	32.734
frb50-23-5	1100	2	**100**	117.730	100	132.791	100	167.929
frb53-24-1	1219	4	**29**	885.421	9	891.715	19	994.628
frb53-24-2	1219	3	**62**	659.381	41	933.330	62	946.14
frb53-24-3	1219	2	**100**	165.720	86	541.244	100	281.189
frb53-24-4	1219	2	51	783.279	**59**	593.589	59	1099.910
frb53-24-5	1219	2	**100**	250.262	94	540.461	99	416.047
frb56-25-1	1344	3	29	773.586	14	809.778	**34**	1255.851
frb56-25-2	1344	2	**32**	730.987	10	806.611	22	1230.623
frb56-25-3	1344	4	**100**	307.964	18	1028.537	99	536.067
frb56-25-4	1344	4	**100**	234.376	89	695.234	98	466.007
frb56-25-5	1344	3	**100**	94.242	96	428.927	100	168.166
frb59-26-1	1475	2	**21**	1015.145	2	748.640	21	1421.241
frb59-26-2	1475	3	**18**	1116.251	2	272.768	11	1149.700
frb59-26-3	1475	3	**44**	726.424	24	934.994	38	1765.950
frb59-26-4	1475	4	**19**	900.275	14	1003.104	6	2069.310
frb59-26-5	1475	1	**100**	334.044	97	472.659	100	478.587

表 3.2　EWLS和其他算法在BHOSLIB基准实例的比较

实例外。对于frb50-23-3，实际上没有明显的占优算法。COVER算法的成功率比EWLS高出5%，但平均时间是EWLS的1.4倍。为了比较EWLS和COVER在这个实例上的性能，我们把COVER的$maxSteps$减少，使得它刚好达到EWLS的成功率(42%)，这时它的平均时间是775.75秒，仍然比EWLS的平均时间要高。EWLS的优秀性能从它和其他算法在规模和难度都是最大的一组实例frb59上的性能差距更能显示出来。结果毫无疑问地表明EWLS给出了当时BHOSLIB上的最佳性能。

3.6.3 挑战实例的新纪录

对于挑战实例frb100-40，该实例的最小顶点覆盖大小为3900，BHOSLIB实例的设计者猜测，这个实例在20年内不会有一个算法能在一个普通计算机上在一天时间内求出最优解。在我们的工作之前，对该实例的求解纪录是COVER算法于2007年创下的，它求得的最优解是3903个点的顶点覆盖。具体地，COVER运行了100次，其中25次找到了3903个点的顶点覆盖，中值运行时间是1193.92秒，运行机器是一个2.13GHz/2GB的个人计算机。

我们在这个实例上运行EWLS($delta = 6$)100次，给定时间限制为4800秒。其中4次运行找到了一个3902个点的顶点覆盖，也即一个98个点的独立集，平均时间为2823.05秒，最快的一次是1239.87秒；另外的96次运行中，59次找到了3903个点的顶点覆盖，其他37次都找到大小为3904 的解。关于我们的这个新纪录，在BHOSLIB基准实例的网页上也有报告 [35]。

这里我们报告EWLS找到的frb100-40的一个98个点的独立集：
5, 54, 113, 145, 177, 212, 253, 293, 331, 366, 439, 470, 512, 528, 562, 618, 656, 694, 744, 787, 832, 868, 891, 941, 964, 1008, 1076, 1094, 1149, 1181, 1238, 1241, 1282, 1348, 1390, 1416, 1474, 1490, 1547, 1578, 1623, 1664, 1681, 1722, 1786, 1806, 1844, 1890, 1955, 1999, 2040, 2046, 2116, 2130, 2188, 2216, 2244, 2326, 2362, 2421, 2480, 2516, 2558, 2589, 2608, 2679, 2698, 2743, 2788, 2820, 2878, 2885, 2935, 2993, 3026, 3078, 3084, 3152, 3213, 3241, 3299, 3357, 3373, 3429, 3444 3515, 3538, 3600, 3638, 3648, 3705, 3760, 3762, 3834, 3875, 3895, 3928, 3994.

3.7 讨论

本节讨论EWLS算法各种因素的有效性或对算法性能的影响。我们讨论的因素包括：边加权策略，$delta$参数，以及从$ChooseSwapPair$中扫描边的策略。

3.7.1 边加权策略的有效性

我们从EWLS中去掉边加权策略，把得到的算法叫EWLS$_0$。EWLS$_0$和EWLS算法除了不更新权值(删除算法1中的第12行)，在其他方面完全一样，也就是说，每条边的权值一直是1，不会改变。我们通过比较EWLS$_0$和EWLS的性能来说明边加权策略的有效性。

算法的性能一般在大实例和困难实例才能真正区分出来。我们在几个比较难的实例上运行EWLS$_0$($delta$参数和EWLS一样)，比较结果如表3.3所示。其中"CT"表示时间限制，单位为秒。结果表明，在这些困难实例上，EWLS$_0$的性能显着地弱于EWLS。我们也在frb100-40上运行了EWLS$_0$，结果有12次运行找到了3903个点的顶点覆盖，并没有任何一次运行能找到3902个点的解。基于这些结果，我们得出结论边加权技术在EWLS 算法中起到一个关键作用。

Graph			EWLS			EWLS$_0$		
Instance	k^*	CT	suc	time	steps	suc	time	steps
C2000.9	1921	2400	**21**	557.745	66666972	1	53.610	8022464
keller6	3302	1000	**100**	4.934	226592	70	456.159	16690123
frb53-24-1	1219	2000	**25**	745.425	129528395	1	560.550	136983373
frb56-25-1	1344	2200	**28**	742.227	124088139	5	675.348	148886230
frb59-26-1	1475	2400	**21**	1015.145	171236143	1	1137.620	231604550

表 3.3　EWLS和EWLS$_0$的性能比较

3.7.2 $delta$参数

EWLS中的$delta$参数必须在运行时输入。这个参数通过控制当前候选解的大小来影响EWLS算法的性能。具体地说，EWLS是根据等式$|C| = ub - delta$来调节当前候选解C的大小的。

我们在4个DIMACS实例和4个BHOSLIB实例上通过实验调研$delta$参数对EWLS的影响。对DIMACS基准实例，我们挑选了$brock$400_4，C2000.5，$MANN_a$81和p_hat1500-1，因为它们来自不同的产生模型并且有适当的难度，而且对它们的最优$delta$参数值也覆盖了所有出现在DIMACS的$delta$参数值。对BHOSLIB基准实例，我们挑选了frb50-23-3，frb53-24-2，frb56-25-3和frb59-26-1，因为它们有不同的规模并且有适当的难度，而且对它们的最优$delta$参数值也覆盖了所有出现在BHOSLIB的$delta$参数值。我们对这些实例在不同的$delta$值下运行EWLS。实验结果如表3.4所示，表格中的单步时间单位为10^{-6}秒。

Instance	d	suc	time	steps	time/step	Instance	d	suc	time	steps	time/step
	1	100	74.146	27107873	2.73		**1**	42	692.210	155456585	4.45
	2	92	53.106	14911288	3.56		2	31	792.994	162250210	4.89
brock400_4	3	0	n/a	n/a	n/a	frb50-23-3	3	0	n/a	n/a	n/a
	4	0	n/a	n/a	n/a		4	0	n/a	n/a	n/a
	5	0	n/a	n/a	n/a		5	0	n/a	n/a	n/a
	1	100	2.471	73177	33.76		1	40	579.803	126594742	4.58
	2	100	4.109	113269	36.11		2	51	540.416	105610398	5.11
C2000.5	3	100	17.181	452588	37.96	frb53-24-2	**3**	58	603.768	109004768	5.54
	4	100	157.638	4109690	38.36		4	48	663.545	110960787	5.98
	5	29	6.871	173373	39.63		5	44	816.342	125398250	6.51
	1	11	934.792	326850188	2.86		1	90	692.210	110050260	4.45
	2	18	510.480	181665726	2.81		2	92	443.208	84831320	5.22
MANN_a81	**3**	22	621.683	207424642	3.00	frb56-25-3	3	100	359.264	64732310	5.55
	4	2	685.240	234671242	2.92		**4**	100	307.964	51838431	5.94
	5	8	440.686	142616979	3.09		5	99	291.235	45923315	6.32
	1	100	13.587	419374	32.39		1	18	1114.757	190556780	5.85
	2	56	9.396	277012	33.92		**2**	21	1116.251	179661855	6.21
p_hat1500-1	3	22	7.315	209301	34.95	frb59-26-1	3	19	1315.865	201295040	6.54
	4	51	11.502	329956	34.86		4	0	n/a	n/a	n/a
	5	0	n/a	n/a	n/a		5	0	n/a	n/a	n/a

表 3.4　EWLS在不同$delta$下的性能

EWLS在$delta$参数不同值下对代表性实例的性能如表3.4所示。结果表明，$delta$参数对EWLS的性能有很大影响。不同实例上需要的最优$delta$参数值不一样；所以，为了获得好的性能必须进行一些手工调试来找出每个实例需要的最优$delta$参数值。我们从实验中还惊奇地发现，对于同一个实例，在不同随

机种子下需要的最优$delta$参数值也是不一样的。所以，靠手工调试来确定每个实例的$delta$参数值还不可靠，最好是在搜索过程中自动调整$delta$参数的值。这不仅可以消去$delta$参数，还可以提高算法的鲁棒性。从表3.4中我们观察到了一些现象，或许对自动调整$delta$参数有启示。

- 随着$delta$值的增加，算法的单步复杂度增加；在某些情况下，适当增加$delta$值可以加速收敛到最优解的速度从而也提高成功率。
  直观上可以这样解释：根据等式$|C| = ub - delta$，$delta$值更大则导致C更小，也就剩下更多未覆盖边，即L更大。而$ChooseSwapPair$函数正是通过扫描未覆盖边集合L来找到候选交换点对。所以，L更大就意味着单步复杂度会更大，并且也意味着有更广的搜索空间，可以提高局部最优值的质量。
- 当$delta$太大时(依赖于实例，但一般大于5的$deltas$都太大了)，EWLS算法会失效。
  对这个现象的一个直观解释是这样的：C^+是简单构造的(请见算法1)，而不是最优构造。即使C和C^+都是最优的，如果C和C^+都具备了一定的规模，$C \cup C^+$也不太可能是一个最优解，因为他们之间不是相互独立的，不能通过简单合并来得到一个最优解。

以上观察提供了一些关于$delta$参数如何影响EWLS性能的见解，这也有助于自动调整该参数。

3.7.3 从老到新的边扫描策略

当EWLS算法扫描未覆盖边集L并从中寻找加入当前候选解的顶点时，算法采用了一个特殊的从老到新的扫描策略。很多成功的局部搜索算法也通过优先选择老的解部件来增加算法的多样性搜索，比如SAT局部搜索算法HSAT [76]，Novelty [52]和Novelty++ [54]等，而COVER算法 [23]优先选择最老的点来加入当前候选解。

不过，和其他算法中的优先选择老的解部件的启发式不一样，EWLS的这个从老到新的策略比较特殊，它不是严格地按照从老到新的顺序。EWLS首先检查L中最老的边，如果没找到可以交换的点对就检查UL中的边，而不是检查L剩下的所有其他边。这个特殊的策略是基于这样的猜想：已经检查过的不符合条件的边在当前局部搜索阶段接下来的搜索步会以很大的概率仍然是不符

合条件，因此扫描$L\backslash UL$的边则是一种效用不大的搜索，可以被跳过来节省时间。为了取得优先考虑老边(可以提高算法的步性能)和降低单步时间复杂度的平衡，于是我们首先检查L中最老的边然后检查UL中的边。

为了验证上面的猜想，我们做实验比较EWLS和它两个其他版本EWLS_1以及EWLS_2的性能。其中EWLS_1是严格地按照从老到新的顺序检查L中所有的边；而EWLS_2则只检查UL的边，也按照严格的从老到新的顺序。

Instance	k^*	CT(s)	Algorithm	suc	time	steps	time/step
brock200_4	183	1000	EWLS	100	3.184	1696748	1.88
			EWLS_1	100	2.657	1362619	*1.95*
			EWLS_2	100	3.667	*2037874*	1.80
C1000.9	932	1000	**EWLS**	100	3.326	896259	3.71
			EWLS_1	100	4.536	1054497	*4.30*
			EWLS_2	100	3.819	*1091106*	3.50
keller6	3302	1000	EWLS	100	4.943	*226592*	21.81
			EWLS_1	100	5.832	216712	*26.91*
			EWLS_2	100	4.164	198643	20.96
MANN_a45	690	2400	**EWLS**	100	194.969	124004605	1.56
			EWLS_1	100	241.581	148208716	*1.63*
			EWLS_2	100	232.121	*150503543*	1.54
p_hat1500-1	1488	1000	**EWLS**	100	13.587	419374	32.39
			EWLS_1	100	15.211	394754	*38.53*
			EWLS_2	100	18.215	*552802*	32.95
frb45-21-1	900	1000	EWLS	100	14.594	3224649	4.53
			EWLS_1	100	13.386	2413533	*5.54*
			EWLS_2	100	18.068	*4417659*	4.09
frb50-23-3	1100	1800	**EWLS**	42	692.210	155456585	4.45
			EWLS_1	30	748.632	150327876	*4.98*
			EWLS_2	35	761.672	*179217109*	4.25
frb53-24-3	1219	2000	**EWLS**	100	165.720	32678443	5.07
			EWLS_1	100	218.007	36825526	*5.92*
			EWLS_2	100	178.661	*38175477*	4.68
frb56-25-4	1344	2200	**EWLS**	100	234.376	39241213	5.97
			EWLS_1	100	321.202	39851453	*8.06*
			EWLS_2	100	378.561	*68579975*	5.52
frb59-26-4	1475	2400	**EWLS**	18	841.930	125795679	6.69
			EWLS_1	7	1281.137	*142665611*	*8.98*
			EWLS_2	13	696.889	114057238	6.11

表 3.5　EWLS在不同的L扫描策略下的性能

表格3.5中的单步时间复杂度的单位为10^{-6}秒，黑体字表示性能最好的算法，斜体字表示性能最差的算法。这次实验采用了5个DIMACS实例和5个BHOSLIB实例。对于DIMACS实例，我们选择$brock$200_4，C1000.9，$keller$6，$MANN_a$45和p_hat1500-1，因为它们来自不同的产生模型并且难度适中。对于BHOSLIB实例，我们选择frb45-21-1，frb50-23-3，frb53-24-3，frb56-25-4和frb59-26-4，因为它们来自不同组并且难度适中。从表3.5我们观察到以下现象，从某种程度上支持了我们的猜想。

- 总的来说，EWLS在这些实例上优胜于EWLS_1和EWLS_2。
- EWLS和EWLS_1的步数性能比较接近，都比EWLS_2好很多；EWLS_2 除了$keller$6和frb59-26-4两个实例外，在其他实例上步数性能最差。
- EWLS和EWLS_2的单步时间复杂度比较接近，都比EWLS_1好很多；EWLS_1在所有实例上单步消耗的时间都最多。
- 对于一些实例，如p_hat1500-1，frb45-21-1和frb56-25-4，重新检查L中最老的边(如EWLS所采取的) 可以在步数性能上得到很大的提高(相对于EWLS_2)，然而重新检查更多的已经检查过的边(如EWLS_1所采取的)则在步数性能上得到的提高已经不明显了。这里体现了局部搜索策略实现的严格程度带来的效益呈边际递减效应。一旦我们体现了从老到新的思想，算法性能就可以获得明显的提高；然而在那以后，即使我们把从老到新的思想进行到底，算法也很难再获得明显的提高。

3.8 EWLS算法在社会网络研究中的应用

在理解社会网络演化原理的基础上，可以设计一个捕获真实世界社会网络特征的生成器。本节基于“团叠加”的社会网络演化原理，利用EWLS算法设计了一个社会网络生成器(也称为图生成器、生成模型等) [77]。该生成器通过模拟社会网络“团叠加”演化行为，产生符合社会网络特征的人工网络。该生成器可以帮助我们进一步了解社会网络演化原理，并能一定程度上，对社会网络的演化行为进行预测。

3.8.1 社会网络的“团叠加”演化

社会网络是如何形成和演化的呢？一个实例可以帮助理解其演化过程。一个研究机构(研究团队)可以看做是一个社会网络平台。研究者可以通过加入

研究机构扩展他们的社会网络，从而促进交流和合作。假如一个研究者A，想加入一个研究机构T，在加入之前，A与T的成员互不相识。A首先要做的是获取加入T的资格。当A成功获得了加入T的资格，即可将A视为T的一员。此时，从社会网络的观点来看，A作为一个孤立点，加入了T表征的社会网络。随后，A可以通过介绍、讨论、合作等方式同T的成员建立联系，从而获取新的社会链接，扩展了自己的社会网络，同时也从整体上改变T的社会网络结构。

在以上所述的研究者实例中，不仅可以得出社会网络的演化过程，还可以学习到两个重要的推论，即社会网络的演化原理。第一，社会网络是来自于附属网络。附属网络是个二部网络，其中一维是行动者集合，另一维是事件集合。行动者参与事件对应于行动者集合中结点到事件集合中结点的边，表征行动者附属于某个事件。而社会网络就是这个附属网络在行动者维的投影。第二，社会网络以“团叠加”形式进行演化。在上面的例子中，参与一个讨论的研究机构成员相互认识(即使当初不认识，也能通过讨论而认识)，即两两之间有边相连，这意味着所有讨论成员构成了一个团。如果一个新研究者加入该讨论，并和所有的讨论成员相识，那么他连同其他所有成员将形成一个更大的团。假设新研究者以前加入过其他团，那么这些团在新研究者结点上，通过新研究者加入讨论这个行为，进行了叠加。因此，如果将一个结点视为一个1-团，两个相连的结点视为一个2-团，那么一个社会网络的演化呈“团叠加”行为。

3.8.2 “团叠加”生成器的构建

本节的目标是设计一个带权无向图生成器，生成器生成的网络能够拥有现实社会网络中观察到的结构属性。

社会网络“团叠加”演化原理是本章建模工作的理论核心。给定一个原始图G(至少包含两个结点)，和一系列到来的新结点，图演化过程可以分为两种不同的行为：点演化或边演化。概率参数p_{node}决定一步演化过程是点演化还是边演化。

在点演化步骤中，仅仅将一个新结点加入到图中。点演化对应于一个行动者加入了一个社会网络扩展平台，但没有与其他社会成员作进一步的接触。这个新结点只可能在以后的边演化步骤中获得链接。要么被选作“新成员”加入到其他团而获得链接，要么被选作规模为1的团，与“新成员”产生链接。

在边演化步骤中，首先在图中随机选择一个结点作为“新成员”，然后在图中随机选择一个k-团(规模为k的完全子图)。考虑到一个“新成员”不太可能和团所有成员成为朋友，设置参数p_{sample}用于决定“新成员”是否和所有团成员成为朋友。以概率p_{sample}，在搜索到的k-团中，随机采样一个小团，规模在0到k之间。连接“新成员”和小团的所有成员；以概率$1-p_{sample}$，连接“新成员”和k-团所有的成员。如果“新成员”和团成员已经有了联系，那么他们之间的边权值加1。如果在图中找不到k-团，模型在这步边演化过程中不再做任何动作，继续循环。模型算法的伪代码如算法3所示。模型算法的核心findClique函数，就是调用了EWLS寻找补图的独立集从而求得原图的团的。

Algorithm 3: Clique-Superposition Model

Input: p_{node}, p_{sample}
Output: G

```
1  begin
2      Graph G := new_graph();
3      while G.getNodes() < N do
4          if Random() < p_node then
5              Node current := newnode();
6              G.add_node(current);
7          else
8              Node newmember := G.getRandomNode();
9              k := getCliqueSize();
10             Clique C := findClique(G; k);
11             if C ≠ NULL then
12                 if Random() < p_sample then
13                     Clique C' := Sample(C);
14                 else
15                     Clique C' := C;
16                 for i := 1 to Size(C') do
17                     G.addEdge(newmember, C'.node[i]);
18     return G;
19 end
```

3.8.3 实验分析

本节的工作是对真实世界社会网络结构模式进行研究。从广义社会网络的角度出发，本章研究所需的数据集必须是附属网络，并且它们的行动者

投影网络具有社会网络的意义。“论文著作网络”是一个理想的数据集。在DBLP可以公开下载。此外，社会书签网站，例如CiteULike 和Delicious，提供了丰富的附属网络资源，比如，“用户-资源”附属网络和“用户-标签”附属网络。从Web2.0网站上获取了三个大规模附属网络数据集，经过挖掘和整理，得到了五个大规模附属网络，并通过投影运算，得到了五个相应的广义社会网络：CiteULike“用户-资源”网络、“用户-标签”网络、“用户-事件”网络GC，Delicious“用户-资源”网络和DBLP“作者-论文”网络。

用我们的模型在不同参数下产生的图和现实世界数据集导出的图在各种属性上相吻合，如致密幂律、缩小的直径、恒定的权值幂律分布、恒定的结点强度幂律分布和权值增量模式等等。由此说明了我们模型的可靠性，从而说明EWLS的有效性。具体实验数据和定理证明请参考文献 [77]。

3.9 本章总结

本章首先介绍了MVC问题和本文算法的相关符号及定义；然后介绍了扩展部分顶点覆盖的MVC局部搜索算法框架和一个新的边加权技术，并基于这两个技术设计了一个MVC局部算法，即EWLS。这里对这两个技术做一个简单的回顾：

- 扩展部分顶点覆盖：以前的算法都是直接以寻找顶点覆盖为目标的，EWLS则是首先寻找一个具备最优解的良好下界的部分顶点覆盖，然后再去覆盖剩余的边，从而得到一个完整的顶点覆盖。
- 新的边加权技术：虽然COVER算法也使用了边加权技术，但是COVER所使用的边加权是在每一个搜索步都更新权值。EWLS采用的边加权只在遇到局部最优解的时候才更新权值。

实验结果说明EWLS算法在DIMACS和BHOSLIB这两组标准实例集都取得了很好的结果，总的来说其性能优于以前的算法。特别值得一提的是，EWLS算法刷新了一个20年挑战实例frb100-40的求解纪录，并且至今仍是该问题的求解记录的保持者。我们还做了充分的实验对EWLS算法中的各个因素进行分析，包括控制部分顶点覆盖大小的$delta$参数，边加权策略，扫描未覆盖边时从老到新的策略等等，从而对算法有了进一步的理解。

在算法的应用方面，我们提出的EWLS算法已经被成功应用于设计基于“团叠加”原理的社会网络生成器。该生成器成功地模拟了现实世界的一些社

会网络，应用规模可达10万顶点以上，这也说明了EWLS算法的有效性。

第四章　格局检测及EWCC算法

本章提出一个新的局部搜索策略，即格局检测(Configuration Checking，简记CC)，用于处理局部搜索中的循环问题。我们使用这个策略改进EWLS算法，得到一个新的算法，叫做边加权格局检测(Edge Weighting Configuration Checking，简记EWCC)算法 [36]。EWCC算法不带任何依赖于实例的参数，并且几乎在所有DIMACS和BHOSLIB基准实例上都大幅度优胜于EWLS算法和其他启发式算法。

4.1 格局检测

局部搜索算法基本都带有贪心搜索，由此很容易出现搜索中的循环现象，也即重复访问某些（贪心的）候选解 [57]。本节提出的格局检测策略是为了减轻局部搜索中的循环问题的，以此提高局部搜索算法的性能。

很多成功的MVC，MC和MIS局部算法都采用了贪心策略，引导搜索去访问一些包含评估函数值较好的点的那些候选解。事实上，对于任何比较成功的局部搜索算法，其贪心搜索都是发挥了重要作用的。一个不带任何贪心搜索的局部搜索算法是不可能在短时间内求解大型难解实例的。这些贪心策略很直观，也确实有起到作用。但是，这样的贪心启发式很容易引发循环搜索现象，也即重复访问某些候选解。这是因为，评估值好的点在被移除出当前候选解不久之后又会被重新选进来。

已经有学者提出了一些朴素的方法来克服局部搜索算法的循环问题，比如随机游动和非改进转移。其中随机游动方法主要是在每一个搜索步以一定的概率从邻居候选解中随机挑选一个作为下一个要访问的候选解。而非改进转移则是在一定条件(比如算法在较长时间内没有找到更好的候选解)下或以一定的概率，允许算法访问评估函数值比当前候选解更差的候选解。非改进转移的一个典型应用就是模拟退火算法。另外一个著名的处理局部搜索循环问题的方法，就是Glover提出的tabu方法 [78, 79]。该方法已经被广泛应用于各种局部搜索算法 [23, 39, 80]。为了避免局部搜索马上返回刚刚访问过的候选解，tabu方法禁止对最近的变化进行反转。tabu方法有一个参数，一般叫做禁忌长度(tabu tenure)，是用来控制多少步之内的变化被禁止反转。该参数越大表明禁忌力度越大。

虽然已经有一些处理局部搜索循环问题的方法，但是就我们所知，所有的MVC局部搜索算法，在选择点加入当前候选解的时候，都只考虑点的信息，比如顶点度数 [30,31]，顶点惩罚值 [4,40]，还要分数 [23,34]，等等。这里我们提出一个新的处理循环问题的方法，据我们所知，该方法第一次在选择点对时候考虑了点的环境信息。正如我们在第2章提到过的，想记住局部搜索算法访问过的所有候选解是不可行的，因为这需要指数空间。一个折中的办法就是记住每个点的环境信息，然后避免一个点落入它刚刚离开的环境。对这个办法的直观解释就是，通过减少局部的结构循环，我们可以减少整个候选解上的循环。当然，格局检测的原理目前还没有得以完全证实，还有待探索。

基于以上考虑，我们提出一个新的处理循环问题的策略，叫做格局检测(Configuration Checking，简记CC)。此策略在向当前候选解加入点的时候会考虑点的环境。要理解CC策略，首先要明白一个关键概念—格局，也就是我们所说的一个点的环境。针对不同的图可以对格局有不同的定义，比如可以有不加权的定义，有点加权的定义和边加权的定义。下面我们给出不加权的格局和边加权的格局的形式化定义。类似地，我们也可以定义点加权的格局。

定义 4.1: 给定一个图$G = (V, E)$且C是当前候选解, 一个点v的状态为$s_v \in \{1, 0\}$, 其中$s_v = 1$表示$v \in C$，而$s_v = 0$表示$v \notin C$.

定义 4.2: **(不带权的格局)** 给定一个图$G = (V, E)$且C是当前候选解, 一个点v的格局就是表示它所有邻居点的当前状态的向量S_v。

定义 4.3: **(边加权的格局)** 给定一个图$G = (V, E)$且C是当前候选解, 一个点v的格局就是表示它所有邻居点的当前状态的向量S_v以及表示它的所有关联边的当前权值的向量W_v。

定义了格局之后，我们就可以描述格局检测策略了。通常情况下，局部搜索算法都会维护一个当前候选解C，当选择一个点来加入C的时候，对于一个点$v \notin C$，如果v的格局自从该点离开C之后就没有改变过，那么v就不能被加入C，如图4.1。在避免循环的层次上，这个策略是合理而直观的，因为这样做避免了算法再次面临之前的场景。

CC策略和之前那些只考虑点的信息而忽视点的环境的启发式不同，它在选点的时候考虑了点的环境信息。我们觉得把这个关心环境的策略加入到基于

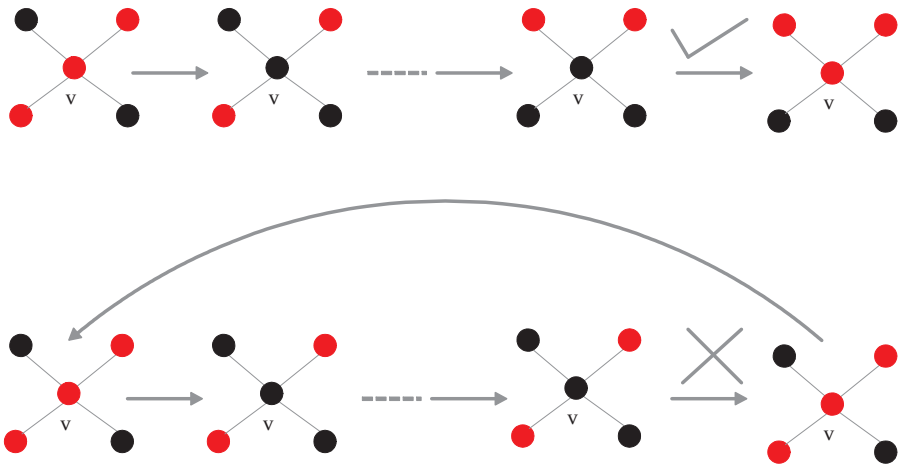

图 4.1　格局检测（红点(浅色点)表示在当前候选解）

点的启发式中是有帮助的，因为对一个点的决定，不仅仅取决于该点的评估信息，也取决于该点的环境。

CC策略的可操作性也很强，其实现相当简单，复杂度也很低，不会给算法带来额外的大开销。为了实现格局检测这个策略，我们引入一个数组$confChange$。$confChange$的每个元素是一个标示器，— $confChange[v] = 1$表示v的格局从上次v离开C之后发生改变了；而$confChange[v] = 0$则相反。在搜索过程中，只有$confChange[v] = 1$的点被允许加入当前候选解。所以，现在我们只要维护好$confChange$这个数组就可以了。$confChange$数组初始化时，所有元素都设为1，也就是所有点一开始都应该允许被加入当前候选解。然后在搜索过程中，按照以下规则进行更新：

- Rule 1: 当把v移出C时, $confChange[v]$设为0。
- Rule 2: 当u改变它的状态时, 对每一个它的邻居点v, $confChange[v]$设为1。

以上的更新规则是针对最简单的无权版本的格局定义的，对点加权和边加权版本，只要多加一个规则，在更新相关权值的时候把该点的$confChange[v]$设为1 就可以了。

这里我们要强调，格局检测其实是一种思想，而不仅仅是一个策略。我们可以根据不同的问题和算法，对格局做不同的定义，从而得到不同的格局检测策略。实际上，这也是格局检测的一大优点，它非常灵活，使用者可以根据自己的需要给出格局的定义。在格局检测思想的引导下，可以演化出各种各样的

适用于各种问题和算法的格局检测策略。在本章的最后，我们将简单地介绍格局检测思想应用于SAT问题所取得的成果。

4.2 EWCC算法描述

我们利用格局检测策略对当时最好的MVC局部搜索算法EWLS [34]进行改进。EWLS的思路是找到一个好的部分覆盖，然后将其扩展为顶点覆盖。在搜索的时候引入了边加权技术，在遇到了局部最优的时候对未覆盖的边的权值加1。EWLS在选点对进行交换的时候利用tabu方法来处理循环问题。我们把EWLS算法中用到的tabu方法替换为格局检测策略，得到新的算法EWCC。

由格局检测代替tabu方法，我们把EWLS改进为EWCC算法，如算法4所示。EWCC算法和EWLS算法只有下面两点不同：

- EWCC在选择点加入当前候选解时使用了格局检测策略而不是tabu策略。实际上，从后面的实验我们可以看到格局检测策略比tabu策略更有效。

- EWCC没有$delta$参数。也就是说，EWCC没有利用部分覆盖的概念。这个参数在EWCC中一直是1，也即每当EWCC找到一个k点的顶点覆盖，它就从当前候选解移除一个点，然后继续找(k-1)个点的顶点覆盖。

4.3 实验分析

在本节中，我们首先介绍实验设置，然后把实验分析分为四部分。第一部分详细地比较EWLS和EWCC的性能，从而说明格局检测策略的有效性。第二部分把EWCC和当前最好的局部搜索算法做比较。第三和第四部分把EWCC和当前最好的精确算法和SAT求解器做比较。

4.3.1 实验设置

在介绍实验结果之前，我们先介绍实验的相关设置。

- **实现：** EWCC由C++语言实现；其他用于比较的算法都是由作者提供源码或从网络下载，它们的源码也是由C++语言实现。所有的算法的源程序都用g++编译器带O2选项编译。

Algorithm 4: EWCC

EWCC($G, maxSteps$)
Input: graph $G = (V, E)$, $maxSteps$
Output: vertex cover of G
begin
  $step := 0$; $L := E$; $UL := E$;
  initialize all edge weights as 1 and compute $dscores$ of vertices;
  initialize $confChange[v]$ as 1 for each vertex v;
  construct C greedily until it's a vertex cover;
  $C^* := C$;
  remove a random vertex with highest $dscore$ from C;
  while $step < maxSteps$ **do**
    if $((u, v) := ChooseSwapPair2(C, L, UL)) \neq (0, 0)$ **then**
      $C := [C \backslash \{u\}] \cup \{v\}$, update the $confChange$ array;
    else
      $w(e) := w(e) + 1$ for each $e \in L$, update the $confChange$ array according to Rule 4;
      $C := [C \backslash \{u\}] \cup \{v\}$ where $u := random(C)$ and $v := random(endpoint(L))$, update the $confChange$ array;
    $tabuRemove := v$;
    if $L = \emptyset$ **then**
      $C^* := C$;
      remove a random vertex with highest $dscore$ from C;
    $step := step + 1$;
  return C^*;
end

Algorithm 5: function ChooseSwapPair2

```
1  ChooseSwapPair2(C,L,UL)
   Input: current candidate solution C, uncovered edge set L, edge set UL of
          uncovered edges unchecked in the current local search stage
   Output: a pair of vertices
2  begin
3      if S := {(u,v)|u ∈ C, v ∈ endpoint(e_oldest), u ≠ tabuRemove,
       confChange[v] = 1 and score(u,v) > 0} ≠ ∅ then
4          return random(S) ;
5      else
6          foreach e ∈ UL, from old to young do
7              if S := {(u,v)|u ∈ C, v ∈ endpoint(e), u ≠ tabuRemove,
               confChange[v] = 1 and score(u,v) > 0} ≠ ∅ then
8                  return random(S);
9      return (0,0);
10 end
```

- **实验平台：** 所有实验都在一台个人计算机上运行，其配置是3 GHz Intel Core 2 Duo CPU E8400和4GB RAM，操作系统是Linux。为了得到该机器的性能，我们在该机器上运行了DIMACS机器基准实例 [75]。这台机器求解r300.5实例需要0.19秒，求解r400.5实例需要1.12秒，求解r500.5实例需要4.24秒。
- **停止标准：**本章的实验是基于给定时间限制的停止标准，我们会在实验报告中给出每个实例的时间限制。
- **实验报告方法：** 对于每个实例，每个算法都执行100次不同随机种子的独立运行。对每个实例，我们报告以下信息：最小（或已知最小）顶点覆盖的大小(k^*)；时间限制(“CT”)；每个算法成功找到k^*解的次数(“suc”)；每个算法的平均成功时间(“time”)，表格中的时间以秒为单位。每个实例上最优算法的结果的字体加黑表示。在详细比较EWLS和EWCC的实验中，我们还报告了平均运行步数以及EWLS算法对每个实例的$delta$参数的值(d)。

4.3.2 比较EWLS和EWCC

为了说明格局检测的有效性，我们在DIMACS和BHOSLIB这两组基准实例

上比较EWCC算法与EWLS算法的性能。

在DIMACS实例上比较EWLS和EWCC

表格4.1显示了EWLS和EWCC算法在DIMACS实例上的性能。大部分实例都很容易而可以被这两个算法在1秒之内以100%成功率求解。在这37个实例中，两个算法都对其中34个实例找到了最优解(已知的最优解)。而在失败的3个实例中有2个是$brock$实例。而且，在34个成功实例中，EWLS和EWCC分别对32和31个达到了100%成功概率。总的来说，两个算法对DIMACS上除了$brock$实例都取得了很好的性能。

由表4.1可以看到，EWCC在大部分的DIMACS实例上都比EWLS表现的更好，包括一些困难实例，比如$brock$实例, $C2000.9$和$keller6$。不过, EWCC在两个$MANN$上反而比EWLS表现得更弱。

既然EWCC和EWLS在算法上唯一不同的就是EWCC在选择加入当前候选解的点时采用了格局检测策略而不是tabu策略，我们就可以说EWCC相比于EWLS的性能提高主要归功于格局检测策略。根据我们前面的直观解释，格局检测策略可以减少循环步数，我们的实验数据也支持这一观点。从表4.1可以看到，EWCC在大部分DIMACS实例上的平均运行步数都少于EWLS。

为了得到两个算法的详细性能，对于那些两个算法都不能有100%成功概率的实例，我们在表格4.2报告更详细的实验结果。其中size一栏显示找到的解的规模的最小值(平均值，最大值)；suc表示找到最小解的次数；平均时间和步数都是对所有找到最小解的运行上的数据求平均的。

在BHOSLIB实例上比较EWLS和EWCC

表4.3显示了EWLS和EWCC算法在BHOSLIB问题实例上的性能。这两个算法都在所有的BHOSLIB实例上找到了最优解，并且他们找到的解最差就比最优解多一个点。根据在BHOSLIB实例上的工作 [35]，包括这些实例的SAT，MAXSAT和CSP版本，EWLS和EWCC以100%的成功概率解出了最多的问题实例(40个实例中的29个)。

从表4.3可以看到，EWCC在BHOSLIB问题实例上明显比EWLS好很多。对于那些两个算法都达到100%成功率的实例，EWCC总是更快地找到最优解；对于其他实例，EWCC都有更高的成功率，除了$frb59\text{-}26\text{-}2$例外。另外，我们应该注意，EWLS的性能是在优化的$delta$参数下获得的，而EWCC是不带参数的算法。

Graph			EWLS				EWCC		
Instance	k^*	CT(s)	d	suc	time	steps	suc	time	steps
brock200_2	188*	1000	1	100	0.264	119476	100	**0.244**	**117800**
brock200_4	183*	1000	1	100	3.184	1696748	100	**2.080**	**1233612**
brock400_2	371*	1000	1	4	338.641	122253257	**12**	374.520	185029672
brock400_4	367*	1000	1	100	74.146	27107873	100	**25.376**	**10305332**
brock800_2	776*	1000	1	0	n/a	n/a	0	n/a	n/a
brock800_4	774*	1000	1	0	n/a	n/a	0	n/a	n/a
C125.9	91*	1000	1	100	< 0.001	**70**	100	< 0.001	73
C250.9	206*	1000	1	100	< 0.001	**1541**	100	< 0.001	1743
C500.9	443	1000	1	100	0.236	115481	100	**0.199**	**99203**
C1000.9	932	1000	1	100	3.326	896259	100	**2.657**	**758546**
C2000.5	1984	1000	1	100	2.471	**73177**	100	**2.429**	76000
C2000.9	1921	2400	1	21	557.745	66666972	**34**	858.733	107898557
C4000.5	3982	2400	1	100	**686.472**	**7257183**	100	738.909	8802400
DSJC500.5	487*	1000	1	100	0.018	3179	100	**0.014**	**2344**
DSJC1000.5	985*	1000	1	100	0.798	69431	100	**0.592**	**58466**
gen200_p0.9_44	156*	1000	1	100	< 0.001	2434	100	< 0.001	**1296**
gen200_p0.9_55	145*	1000	1	100	< 0.001	299	100	< 0.001	**242**
gen400_p0.9_55	345*	1000	1	100	0.074	41906	100	**0.049**	**29450**
gen400_p0.9_65	335*	1000	1	100	< 0.001	1724	100	< 0.001	**1586**
gen400_p0.9_75	325*	1000	1	100	< 0.001	1074	100	< 0.001	**920**
hamming8-4	240*	1000	1	100	< 0.001	1	100	< 0.001	1
hamming10-4	984*	1000	1	100	0.030	5010	100	**0.019**	**2913**
keller4	160*	1000	1	100	< 0.001	74	100	< 0.001	**70**
keller5	749*	1000	1	100	0.025	5314	100	**0.019**	**4373**
keller6	3302	1000	1	100	4.934	226592	100	**3.757**	**185507**
MANN_a27	252*	1000	1	100	0.002	**7195**	100	0.002	8105
MANN_a45	690*	2400	1	**100**	**194.969**	**124004605**	94	698.505	404529167
MANN_a81	2221	2400	3	**22**	**621.683**	207424642	1	634.460	145174870
p_hat300-1	292*	1000	1	100	0.002	215	100	0.002	**148**
p_hat300-2	275*	1000	1	100	0.001	80	100	0.001	**75**
p_hat300-3	264*	1000	1	100	< 0.001	794	100	< 0.001	**679**
p_hat700-1	689*	1000	1	100	0.035	2363	100	**0.028**	**1794**
p_hat700-2	656*	1000	1	100	0.009	222	100	0.009	**212**
p_hat700-3	638*	1000	1	100	0.007	970	100	**0.006**	**884**
p_hat1500-1	1488*	1000	1	100	13.587	419374	100	**9.784**	**334296**
p_hat1500-2	1435*	1000	1	100	0.091	906	100	**0.081**	**708**
p_hat1500-3	1406	1000	1	100	0.053	3309	100	**0.042**	**2779**

表 4.1　EWLS和EWCC在DIMACS实例的比较，k^*列标注星号的表示k^*已经被证明是最优解的规模

Graph	EWLS				EWCC			
Instance	size	suc	time	steps	size	suc	time	steps
brock400_2	371(374.84,375)	4	338.641	122253257	371(374.52,375)	12	374.520	185029672
brock800_2	779	100	0.366	59157	779	100	0.489	75309
brock800_4	779	100	0.617	98963	779	100	0.619	103494
C2000.9	1921(1921.84,1923)	21	557.745	66666972	1921(1921.66,1922)	34	858.733	107898557
MANN_a45	690	100	194.969	124004605	690(690.06,691)	94	698.505	404529167
MANN_a81	2221(2221.78,2222)	22	621.683	207424642	2221(2222.89,2223)	1	634.460	145174870

表 4.2　EWLS和EWCC在表现较差的DIMACS实例上的性能

如同表4.1中DIMACS实例的实验结果一样，EWCC在BHOSLIB实例上又比EWLS显示出显著的步数性能的提高。从frb35-17-2开始，除frb56-25-3之外，对所有两个算法都达到100%成功概率的实例，EWCC的平均步数都比EWLS少。这再一次支持了我们对格局检测能减少局部搜索中的循环步数的猜想。有人可能会怀疑，EWCC的好性能会不会是来自EWCC比较优化的实现，而不是格局检测策略。实际上，EWCC是在EWLS的代码的基础上实现的，修改的代码也只有几行而已。

4.3.3 与其他启发式算法比较

我们把EWLS，EWCC和目前最好的其他两个求解MVC(MC,MIS)问题的启发式算法做比较。他们的资料如下所示：

(1) **PLS** [40]：最大团启发式算法。它是局部搜索算法DLS-MC [4]的改进版。文献 [4]中的实验结果显示，DLS-MC算法在大部分DIMACS实例上优胜于之前最好的最大团启发式算法。而文献 [40]中提出的PLS算法不但消除了DLS-MC算法的参数，而且在全部DIMACS实例上比DLS-MC算法具备可比的甚至更好的性能。所以，PLS可以被认为是本章工作发表时最好的最大团启发式算法。

(2) **COVER** [23]：最小顶点覆盖启发式算法。相关论文 [23]在德国人工智能2007年会议上发表，并成为当时3篇竞争最佳论文的其中之一[1]。文献 [23]的实验结果表明，COVER在DIMACS实例上和DLS-MC算法有着可竞争的性能，而对于BHOSLIB实例则是当时性能最佳的算法。特别值得

[1]http://www.ki2007.uos.de/

Graph			EWLS				EWCC		
Instance	k^*	CT(s)	d	suc	time	steps	suc	time	steps
frb30-15-1	420	1000	2	100	0.069	**29095**	100	**0.062**	29854
frb30-15-2	420	1000	1	100	0.096	43222	100	**0.090**	**42847**
frb30-15-3	420	1000	2	100	0.465	199161	100	**0.329**	**160758**
frb30-15-4	420	1000	2	100	0.071	**30075**	100	**0.069**	33223
frb30-15-5	420	1000	2	100	0.246	106133	100	**0.169**	**82382**
frb35-17-1	560	1000	2	100	1.178	**419575**	100	**1.061**	436347
frb35-17-2	560	1000	2	100	1.316	460821	100	**0.899**	**363737**
frb35-17-3	560	1000	2	100	0.305	112705	100	**0.209**	**88393**
frb35-17-4	560	1000	2	100	1.326	476318	100	**1.057**	**441708**
frb35-17-5	560	1000	2	100	0.711	251727	100	**0.580**	**236252**
frb40-19-1	720	1000	2	100	0.722	226711	100	**0.552**	**199420**
frb40-19-2	720	1000	2	100	17.680	5216744	100	**11.295**	**3935909**
frb40-19-3	720	1000	3	100	3.626	1014101	100	**2.965**	**1012203**
frb40-19-4	720	1000	1	100	18.751	5735325	100	**13.787**	**4592540**
frb40-19-5	720	1000	2	100	87.495	25283646	100	**41.714**	**14231964**
frb45-21-1	900	1000	4	100	14.594	3224649	100	**9.066**	**2583572**
frb45-21-2	900	1000	2	100	26.637	6507751	100	**14.927**	**4245886**
frb45-21-3	900	1000	2	100	96.047	23568777	100	**56.404**	**16118564**
frb45-21-4	900	1000	2	100	18.628	46633662	100	**14.671**	**4290939**
frb45-21-5	900	1000	3	100	84.679	19147891	100	**41.861**	**11865589**
frb50-23-1	1100	1800	2	100	262.313	56455242	100	**123.803**	**29773443**
frb50-23-2	1100	1800	2	59	509.296	106684368	**80**	631.688	154734494
frb50-23-3	1100	1800	1	42	692.210	155456585	**54**	797.524	189561481
frb50-23-4	1100	1800	1	100	31.573	7299068	100	**23.847**	**5792613**
frb50-23-5	1100	1800	2	100	117.730	25697612	100	**84.832**	**20774461**
frb53-24-1	1219	2000	4	25	745.425	129528395	**30**	985.294	217902696
frb53-24-2	1219	2000	3	58	603.768	109004768	**81**	772.619	170480847
frb53-24-3	1219	2000	2	100	165.720	32678443	100	**116.726**	**26143480**
frb53-24-4	1219	2000	2	50	772.885	147937643	**81**	**642.068**	**140122496**
frb53-24-5	1219	2000	2	100	250.262	48559793	100	**125.366**	**27379752**
frb56-25-1	1344	2200	3	28	742.227	124088139	**62**	826.916	165580423
frb56-25-2	1344	2200	2	32	730.987	134692382	**54**	868.426	178346136
frb56-25-3	1344	2200	4	100	307.964	**51838431**	100	**285.046**	57511723
frb56-25-4	1344	2200	4	100	234.376	39241213	100	**183.438**	**37005137**
frb56-25-5	1344	2200	3	100	94.242	16390263	100	**79.865**	**15955311**
frb59-26-1	1475	2400	2	21	1015.145	171236143	**22**	**1012.230**	186973450
frb59-26-2	1475	2400	3	**18**	**1116.251**	**179661855**	8	1139.360	209757895
frb59-26-3	1475	2400	3	42	664.507	103219920	**65**	913.653	168513191
frb59-26-4	1475	2400	4	18	841.930	125795679	**26**	1080.650	197407398
frb59-26-5	1475	2400	1	100	334.044	60010102	100	**173.675**	**32789882**

表 4.3　EWLS和EWCC在BHOSLIB 实例的比较

Graph			EWLS		EWCC		PLS		COVER	
Instance	k^*	CT(s)	suc	time	suc	time	suc	time	suc	time
brock400_2	371	1000	4	338.641	12	347.520	**100**	**0.145**	3	228.520
brock400_4	367	1000	100	74.146	100	25.376	**100**	**0.029**	47	702.573
brock800_2	776	1000	0	n/a	0	n/a	**100**	**3.886**	0	n/a
brock800_4	774	1000	0	n/a	0	n/a	**100**	**1.314**	0	n/a
C2000.9	1921	2400	21	557.745	**34**	**858.733**	34	1014.611	7	1337.397
C4000.5	3982	2400	100	686.472	100	738.909	**100**	**66.702**	100	658.33
gen400_p0.9_55	345	1000	100	0.074	**100**	**0.049**	100	15.171	100	0.353
keller6	3302	1000	100	4.934	**100**	**3.757**	90	342.972	100	68.214
MANN_a45	690	2400	**100**	**194.969**	94	698.505	1	985.910	97	643.638
MANN_a81	2221	2400	**22**	**621.683**	1	634.460	0	n/a	2	1875.320
p_hat1500-1	1488	1000	100	13.587	100	9.784	**100**	**2.355**	100	18.095

表 4.4　DIMACS实例比较结果

一提的是，这个算法发表的时候刷新了三个大规模的难解实例的最优解纪录（$MANN\_a45$，$MANN_a81$和挑战实例$frb100\text{-}40$）。

总的来说，至本章为止，PLS和COVER分别是除了EWLS和EWCC之外最好的MC启发式算法和最好的MVC启发式算法。而我们于文献中找到的最好的MIS启发式算法 [27]则除了两个$MANN$实例，在DIMACS实例上表现弱于DLS-MC [4,27]。因此PLS和COVER是本比较实验的最佳候选算法。

在DIMACS实例的比较结果

DIMACS实例很多对于这四个算法而言都太容易了，可以在2秒之内以100%成功率找到最优解，因此没有报告那些实例的结果。表4.4显示了4个算法在DIMACS困难实例上的性能比较。由表4.4可见，PLS和EWLS，EWCC在DIMACS上平分秋色。PLS在brock图和C4000.5以及p_hat1500-1上明显占优。EWLS在两个MANN实例上明显占优；而EWCC在其他4个困难实例上明显占优。COVER则在这些实例上没有任何优势。

在BHOSLIB实例的比较结果

表4.5显示了四个算法在BHOSLIB实例上的性能比较。为了关注于算法的性能差距，我们不报告两组小实例($frb30$和$frb35$)的实验结果，因为它们对四个算法来说都太容易了。表4.5阐述了在BHOSLIB实例上，EWCC占绝对优势。其次是EWLS，跟着是COVER，而PLS在BHOSLIB上表现很差。特别的，

Graph			EWLS		EWCC		PLS		COVER	
Instance	k^*	CT(s)	suc	time	suc	time	suc	time	suc	time
frb40-19-1	720	1000	100	0.722	**100**	**0.552**	100	10.421	100	1.580
frb40-19-2	720	1000	100	17.680	**100**	**11.295**	100	85.254	100	17.180
frb40-19-3	720	1000	100	3.626	**100**	**2.965**	100	9.058	100	5.059
frb40-19-4	720	1000	100	18.751	100	13.787	100	77.393	**100**	**11.794**
frb40-19-5	720	1000	100	87.495	**100**	**41.714**	86	373.058	100	124.153
frb45-21-1	900	1000	100	14.594	**100**	**9.066**	100	52.175	100	14.427
frb45-21-2	900	1000	100	26.637	**100**	**14.927**	100	170.308	100	37.692
frb45-21-3	900	1000	100	96.047	**100**	**56.404**	8	442.780	100	109.831
frb45-21-4	900	1000	100	18.628	**100**	**14.671**	100	111.375	100	20.888
frb45-21-5	900	1000	100	84.679	**100**	**41.861**	97	248.291	100	105.138
frb50-23-1	1100	1800	100	262.313	**100**	**123.803**	30	859.997	100	267.752
frb50-23-2	1100	1800	59	509.296	**80**	631.688	3	547.666	48	594.427
frb50-23-3	1100	1800	42	692.210	**54**	797.524	2	1429.810	37	585.273
frb50-23-4	1100	1800	100	31.573	**100**	**23.847**	100	92.946	100	32.734
frb50-23-5	1100	1800	100	117.730	**100**	**84.832**	77	680.537	100	167.929
frb53-24-1	1219	2000	25	745.425	**30**	985.294	1	231.270	17	796.249
frb53-24-2	1219	2000	58	603.768	**81**	772.619	6	1316.510	50	558.181
frb53-24-3	1219	2000	100	165.720	**100**	**116.726**	20	856.849	99	256.704
frb53-24-4	1219	2000	50	772.885	**81**	**642.068**	21	962.502	48	810.777
frb53-24-5	1219	2000	100	250.262	**100**	**125.366**	10	1549.25	95	341.714
frb56-25-1	1344	2200	28	742.227	**62**	826.916	1	1259.190	26	839.331
frb56-25-2	1344	2200	32	730.987	**54**	868.426	0	n/a	17	815.253
frb56-25-3	1344	2200	100	307.964	**100**	**285.046**	0	n/a	97	492.755
frb56-25-4	1344	2200	100	234.376	**100**	**183.438**	12	1225.533	93	362.35
frb56-25-5	1344	2200	100	94.242	**100**	**79.865**	30	959.249	100	168.166
frb59-26-1	1475	2400	21	1015.145	**22**	**1012.230**	0	n/a	17	883.755
frb59-26-2	1475	2400	**18**	**1116.251**	8	1139.360	0	n/a	9	678.554
frb59-26-3	1475	2400	42	664.507	**65**	913.653	6	1537.731	26	1149.02
frb59-26-4	1475	2400	18	841.930	**26**	1080.650	1	2224.220	4	1596.32
frb59-26-5	1475	2400	100	334.044	**100**	**173.675**	37	1202.803	98	430.602

表 4.5　BHOSLIB实例比较结果

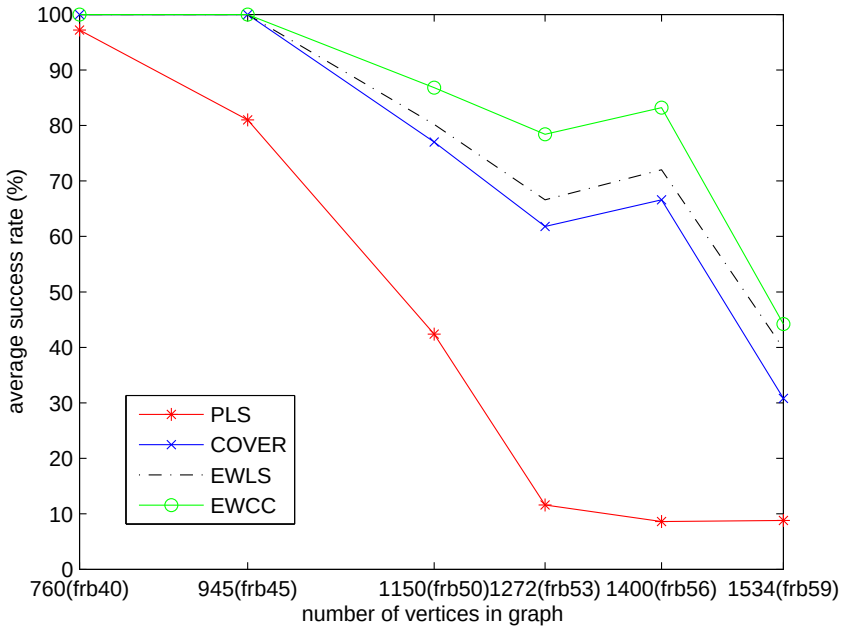

图 4.2　EWLS和EWCC与其他算法在BHOSLIB实例上的平均成功率比较

无论是成功率还是平均时间，EWCC对除了frb40-19-4和frb59-26-2之外的所有实例，都是性能最好的算法。我们还总结了四个算法的在每一组实例上的平均成功率，见图4.2。这些结果毫无疑问地表明，在本工作发表之时，EWCC给出了BHOSLIB实例上最好的求解结果。这个结论即使是和这些BHOSLIB实例的SAT，MAXSAT和CSP问题版本的算法相比也是成立的 [35]。

另外值得一提的是，对于BHOSLIB的挑战实例frb00-40，由2007年COVER给出了一个3903点的顶点覆盖，然后是2010年EWLS刷新了求解纪录，找到了3902点的顶点覆盖，而EWCC也找到了3902点的顶点覆盖。具体地，我们在这个实例上对EWCC进行100次独立运行，给定时间限制为4800秒。其中1次运行找到了一个3902个点的顶点覆盖，时间为2856.43秒；82次找到了3903个点的顶点覆盖，平均时间为2145.09秒。

4.3.4 与精确算法比较

某种程度上，精确算法和局部搜索算法在应用上是互补的。精确算法经常比局部搜索算法在结构化实例上表现更好，而局部搜索算法则在随机实例上表现更好。比如，这个现象在SAT问题的求解中就很典型，从每次的SAT算法

Graph		EWCC		MaxCLQdyn+EFL	Graph		EWCC		MaxCLQdyn+EFL
Instance	k^*	suc	time	+SCR time	Instance	k^*	suc	time	+SCR time
brock400_1	373	0	n/a	**286.578**	p_hat300-3	264	100	**0.001**	1.307
brock400_2	371	12	374.520	**125.062**	p_hat500-3	450	100	**0.007**	50.409
brock400_3	369	100	**45.054**	251.442	p_hat700-2	656	100	**0.009**	3.272
brock400_4	367	100	**25.376**	119.243	p_hat700-3	638	100	**0.006**	1141.920
brock800_1	777	0	n/a	**5861.124**	p_hat1000-2	954	100	**0.022**	108.944
brock800_2	776	0	n/a	**5138.098**	p_hat1000-3	932	100	**0.029**	113860.404
brock800_3	775	0	n/a	**3298.392**	p_hat1500-1	1488	100	9.784	**3.096**
brock800_4	774	0	n/a	**2391.444**	p_hat1500-2	1435	100	**0.081**	866.514
keller5	749	100	**0.019**	6884.461	sanr200_0.9	158	100	**0.001**	5.20
MANN_a45	690	94	698.505	**21.169**	sanr400_0.7	379	100	**0.012**	97.722

表 4.6　EWCC与精确算法MaxCLQdyn+EFL+SCR在DIMACS实例上的比较

比赛结果也可以看出来 [81]。因此，一个自然的问题就是，这个现象在MVC (MIS，MC)求解中是否也会发生。

对于MVC，MC和MIS这三个问题，大部分精确算法都是针对最大团问题的 [13–19]。最近提出的一个叫MaxCLQ的最大团分支限界算法，使用了MaxSAT推理技术来求上界从而获得了很大的进步 [18]。文献 [18]中在DIAMCS实例上的实验说明MaxCLQ明显地优于之前其他最大团精确算法。后来，MaxCLQ的作者又利用扩展失败文字检查和软子句放松这两个策略提高了MaxCLQ算法，得到的更好的算法叫MaxCLQdyn+EFL+SCR [19]。因为MaxCLQdyn+EFL+SCR的巨大成功，在本工作中我们的算法只与MaxCLQdyn+EFL+SCR做了比较。实验结果如表4.6 所示。

MaxCLQdyn+EFL+SCR的运行时间是文献 [19]中的时间归一化到本工作的机器的。在文献 [19]中实验的机器是一个3.33 GHz Intel Core 2 Duo CPU 4 Gb memory，操作系统是linux。该机器在运行DIMACS机器基准实例时，对r300.5需要0.172秒，对r400.5需要1.016秒，对r500.5需要3.872秒。而我们的机器相应的时间为0.19，1.12, 4.24秒。所以，我们把文献 [19]中MaxCLQdyn+EFL+SCR的运行时间乘以(=(4.242/3.872+1.118/1.016)/2=1.098 来得到归一到本机器的时间。这种归一化方法由“Second DIMACS Implementation Challenge for Cliques, Coloring, and Satisfiability”提出并被广泛用于比较不同平台算法性能 [4, 18, 19, 40]。

Instance	SAT 2004 results	EWCC suc	EWCC time	Instance	SAT 2004 results	EWCC suc	EWCC time
frb40-19-1	Solved by 28 solvers	100	0.552	frb53-24-1	Unsolved	6	152.252
frb40-19-2	Solved by 27 solvers	100	11.295	frb53-24-2	Unsolved	21	131.210
frb45-21-1	Solved by 8 solvers	100	9.066	frb56-25-1	Unsolved	19	135.085
frb45-21-2	Solved by 5 solvers	100	14.927	frb56-25-2	Unsolved	18	113.936
frb50-23-1	Solved by 1 solver	91	92.471	frb59-26-1	Unsolved	4	112.193
frb50-23-2	Solved by 1 solver	28	137.158	frb59-26-2	Unsolved	1	56.150

表 4.7 EWCC与2004年SAT比赛结果比较

结果显示EWCC在随机实例比如p_hat和$sanr$实例上的性能比MaxCLQdyn+EFL+SCR好得多。文献 [19]中并没有在BHOSLIB实例上评估MaxCLQdyn+EFL+SCR的性能，因为这些实例对该算法而言太难了。我们相信EWCC在困难的随机实例组BHOSLIBshang也能优胜于MaxCLQdyn+EFL+SCR。

对于困难的结构化实例，文献 [19]主要评估了MaxCLQdyn+EFL+SCR在$brock$实例上的性能，而对难度最大的5个开放实例($MANN_a$81, $hamming$10-4, $johnson$32-2-4, $keller$6和p_hat1500-3)则没有评估。虽然总的来说MaxCLQdyn+EFL+SCR 在这些结构化实例上比EWCC表现更好，不过EWCC也在一些结构化实例比如两个$brock$实例和$keller$5 胜于MaxCLQdyn+EFL+SCR。另外，一些基于点惩罚的局部搜索算法如DLS-MC [4]和PLS [40]在$brock$明显比MaxCLQdyn+EFL+SCR表现好很多。

基于以上比较结果，我们可以说，对于MVC (MIS, MC)问题，局部搜索算法在随机实例上显著占优。更有趣的是，局部搜索算法在结构化实例上也能和精确算法抗衡。

4.3.5 与SAT算法比较

BHOSLIB实例中一些对应的SAT实例也被广泛用于SAT比赛。在BHOSLIB主页 [35]上显示了2004年SAT比赛上这些BHOSLIB实例的比赛结果。我们把EWCC在这些实例上的结果和2004年SAT比赛结果做一下比较。

2004年SAT比赛分为两个阶段。在第一阶段，所有的55个求解器都对每个实例运行一次，时间限制为600秒。比赛的机器是Athlon 1800+ CPU和Intel

Xeon 2.4 GHz CPU 1 GB 内存 [82]。为了获得有意义的比较，我们把EWCC对每个实例的运行时间限制为300秒(是第一阶段时间限制的一半)。读者可能会考虑机器的区别和速度的归一不精确会失去实验结果的有效性，不过一些明显的差距还是能说明问题的。表4.7显示出EWCC在BHOSLIB实例上的性能相对于2004年SAT比赛的结果是一个显著的提高。2011年这些BHOSLIB SAT实例又被用于SAT算法比赛，从比赛结果可以看出，SAT算法对这些实例的求解仍然比EWCC明显要差 [37]。

4.4 讨论

本节首先通过实验分析EWLS和EWCC算法的运行时间分布和运行长度分布，来理解两个算法的运行时行为。然后通过实验证据支持格局检测减少循环的猜想，并解释了格局检测所采取的近似实现及其单步时间复杂度。最后简单介绍我们把格局检测策略应用于SAT局部搜索算法的工作。

4.4.1 运行时分析

一个随机算法随着运行时间的增加其行为如何变化是该算法一个重要方面，也可以由此看出算法在实际运行中大概会出现什么样的行为。对这方面的刻画主要是基于Hoos和Stützle提出的运行时间分布(run-time distributions，简记RTDs)和运行长度分布(run-length distributions，简记RLDs) [47, 83]。这个方法已经被广泛应用于分析随机局部搜索算法 [4, 65–67]。

我们对EWLS和EWCC在一些典型实例上分析了算法随着运行时间和运行步数的增加，其求解成功率的变化情况。对于DIMACS实例，我们挑选了C1000.9和p_hat1500-1，这两个实例都具备一定规模和难度。而且它们的结构还不一样，C1000.9是一个稠密图，而p_hat1500-1的度分布则很广。对于BHOSLIB实例，我们挑选了frb53-24-5和frb59-26-5，这是适合用于学习EWLS和EWCC算法运行时行为的实例，因为它们不是很容易以至于很快被解出来，也不是很难以至于算法没有好的成功率。两个算法对挑选的这些实例都执行100次独立运行，并且两个算法对每个实例的每次运行都找到了最优解。所以，这些实例能全面反映算法随着时间变化其可以达到的结果如何变化。

图4.3和4.4分别显示了EWLS和EWCC两个算法在DIMACS和BHOSLIB实例

上的RTD及RLD分析结果。由图中可以看出，EWLS和EWCC两个算法都相对于运行长度和运行时间有很好的伸缩性。我们的进一步观察发现，这些RLDs和RTDs都可以用指数分布$ed[m](x) = 1 - 2^{-x/m}$很好地近似，其中m是该指数分布的中值。我们使用$Kolmogorov\text{-}Smirnov$检验测试近似的有效性，在标准显著水平为$\alpha = 0.05$的检验下不能拒绝原假设，即这些RLDs和RTDs可以看成是我们给出的近似指数分布的一些随机采样，p值在0.19和0.96之间。

这个指数近似现象也在其他的高性能随机局部搜索算法出现过，比如用于求解最大团问题 [4]，SAT问题 [65, 83]，MAXSAT问题 [66]和规划问题 [67]的随机局部搜索算法。文献 [47, 65, 83]对具有指数近似的RTD(和RLD)的随机局部搜索算法有这样的结论，他们在一定时间（步数）之内找到最优解的概率不依赖于什么时候重启。也就是说，这样的算法对时间和步数限制具有很好的伸缩性，所以有没有重启以及何时重启算法，对算法的性能不会有太大影响。因此，可以说这样的算法具有并行最优性，即算法在不同随机种子下并行可以达到最佳的并行加速比。而我们的实验说明，EWLS和EWCC就是具有这样良好属性的算法。

这些RTD和RLD分析图也表明EWCC优胜于EWLS，尤其对两个BHOSLIB实例更明显。这和表4.1 及表4.3的结果是一致的。

4.4.2 关于格局检测的更多见解

我们现在通过实验更好地理解CC策略的机制。CC策略可以被用来处理循环问题，从而提高局部搜索算法的性能。现在我们通过实验来说明，CC策略能减少循环搜索。对于DIMACS实例，除掉在1秒之内能被EWLS和EWCC成功求解的实例，其他实例都出现在本实验中。对于BHOSLIB实例，我们对5个较大的实例组，每一组挑选两个不同难度的实例，另外还有挑战实例frb100-40。

令$S = \{v | v \in V \backslash C \text{ and } confChange[v] = 0\}$，这个顶点集合的元素是那些从上一次离开当前候选解之后其格局就没有改变过的顶点。我们把EWCC在每个实例上执行10次，每次执行的随机种子不一样，对每个实例采用表4.1和表4.3里面该实例上的时间限制。对每次运行，我们统计S的大小($|S|$) 和比率$\frac{|S|}{|V \backslash C|}$，这些数据是对该次运行所有搜索步求平均的结果。我们对每个实例上的10次运行得到的$|S|$和$\frac{|S|}{|V \backslash C|}$的值求均值，在表4.8中以$\frac{\text{均值}}{\text{标准方差}}$的形式报告这些结果。

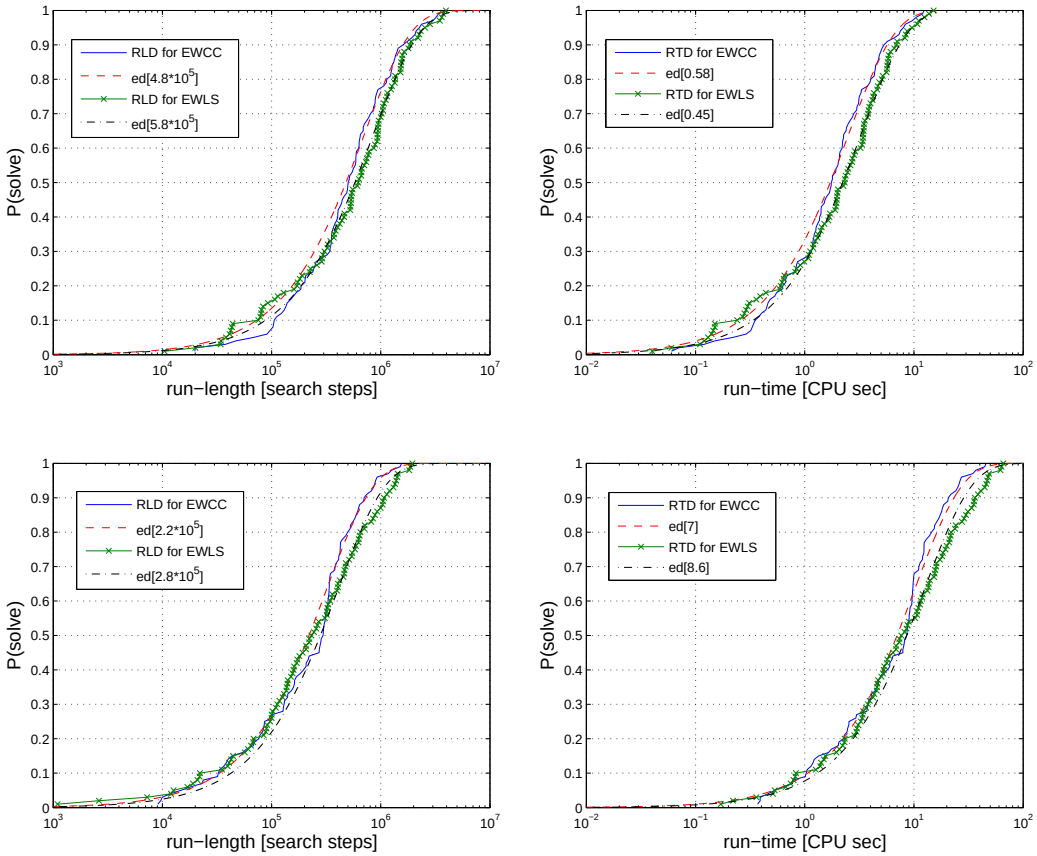

图 4.3 EWLS和EWCC在$C1000.9$(上)和$p_hat1500\text{-}1$(下)的RLD(左)和RTD(右)
其中近似指数函数为$ed[m](x) = 1 - 2^{-x/m}$

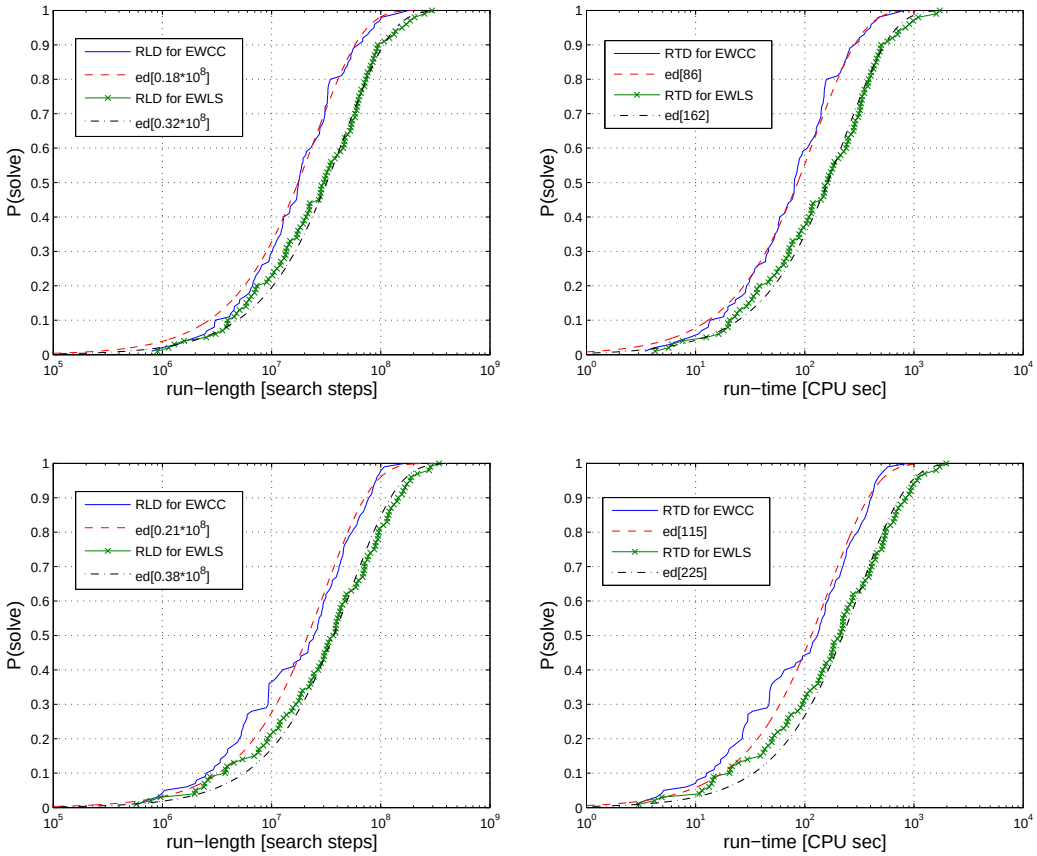

图 4.4 EWLS和EWCC在frb53-24-5(上)和frb59-26-5(下)的RLD(左)和RTD(右)
其中近似指数函数为$ed[m](x) = 1 - 2^{-x/m}$

Instance	$\|S\|$	$\frac{\|S\|}{\|V\setminus C\|}$	Instance	$\|S\|$	$\frac{\|S\|}{\|V\setminus C\|}$
brock200_4	$\frac{0.8456}{0.00708}$	$\frac{0.04975}{0.00044}$	frb45-21-1	$\frac{2.3354}{0.03511}$	$\frac{0.0523}{0.00108}$
brock400_2	$\frac{1.4587}{0.00216}$	$\frac{0.0561}{0.00008}$	frb45-21-3	$\frac{2.2947}{0.10585}$	$\frac{0.0521}{0.00057}$
brock400_4	$\frac{1.4418}{0.02680}$	$\frac{0.0555}{0.00106}$	frb50-23-1	$\frac{2.5227}{0.19606}$	$\frac{0.0517}{0.00130}$
C1000.9	$\frac{3.4168}{0.04079}$	$\frac{0.0506}{0.00053}$	frb50-23-3	$\frac{2.85892}{0.06701}$	$\frac{0.0572}{0.00132}$
C2000.5	$\frac{0.7568}{0.00503}$	$\frac{0.0474}{0.00035}$	frb53-24-1	$\frac{2.8439}{0.03577}$	$\frac{0.0536}{0.00067}$
C2000.9	$\frac{4.4805}{0.21037}$	$\frac{0.0569}{0.00257}$	frb53-24-3	$\frac{2.7152}{0.04402}$	$\frac{0.0516}{0.00087}$
C4000.5	$\frac{0.8247}{0.00413}$	$\frac{0.0489}{0.00028}$	frb56-25-1	$\frac{2.9740}{0.04743}$	$\frac{0.0531}{0.00082}$
keller6	$\frac{2.6849}{0.19274}$	$\frac{0.0473}{0.00369}$	frb56-25-3	$\frac{2.7903}{0.06534}$	$\frac{0.0500}{0.00108}$
MANN_a45	$\frac{2.8835}{0.03406}$	$\frac{0.0084}{0.00009}$	frb59-26-1	$\frac{3.0517}{0.07798}$	$\frac{0.0517}{0.00124}$
MANN_a81	$\frac{4.2277}{0.04637}$	$\frac{0.0039}{0.00004}$	frb59-26-3	$\frac{3.0913}{0.05977}$	$\frac{0.0525}{0.00089}$
p_hat1500-1	$\frac{0.5308}{0.04173}$	$\frac{0.0473}{0.00357}$	frb100-40	$\frac{4.8507}{0.08021}$	$\frac{0.0484}{0.00098}$

表 4.8　关于格局检测策略的统计

如表4.8所示，在这些实例上比率$\frac{|S|}{|V\setminus C|}$的值在基本在4.73%和5.72%之间，也就是说$V\setminus C$集合里有一小部分比例的顶点是从离开$C$以后就没有改变过格局的($confChange[v] = 0$)。这样的顶点虽然不是很多，但是通过禁止把它们加入当前候选解，我们就大大地提高了EWLS的性能。对于$MANN_a45$和$MANN_a81$，比率$\frac{|S|}{|V\setminus C|}$的值仅为0.84%和0.39%，所以CC策略很难发挥作用。这实际上也解释了为什么EWCC在这两个实例上的性能不是很好。

不过，这里我们要说明，本工作中的CC策略的实现是一个近似实现，而并没有完全地反映CC策略的机制。为了看清楚这一点，让我们考虑以下情景：有一条边(u, v)，其中$u \notin C$而$v \in C$，v被从C中移除(这时$confChange[v]$被赋值为0)，接着u被插入到C中然后又被移除(这时$confChange[v]$为1)。假设v的其他邻居点都没有改变状态，那么v的格局实际上就没有改变。可是在本工作的实现中，$confChange[v]$最后为1，也就是认为v的格局已经改变了。

对CC策略一个比较朴素的精确实现是这样的：每当一个顶点从C中移除的时候就记下它当时的格局，然后在考虑它是否可以作为加入C的候选点时检查它的格局是否已经改变了。不过这种实现方法很耗时。我们可以把EWCC中CC策略的实现替换为这种精确的实现，那样得到的新算法对一个点存储和检查格局都需要$O(\Delta(G))$时间，所以单步时间复杂度为$O(\Delta(G)|L|) + O(\Delta(G)) = O(\Delta(G)|L|)$。然而在EWCC中，CC策略的单步时间复杂度仅为$O(|L|) + 1 + O(\Delta(G)) = O(max(\Delta(G), |L|))$（检查$L$中的边的端点的$confChange$，重设被移除的点的$confChange$，更新被移除的点的邻居点的$confChange$）。为了在CC策略的精确度和低的单步复杂度之间取得平衡，我们在EWCC中对CC策略采用了近似实现。

4.5 格局检测在其他算法的应用

格局检测是一个普适性的局部搜索策略，能广泛应用于提高组合问题的局部搜索算法。除了EWCC，我们还把格局检测应用于改进COVER算法，得到的COVERcc算法在性能上比COVER算法有明显提高。另外，我们还运用格局检测策略设计了SAT局部搜索算法和带权Max2SAT局部搜索算法，在广泛的实例上都取得了世界领先的求解性能。

4.5.1 利用格局检测改进COVER算法

为了说明格局检测策略对MVC局部搜索算法的有效性是普遍成立的，我们还把格局检测用于改进另一个MVC局部搜索算法，也即COVER算法。我们把改进COVER算法得到的新算法叫做COVERcc (COVER with Configuration Checking)。

COVERcc算法和COVER算法的唯一不同就是，在一个交换步中，选择被加入的点的时候，COVERcc使用了CC策略而不是像COVER 那样使用tabu策略。我们给出了COVERcc的伪代码，如算法6所示。COVERcc算法的其他部分是和COVER算法一样的，描述如下。有兴趣对COVER算法做详细了解的读者请参考 [23]。

如同COVER一样，在COVERcc中，我们说一个点v潜在覆盖了一条边$e(u,v)$，如果该边的另一点$u \notin C$。COVERcc使用了边加权技术，每一个搜索步的最后都增加未覆盖的边权值。对于一个顶点v，它的权值$weight(v)$就

是所有v潜在覆盖的边的权值之和。一个搜索不就是交换两个点，从当前候选解C移出一个点u，然后再选一个点v加入到C中。交换一对顶点的收益定义为$gain(u,v) = weight(v) - weight(u) + \delta$，其中$\delta$就是$w(e(u,v))$，如果$e(u,v)$不存在，$\delta$就是0。

每一步，COVERcc就随机选一条未覆盖的边e，然后选两个顶点$u \in C$和$v \in endpoint(e)$，这两个顶点必须满足$confChange[v] = 1$，且u不是上一步刚加入C的点的前提下，最大化$gain(u,v)$。在选择交换顶点对时，如果出现相同$gain(u,v)$的顶点对，就选择v更老的那个顶点对来交换。交换了顶点对之后，COVERcc就对每条未覆盖的边的权值加1，并更新每个点的权值。

Algorithm 6: COVER$_{cc}$

```
1  COVER_cc(G, maxSteps)
   Input: graph G = (V, E), maxSteps
   Output: vertex cover of G
2  begin
3      step := 0;
4      initialize edge weight w(e) to be 1 for each edge e;
5      initialize confChange[v] to be 1 for each vertex v;
6      construct the current candidate solution C greedily until it's a vertex cover;
7      initialize the best solution as C* := C;
8      remove a random vertex from C;
9      while step < maxSteps do
10         choose an uncovered edge e randomly;
11         choose a vertex pair u ∈ C and v ∈ endpoint(e) that u ≠ tabuRemove
           and confChange[v]=1 according to max gain criterion;
12         C := [C ∪ {v}]\{u}, update the confChange array;
13         tabuRemove := v;
14         u.time_stamp = step;
15         w(e) := w(e) + 1 for each uncovered edge e;
16         if there is no uncovered edge then
17             C* := C;
18             remove a random vertex from C;
19         step := step + 1;
20     return C*;
21 end
```

为了说明格局检测对COVER算法的有效性，我们把COVERcc和COVER在DIMACS和BHOSLIB实例上做了比较。实验的软

Graph			COVER		COVERcc	
Instance	k^*	CT(s)	suc	time	suc	time
brock200_4	183	1000	100	6.076	100	**2.779**
brock400_2	371	1000	3	228.52	**14**	404.907
brock400_4	367	1000	47	702.573	**92**	**449.948**
C1000.9	932	1000	100	6.786	100	**5.909**
C2000.5	1984	1000	100	13.958	100	**11.949**
C2000.9	1921	2400	7	1337.397	**33**	**1016.466**
C4000.5	3982	2400	100	**658.330**	100	769.003
DSJC1000.5	985	1000	100	3.268	100	**2.989**
hamming10-4	984	1000	100	1.132	**100**	**0.397**
keller5	749	1000	100	0.668	**100**	**0.019**
keller6	3302	1000	100	**68.214**	100	72.875
MANN_a45	690	2400	97	643.638	97	**537.771**
MANN_a81	2221	2400	**2**	**1875.320**	0	n/a
p_hat700-1	689	1000	100	0.643	100	**0.618**
p_hat1500-1	1488	1000	100	18.095	100	**16.430**
p_hat1500-2	1435	1000	100	4.997	100	**3.296**
p_hat1500-3	1406	1000	100	3.013	100	**2.031**

表 4.9　COVER与COVERcc算法在DIMACS实例上的比较

硬件配置和评估方法和本章的EWLS与EWCC的比较实验一致，在此不再详述。

COVERcc和COVER在DIMACS实例上的比较结果见表4.9所示。为了关注两个算法的比较，两个算法都能以100%成功率在0.01秒之内求解的实例和两个brock800实例（两个算法都求不出最优解）没有报告。从实验结果可以看出，在绝大部分DIMACS实例上，COVERcc算法无论是成功率还是平均成功时间，都优于COVER算法。在所有比较的实例中，除了C4000.5, keller6和MANN_a81，COVERcc的性能都优胜于原来的算法COVER。

COVERcc和COVER在BHOLISB实例上的比较结果见表4.10所示。为了关注两个算法的比较，我们同样不报告两组小实例的实验结果。从实验结果可以看出，在所有BHOSLIB实例上，COVERcc算法无论是成功率还是平均成功时间，都大幅度地优于COVER算法。从稳定求解的实例个数来看，COVERcc比COVER多了5个成功率为100%的实例。从平均运行时间来看，对于那些两个算法都达到100%成功率的实例，COVERcc的时间远少于COVER，在很多这样的实例上COVERcc的时间大约是COVER的一半。从

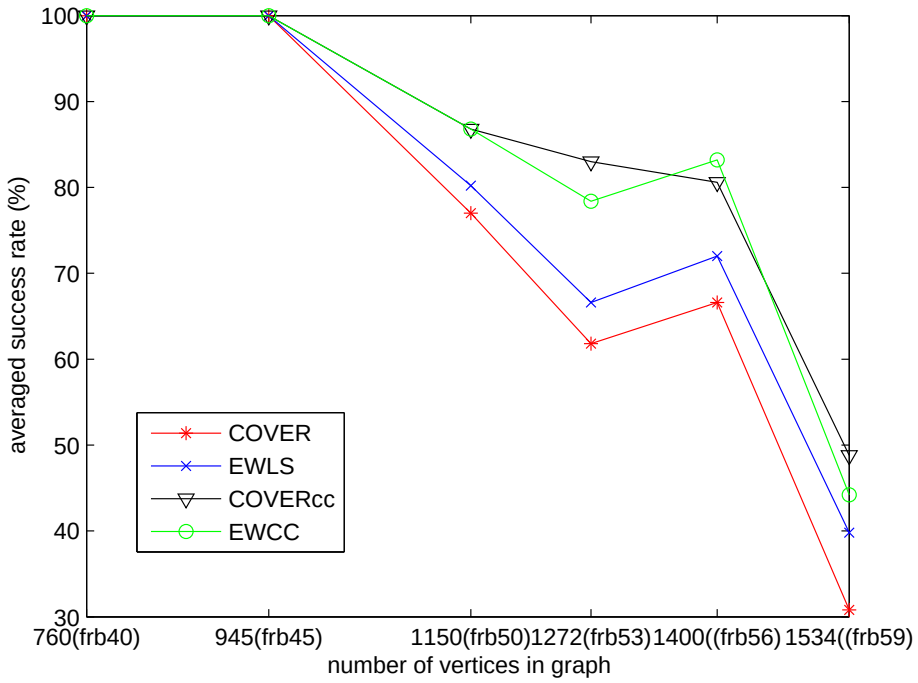

图 4.5　EWLS，COVER与EWCC，COVERcc算法在BHOSLIB实例上的平均成功率比较

平均成功率来看，除了那些两个算法都达到100%成功率的实例，COVERcc总的平均成功率大约是COVER的两倍。

通过在DIMACS和BHOSLIB两个标准测试集上的比较，足以看出，格局检测策略对COVER算法的改进幅度是相当大的。它甚至使得COVERcc在BHOSLIB上达到了和EWCC相媲美的水平。而COVERcc的实现是基于COVER代码的，所以编程技巧上没有什么不同。实际上，只是在COVER的代码上修改了若干行(不超过10行)代码。这说明了格局检测策略是一个简单易用而其显著有效的局部搜索策略。

为了比较直观地说明格局检测策略对MVC局部搜索算法的作用，我们在图4.5中总结了EWLS和COVER算法以及它们由格局检测改进的版本EWCC和COVERcc算法在BHOSLIB实例上的平均成功率。从图4.5 可以看出，两个改进算法的平均成功率都明显高于原始算法。并且，COVERcc算法已经取得了和EWCC算法不相上下的平均成功率。这说明了格局检测策略在MVC局部搜索算法上的有效性和普适性。

Graph			COVER		COVER_{cc}	
Instance	k^*	CT(s)	suc	time	suc	time
frb40-19-1	720	1000	100	1.575	100	1.495
frb40-19-2	720	1000	100	17.180	100	12.079
frb40-19-3	720	1000	100	5.509	100	4.033
frb40-19-4	720	1000	100	11.794	100	10.258
frb40-19-5	720	1000	100	124.153	100	35.708
frb45-21-1	900	1000	100	14.429	100	11.988
frb45-21-2	900	1000	100	37.692	100	18.384
frb45-21-3	900	1000	100	109.831	100	65.600
frb45-21-4	900	1000	100	20.888	100	12.981
frb45-21-5	900	1000	100	105.138	100	37.935
frb50-23-1	1100	1800	100	267.752	100	122.380
frb50-23-2	1100	1800	48	594.427	81	589.527
frb50-23-3	1100	1800	37	585.273	53	852.722
frb50-23-4	1100	1800	100	32.734	100	28.858
frb50-23-5	1100	1800	100	167.929	100	87.436
frb53-24-1	1219	2000	17	796.249	35	848.463
frb53-24-2	1219	2000	50	558.181	92	594.570
frb53-24-3	1219	2000	99	256.704	100	146.391
frb53-24-4	1219	2000	48	810.777	88	697.331
frb53-24-5	1219	2000	95	341.714	100	180.036
frb56-25-1	1344	2200	26	839.331	61	797.478
frb56-25-2	1344	2200	17	815.253	42	955.088
frb56-25-3	1344	2200	97	492.755	100	344.292
frb56-25-4	1344	2200	93	362.350	100	201.614
frb56-25-5	1344	2200	100	168.166	100	99.601
frb59-26-1	1475	2400	17	883.755	35	933.233
frb59-26-2	1475	2400	9	678.554	19	1264.880
frb59-26-3	1475	2400	26	1149.020	59	1131.190
frb59-26-4	1475	2400	4	1596.320	31	1129.340
frb59-26-5	1475	2400	98	430.602	100	220.653

表 4.10　COVER与COVERcc算法在BHOSLIB实例上的比较

4.5.2 格局检测在SAT局部搜索的应用

除了本文提到的最小顶点覆盖局部搜索算法，我们还把格局检测应用于SAT局部搜索算法，并设计了新的SAT局部搜索算法，取得了非常好的成果。基于格局检测策略，我们设计了三个高效的SAT局部搜索算法，即Swcc (Smoothed Weighting with Configuration Checking) 算法 [43]，Swqcc (Smoothed Weighting with Quantitative Configuration Checking)算法 [44]和Swcca (Smoothed Weighting and Configuration Checking with Aspiration)算法 [45]。Swcc算法在随机3SAT实例上的性能明显优于2009年SAT比赛随机可满足组的冠军算法TNM [43]。而Swcca算法则更进一步，在大量随机实例上优胜于最后的一个"地标式"（landmark）算法Sparrow2011，在结构化实例上也和最后一个"地标式"启发式算法Sattime不相上下，这在SAT算法历史上是相当罕见的 [45]。Swcca也是我们知道的解随机3SAT规模最大的算法，它在1000秒的截至时间内能以几乎100%的成功率求解10万以内的随机3SAT实例 [45]。我们还基于格局检测策略设计了一个带权MAX2SAT局部搜索算法，其性能也与最先进的带权MAX2SAT局部搜索算法相持平 [46]。我们猜测格局检测也能有效地提高其他组合搜索问题的局部搜索算法，尤其是和MVC以及SAT类似的问题，比如图着色问题等。

4.6 本章总结

局部搜索算法的循环问题严重地限制了算法的性能。本章介绍了一个新的局部搜索策略—格局检测，用于处理局部搜索中的循环问题，从而提高局部搜索算法。格局检测策略在MVC问题上具体表现为：一个点所有邻居点的状态构成了该点的格局，如果一个点v的格局自从该点离开当前候选解之后就没有改变过，那么v就不能被加入当前候选解。

我们利用格局检测策略改进了EWLS算法，得到的EWCC算法的性能比EWLS有大幅度提高。并且EWCC算法没有任何依赖于实例的参数。我们把EWCC算法与其他启发式算法，精确算法以及SAT算法做了比较，这些实验结果都说明了EWCC的优越性。

我们还通过实验分析EWLS和EWCC算法的运行时间分布和运行长度分布，来理解两个算法的运行时行为。这些运行时分析说明

了EWLS和EWCC的RLD和RTD服从指数分布，从而这两个算法对截至步数和截至时间具有很强的鲁棒性，也因此具有最佳并行属性，也即并行的时候能达到最高的加速比。我们还通过实验统计格局检测发生作用的顶点个数，以此从实验上说明格局检测的运行机制。本章的格局检测策略的实现采取的是近似实现，我们解释了格局检测所采取近似实现的原因以及该实现的单步最坏时间复杂度。

另外，格局检测是一个一般化的局部搜索策略，可以广泛地应用于设计各种组合问题的局部搜索算法。为了说明格局检测的一般有效性，我们把它运用于提高另一个MVC局部搜索算法即COVER算法，得到了新算法COVERcc。格局检测策略简单易用，COVERcc的代码仅在COVER的基础上做了几行修改，就达到了显著的提高。从标准测试实例集合DIMACS和BHOSLIB上的性能比较说明，COVERcc的性能大幅度地优于COVER算法。我们也简单介绍了格局检测策略应用于SAT和带权MAX2SAT问题的局部搜索算法所取得的显著成果。

第五章 NuMVC算法

在本章中，我们指出时下MVC局部搜索算法的两个缺陷，并介绍两个策略对其进行改进。目前的MVC局部搜索算法有两个缺陷：首先，他们在选点对进行交换的时候是两个点同时挑选的，这么做非常耗时；其次，虽然很多成功的MVC局部搜索算法都使用了边加权策略，但是这些算法都没有一个机制用于减少权值。

针对这两个不足，我们提出了两个新策略：两阶段交换和带遗忘的边加权。我们使用这两个策略设计了一个新的MVC 局部搜索算法，我们把这个新的算法叫做NuMVC [45]。实验结果表明，NuMVC在大部分难解的DIMACS实例和所有的BHOSLIB实例上都大幅度地优于已有的启发式算法。所以，NuMVC可以看作是MVC(MC,MIS)启发式算法的一个突破。

5.1 两阶段交换

本节介绍两阶段交换的框架，NuMVC算法就是基于这个框架来交换一对顶点的。

首先介绍一下交换点对在NuMVC算法中的基础性，其实对于其他MVC算法也是如此。NuMVC是一个迭代k-顶点覆盖算法，即每当NuMVC找到一个k点的顶点覆盖时，它就从当前候选解移除一个点，然后继续找(k-1)个点的顶点覆盖。这样的话，NuMVC的核心就是一个k-顶点覆盖算法，即给定一个正整数k，寻找一个大小为k的顶点覆盖。为了找到一个k顶点覆盖，NuMVC算法从一个大小为k的候选解C 开始，每一步交换集合内外一对顶点，直到C成为一个顶点覆盖。目前最好的MVC局部搜索算法如COVER，EWLS和EWCC等都采用了这个算法框架。

时下很多MVC局部搜索算法在选点对进行交换的时候都根据某启发式同时挑选两个点。比如COVER算法就挑选最大化$gain(u,v)$的一对顶点 [23]；EWLS [34]和EWCC [36]则选择$score(u,v) > 0$的一个随机顶点对。这样同时选择两个交换顶点的缺陷是，对一对顶点对评估不仅仅依赖于两个顶点各自的信息(比如$dscore$)，还依赖两个顶点对关系，如“他们是否属于同一条边”。所以评估每个候选交换点对是非常耗时的。

和之前的MVC局部搜索相反，NuMVC在选交换点对的时候是分两个

阶段的。每一个搜索步中，NuMVC首先选择一个点$u \in C$从C中移除，然后NuMVC再随机选一条边e，从e的端点中选一个点加入C。这里NuMVC对两个点是分开选取的，对它们的交换也是分两个阶段进行的。

分开选取交换的顶点可能会错过一些更贪心的交换点对（某些由两个邻居点构成的点对）。不过，这样的好处却是非常明显的，可以大大地降低MVC局部搜索算法的单步复杂度。令R和A分别表示待移除的候选点的集合和待加入的候选点的集合。同时从两个集合中各选取一个点需要的复杂度是$|R| \cdot |A|$；而分开从两个集合中各选取一个点需要的复杂度只有$|R| + |A|$。所以，采取两阶段交换的搜索框架使得算法的单步最坏复杂度从$O(n^2)$降低到$O(n)$，其中n为图的顶点个数。这个加速是非常显著的，它使得算法在给定时间内可以执行更多的搜索步。

总的来说，两阶段交换的框架减低了算法的贪心度，而显著降低了单步复杂度。值得注意的是，局部搜索的启发式策略基本都是基于直觉和经验得到的，而不是通过理论推导或实验分析得出的准确规律，我们并不能确定地说减低贪心度就是坏事 [47]。然而，对一个局部搜索算法而言，一个低的单步时间复杂度总是我们所追求和期望的。

5.2 带遗忘的边加权

本节介绍一个新的边加权策略，叫做带遗忘边加权。这个策略在NuMVC中起到核心作用。

带遗忘边加权策略的工作机制如下：每条边的权值初始化为1，接着在每一个搜索步，未覆盖的边的权值加1；而且，当所有边的平均权值达到一个阈值的时候，所有的边权都会根据公式$w(e) := \rho \cdot w(e)$(其中$\rho < 1$)来减去一些权值，以此遗忘早期的加权决定。

边加权技术，作为一种增加搜索多样性的形式，最近在MVC局部搜索算法中开始被关注和使用。比如，COVER算法 [23]每一步更新边权，而EWLS [34]和EWCC算法 [36]则在遇到局部最优的时候更新边权。不过，这些边加权技术都是在搜索过程中只会增加边权，而没有一个机制可以减少边权。一些较久之前的加权决定带来的权值对搜索不但没有作用，反而会误导搜索。带遗忘边加权和以前的边加权技术的本质不同就是它引入了一个遗忘机制，在搜索过程中定期地减少边权，从而使得搜索过程中，近期的加权决定比之前的加权决定

更重要，更能引导算法。

遗忘机制的直观解释就是那些太久以前做出的加权决定已经没有帮助甚至会误导搜索。所以，这些早期的加权决定应该是没有最近的加权决定那么重要。这里我们举一个例子来说明遗忘机制的作用。考虑两条边e_1和e_2，在某一步的时候$w(e_1) = 1000$而$w(e_2) = 100$。根据评估函数，在接下来的搜索阶段，相对于e_2而言算法更倾向于覆盖e_1。所以我们可以假设在这个阶段两条边权值的增量为$\Delta w(e_1) = 50$，$\Delta w(e_2) = 500$。这就使得它们的权值更新为$w(e_1) = 1000 + 50 = 1050$，$w(e_2) = 100 + 500 = 600$。在没有遗忘机制的情况下，算法在将来的搜索中仍然会倾向于覆盖e_1而不是e_2。在我们看来这不是有效的启发式，因为在近期之内e_1经常被覆盖，所以这么做很可能导致一些循环。实际上，近期内e_2比起e_1而言被覆盖的时间很少，因而应该优先考虑被覆盖e_2。现在考虑有遗忘机制的情况下会怎么样。这里我们设遗忘公式中的参数设置为$\rho = 0.3$，和NuMVC 算法中的设置一样。同样我们假设在执行遗忘的时候$w(e_1) = 1000$而$w(e_2) = 100$。那么执行完遗忘机制之后，权值就变为$w(e_1) = 1000 \times 0.3 = 300$和$w(e_2) = 100 \times 0.3 = 30$。过一段时期之后，同样因为$w(e_1) > w(e_2)$，则一般有$\Delta w(e_1) < \Delta w(e_2)$，所以我们可以假设$\Delta w(e_1) = 50$ and $\Delta w(e_2) = 500$。那么，权值就变为$w(e_1) = 300 + 50 = 350$，$w(e_2) = 30 + 500 = 530$。这样的话，算法在接下来的搜索中就会优先覆盖e_2而不是e_1，正如我们期望的。

5.3 NuMVC算法描述

NuMVC除了上面介绍的两个策略，还用到了上一章介绍的格局检测策略 [36]，在此我们不在详述。我们在算法7中给出了NuMVC的伪代码，下面具体地描述NuMVC算法。算法开始时做一些初始化：所有边的边权都设置为1，然后根据边权计算每个顶点的$dscore$值；每个顶点的$confChange[v]$值都设为1；当前候选解C从空集合开始通过贪心法构造，每一步把$dscore$最大的顶点加入到C中（有多个这样的顶点时随机挑一个），直至C成为一个顶点覆盖。

初始化之后，算法执行一个循环(第7-19行)直到算法运行时间达到给定的时间限制。在搜索过程中，一旦所有边都被覆盖了，也就意味这C已经是一个顶点覆盖，NuMVC就把已经找到的最优解C^*更新为C(第9行)，然后从C中移掉$dscore$值最大的顶点(第10行)，接着继续寻找更小的顶点覆盖。

在循环中的每一次迭代，NuMVC根据两阶段交换策略交换两个顶点(第12-16行)。具体的，它首先选择C中具有最大$dscore$值的一个顶点然后把该点从C中移除，如果存在多个这样的顶点，就选取最老的那个。移除顶点u之后，NuMVC选择一条随机的未覆盖的边e，然后按照以下方法从它的两个端点中选取一个来加入C：如果e的端点中只有一个点满足$confChange[v] = 1$，那么就选择那个点；如果两个端点都满足这个条件，那么NuMVC就选择$dscore$值更大的那个顶点；如果两个端点不仅都满足$confChange[v] = 1$，而且它们的$dscore$值也一样大，那么就选择比较老的那个顶点。当从e的端点中选出这样的顶点时，NuMVC把它加入到当前候选解C。这就完成了点对交换的整个流程。在这个流程中，$confChange$数组也伴随着被更新。

在每一次迭代的最后，NuMVC更新边权值(第17-18行)。所有没有被覆盖的边的权值都加1。并且，如果所有边的平均权值超过了某个阈值，则所有边的权值都乘以一个常数因子ρ ($\rho < 1$) 来减少权值。更新后的权值要向下取整，因为在NuMVC里面边权是定义为一个整数的。

本节最后，我们给出一个定理。该定理保证了NumVC算法中的第15行伪代码的可执行性。

定理 5.1: 对于一条未覆盖边e, 至少有一个顶点$v \in e$ 满足$confChange[v] = 1$。

证明: 令$e = \{v_1, v_2\}$，本证明分两种情况。
(a)从初始化之后，v_1和v_2至少有一个点就没有改变过状态。不失一般性地，我们假设v_1就是这样的点。在初始化的时候，我们设置了$confChange[v_1] = 1$。从那以后，只有把v_1从C中移除(对应着v_1的状态s_v从0变为1)才会使得$confChange[v_1] = 0$。但是根据我们的假设v_1从初始化之后就没有改变过状态，所以$confChange[v_1] = 1$。
(b)从初始化之后，v_1和v_2都改变了状态。既然$e(v_1, v_2)$是没有被覆盖的，则$v_1 \notin C$且$v_2 \notin C$。不失一般性地，我们假设v_1的最后一次被移除发生在v_2最后一次被移除之前。当v_1最后一次被移除发生的时候，有$v_2 \in C$。之后，v_2也被移除，也就是说v_2改变了状态，所以$confChange[v_1]$就会被设置为1，因为$v_1 \in N(v_2)$。□

Algorithm 7: NuMVC

NuMVC (*G,cutoff*)
Input: graph $G = (V, E)$, the *cutoff* time
Output: vertex cover of G
begin
  initialize edge weights and $dscores$ of vertices;
  initialize the $confChange$ array as an all-1 array;
  construct C greedily until it is a vertex cover;
  if *elapsed time* $\geq$ *cutoff* **then return** C;
  while *elapsed time* $<$ *cutoff* **do**
    if *there is no uncovered edge* **then**
      $C^* := C$;
      remove a vertex with the highest $dscore$ from C;
      continue;
    choose a vertex $u \in C$ with the highest $dscore$, breaking ties in favor of the oldest one;
    $C := C \backslash \{u\}$, $confChange(u) := 0$ and $confChange(z) := 1$ for each $z \in N(u)$;
    choose an uncovered edge e randomly;
    choose a vertex $v \in e$ such that $confChange[v] = 1$ with higher $dscore$, breaking ties in favor of the older one;
    $C := C \cup \{v\}$, $confChange(z) := 1$ for each $z \in N(v)$;
    $w(e) := w(e) + 1$ for each uncovered edge e;
    if $\overline{w} \geq \gamma$ **then** $w(e) := \lfloor \rho \cdot w(e) \rfloor$ for each edge e;
  return C^*;
end

5.4 实验分析

本节首先介绍实验采取的相关设置；接着对算法在DIMACS和BHOSLIB两个基准实例组上进行评估。对每个实例组，我们都将NuMVC和当前最好的启发式求解器做比较。从最近的文献看，目前在求解MVC (MC, MIS)问题上领先的启发式算法有5个：三个MVC局部搜索算法COVER [23]，EWLS [34]和EWCC [36]，以及两个MC局部搜索算法DLS-MC [4]和PLS [40]。其中，EWCC和PLS分别是EWLS和DLS-MC的改进版本，并且它们的性能比原有版本更好。所以，我们只将NuMVC和PLS，COVER以及EWCC三个算法做比较。

5.4.1 实验设置

- **实现：** NuMVC由C++语言实现；其他用于比较的算法都是由作者提供源码或从网络下载，它们的源码也是由C++语言实现。所有的算法的源程序都用g++编译器带O2选项编译。
- **NuMVC参数设置：** 在本工作的实验中，我们设$\gamma = |V|/2$，$\rho = 0.3$。在挑战实例frb100-40上，我们设$\gamma = 5000$，$\rho = 0.3$。这两个参数是通过初步实验调出来的。
- **实验平台：** 所有实验都在一台个人计算机上运行，其配置是3 GHz Intel Core 2 Duo CPU E8400和4GB RAM，操作系统是Linux。为了得到该机器的性能，我们在该机器上运行了DIMACS机器基准实例 [75]。这台机器求解r300.5实例需要0.19秒，求解r400.5实例需要1.12秒，求解r500.5实例需要4.24秒。
- **停止标准：** 本章的实验是基于给定时间限制的停止标准，所有实例的时间限制统一为2000秒。
- **实验报告方法：** 对于每个实例，每个算法都执行100次不同随机种子的独立运行。对NuMVC算法在每个实例上的运行，我们报告以下信息：
  - 最小（或已知最小）顶点覆盖的大小(k^*)；
  - 算法成功找到k^*解的次数(“suc”)；
  - 算法在100次运行里找到的最小（平均，最大）顶点覆盖规模(“VC size”) ；
  - 算法的平均运行时间(“time”)，对于成功的运行，运行时间为找到k^*解需要的时间，对于失败的运行，其运行时间以cutoff时间

即2000秒计算。对于那些成功率不是100%的实例，我们还报告平均成功运行时间(“suc time”)，是指所有成功运行的平均时间；

– 算法的平均运行步数(“steps”)，对于成功的运行，运行时间为找到k^*解需要的步数，对于失败的运行，其运行时间到cutoff时所执行的步数。对于那些成功率不是100%的实例，我们还报告平均成功运行步数(“suc step”)，是指所有成功运行的平均运行步数。如果算法没能找到一个k^*解，则“time”和“steps”标记“n/a”。

5.4.2 NuMVC性能

本节详细报告NuMVC算法求解DIMACS和BHOSLIB两个组benchmark的性能。

从表格5.1可以看出，除了两个brock实例外，NuMVC对所有DIMACS实例都找到了最优解。在35个找到最优解的实例中，有32个是以100%的成功率求得最优解的，而这其中有24个实例的平均运行时间小于0.1秒。这足可显示NuMVC求解DIMACS的高效性能。对于C2000.9实例，虽然NuMVC只有一次运行找到已知最小解1920，但是就我们所知，这是第二个能对C2000.9找到1920点的顶点覆盖（等价于80点的团）的算法，在文献 [63]中第一次求出该实例一个80点的团。另外对MANN_a81这个困难实例，NuMVC则达到了最好的性能，成功率远远超过了以前能求得2221规模顶点覆盖的几个算法，包括COVER，EWLS和EWCC。

NuMVC在BHOSLIB实例的表现上更是明显地显示了该算法非常高效的求解性能，可以说，对BHOSLIB实例组，NuMVC取得了一次大的飞跃。对所有40个BHOSLIB实例，NuMVC算法都找到最优解，其中33个实例达到100%成功率。对于没有达到100%成功率的实例，其成功率也是相当高的。事实上，除了frb59-26-2，其他实例都有希望通过适当加大运行时间（比如把时间限制设置为1小时）来取得100%成功率。这对以前的算法来说基本是不可能的。对于BHOSLIB这组以高难度著称的benchmark，NuMVC的表现可以说是求解BHOSLIB实例一个里程碑了。

5.4.3 比较实验结果

本节比较NuMVC和当前最好的MVC，MC局部搜索算法

Graph			NuMVC			
Instance	Vertices	k^*	suc	VC size	time(suc time)	steps(suc steps)
brock200_2	200	188*	100	188	0.126	137610
brock200_4	200	183*	100	183	1.259	1705766
brock400_2	400	371*	96	371(371.16,375)	572.390(512.906)	645631471(585032783)
brock400_4	400	367*	100	367	4.981	4.981
brock800_2	800	776*	0	779	n/a	n/a
brock800_4	800	774*	0	779	n/a	n/a
C125.9	125*	91	100	91	< 0.001	136
C250.9	250*	206	100	206	< 0.001	3256
C500.9	500	443	100	443	0.128	133595
C1000.9	1000	932	100	932	2.020	1154155
C2000.5	2000	1984	100	1984	2.935	231778
C2000.9	2000	1920	1	1920(1921.29,1922)	1994.561(1393.303)	777848959(564895994)
C4000.5	4000	3982	100	3982	152.807	7802785
DSJC500.5	500	487*	100	487	0.012	3800
DSJC1000.5	1000	985*	100	985	0.615	134796
gen200_p0.9_44	200	156*	100	156	< 0.001	1695
gen200_p0.9_55	200	145*	100	145	< 0.001	69
gen400_p0.9_55	400	345*	100	345	0.035	38398
gen400_p0.9_65	400	335*	100	335	< 0.001	1522
gen400_p0.9_75	400	325*	100	325	< 0.001	203
hamming8-4	256	240*	100	240	< 0.001	1
hamming10-4	1024	984*	100	984	0.062	23853
keller4	171	160*	100	160	< 0.001	42
keller5	776	749*	100	749	0.038	15269
keller6	3361	3302	100	3302	2.51	384026
MANN_a27	378	252*	100	252	< 0.001	6651
MANN_a45	1035	690*	100	690	86.862	90642150
MANN_a81	3321	2221	27	2221(2221.94,2223)	1657.880(732.897)	571607432(251509010)
p_hat300-1	300	292*	100	292	0.003	100
p_hat300-2	300	275*	100	275	< 0.001	98
p_hat300-3	300	264*	100	264	0.001	1863
p_hat700-1	700	689*	100	689	0.011	1248
p_hat700-2	700	656*	100	656	0.006	1103
p_hat700-3	700	638*	100	638	0.008	2868
p_hat1500-1	1500	1488*	100	1488	3.751	445830
p_hat1500-2	1500	1435*	100	1435	0.071	5280
p_hat1500-3	1500	1406	100	1406	0.060	10668

表 5.1　NuMVC在DIMACS实例上的性能，k^*列标注星号的表示k^*已经被证明是最优解的规模

在DIMACS和BHOSLIB实例上的性能。在报告比较结果时，我们只报告成功率(“suc”)和平均运行时间(“time”)。我们这里不报告平均成功时间，因为这个统计量只有在比较的算法具有很接近的成功率的时候才能比较准确的反映求解速度；而且平均成功时间也可以有平均时间和时间限制按公式$\frac{\text{“}time\text{”}*100-cutoff*(100-\text{“}suc\text{”})}{\text{“}suc\text{”}}$计算出来。本文作者提倡，在随机算法的实验分析中，应该使用平均时间而不是平均成功时间来衡量其求解速度。

DIMACS实例实验结果

表5.3显示了在DIMACS实例上的实验结果。大部分的DIMACS实例对于这

Graph			NuMVC			
Instance	Vertices	k^*	suc	VC size	time (suc time)	steps (suc steps)
frb30-15-1	450	420	100	420	0.045	37963
frb30-15-2	450	420	100	420	0.053	44632
frb30-15-3	450	420	100	420	0.191	173708
frb30-15-4	450	420	100	420	0.049	41189
frb30-15-5	450	420	100	420	0.118	105468
frb35-17-1	595	560	100	560	0.515	386287
frb35-17-2	595	560	100	560	0.447	334255
frb35-17-3	595	560	100	560	0.178	129279
frb35-17-4	595	560	100	560	0.563	422638
frb35-17-5	595	560	100	560	0.298	218800
frb40-19-1	760	720	100	720	0.242	208115
frb40-19-2	760	720	100	720	4.083	3679770
frb40-19-3	760	720	100	720	1.076	959874
frb40-19-4	760	720	100	720	2.757	2473081
frb40-19-5	760	720	100	720	10.141	9142719
frb45-21-1	945	900	100	900	2.708	2029588
frb45-21-2	945	900	100	900	4.727	3605881
frb45-21-3	945	900	100	900	13.777	10447444
frb45-21-4	945	900	100	900	3.973	3000680
frb45-21-5	945	900	100	900	10.661	8059236
frb50-23-1	1150	1100	100	1100	38.143	24628019
frb50-23-2	1150	1100	100	1100	176.589	113569606
frb50-23-3	1150	1100	95	1100(1100.05,1101)	606.165(532.805)	386342329(343518242)
frb50-23-4	1150	1100	100	1100	7.89	5092072
frb50-23-5	1150	1100	100	1100	19.529	12690957
frb53-24-1	1272	1219	86	1219(1219.14,1220)	895.006(715.123)	514619149(416394360)
frb53-24-2	1272	1219	100	1219	205.352	117980833
frb53-24-3	1272	1219	100	1219	51.227	29376406
frb53-24-4	1272	1219	100	1219	266.871	152982736
frb53-24-5	1272	1219	100	1219	39.893	22817023
frb56-25-1	1400	1344	100	1344	470.682	259903023
frb56-25-2	1400	1344	97	1344(1344.03,1345)	658.961(617.485)	350048132(326853745)
frb56-25-3	1400	1344	100	1344	121.298	67043078
frb56-25-4	1400	1344	100	1344	49.446	26030031
frb56-25-5	1400	1344	100	1344	26.761	14109165
frb59-26-1	1534	1475	88	1475(1475.12,1476)	843.304(687.845)	440874471(350993718)
frb59-26-2	1534	1475	38	1475(1475.62,1476)	1677.801(1160.020)	875964146(592010913)
frb59-26-3	1534	1475	96	1475(1475.04,1476)	644.831(580.032)	325417225(295226277)
frb59-26-4	1534	1475	79	1475(1475.21,1476)	1004.550(741.208)	517521634(375976753)
frb59-26-5	1534	1475	100	1475	61.907	31682895

表 5.2　NuMVC在BHOSLIB实例上的性能

四个算法而言都太容易了，可以在2秒之内以100%成功率找到最优解，因此没有报告那些实例的结果。

结果显示，在这些难解的DIMACS实例上，NuMVC都表现得比COVER和EWCC更好，而与PLS则是各自有自己占优的实例。NuMVC在C2000.9，keller6，MANN_a45，MANN_a81和gen400_p0.9_55这5个实例上显著地优胜于PLS，其中C2000.9和MANN_a81是公认的两个最难的DIMACS实例 [23, 36, 63]。对于C2000.9，只有NuMVC找到了1920个点的解；并且，NuMVC还有70次运行找到了1921个点的解，而PLS，COVER和EWCC分

Graph		PLS		COVER		EWCC		NuMVC	
Instance	k^*	suc	time	suc	time	suc	time	suc	time
brock400_2	371	**100**	**0.15**	3	1947	20	1778	96	572
brock400_4	367	100	**0.03**	82	960	100	25.38	100	4.89
brock800_2	776	**100**	**3.89**	0	n/a	0	n/a	0	n/a
brock800_4	774	**100**	**1.31**	0	n/a	0	n/a	0	n/a
C2000.9	1920	0	n/a	0	n/a	0	n/a	**1**	**1994**
C4000.5	3982	100	**67**	100	658	100	739	100	152
gen400_p0.9_55	345	100	15.17	100	0.35	100	0.05	100	**0.02**
keller6	3302	92	559	100	68	100	3.76	**100**	**2.37**
MANN_a45	690	1	1990	94	714	88	763	**100**	**86**
MANN_a81	2221	0	n/a	1	1995	1	1986	**27**	**1657**
p_hat1500-1	1488	100	**2.36**	100	18.10	100	9.79	100	3.15

表 5.3　NuMVC和其他算法在DIMACS实例的比较

别只有31，6和32次运行找到1921点的解。另外，NuMVC表现不好的实例都集中在$brock$图上，这是一种人工构造图，专门用于对抗带度贪心的启发式算法的。

BHOSLIB实例实验结果

表5.4总结了几个算法在BHOSLIB实例上的性能。与前面两章类似，为了关注于算法的性能差距，我们不报告两组小实例($frb30$和$frb35$)的实验结果。NuMVC在这两组小实例上也是大幅度领先的。

结果显示NuMVC在所有的BHOSLIB实例上，不管是考虑成功率还是平均运行时间，其性能都显著地比所有其他算法好。我们进一步考察NuMVC和EWCC之间的性能差距，因为EWCC在本组实例上明显优于其他两个算法PLS和COVER。首先我们比较40个实例中成功率达到100%的实例个数，NuMVC在33个实例上达到了100%成功率，而EWCC达到这种效果的只有29个实例；其次，对于两个算法都以100%成功率求解的所有实例的运行，NuMVC的平均运行时间是25秒，而EWCC的平均运行时间是74秒，大约是NuMVC的3倍；最后，对于其他实例，NuMVC达到了90%的平均成功率，而EWCC的平均成功率只有50%。关于这几个算法在BHOSLIB各组实例上的平均运行时间和平均成功率的详细比较，分别见图5.1和图5.2。从这两个图也可以看出NuMVC在BHOSLIB各组实例上显著地领先于其他算法。

Graph		PLS		COVER		EWCC		NuMVC	
Instance	k^*	suc	time	suc	time	suc	time	suc	time
frb40-19-1	720	100	10.42	100	1.58	100	0.55	100	**0.24**
frb40-19-2	720	100	85.25	100	17.18	100	11.30	100	**4.08**
frb40-19-3	720	100	9.06	100	5.06	100	2.97	100	**1.07**
frb40-19-4	720	100	77.39	100	11.79	100	13.79	100	**2.76**
frb40-19-5	720	95	496	100	124	100	41.71	100	**10.14**
frb45-21-1	900	100	52.31	100	14.34	100	9.07	100	**2.71**
frb45-21-2	900	100	170	100	38	100	15	100	**5**
frb45-21-3	900	21	1737	100	110	100	56	100	**14**
frb45-21-4	900	100	111	100	21	100	15	100	**4**
frb45-21-5	900	100	261	100	105	100	42	100	**11**
frb50-23-1	1100	30	1658	100	268	100	124	100	**38**
frb50-23-2	1100	3	1956	48	1325	82	905	**100**	**177**
frb50-23-3	1100	2	1989	39	1486	56	1348	**95**	**606**
frb50-23-4	1100	100	93	100	33	100	24	100	**8**
frb50-23-5	1100	79	967	100	168	100	85	100	**19**
frb53-24-1	1219	1	1982	17	1796	30	1696	**86**	**895**
frb53-24-2	1219	6	1959	50	1279	81	1006	**100**	**205**
frb53-24-3	1219	20	1771	99	273	100	117	100	**51**
frb53-24-4	1219	21	1782	48	1428	81	900	**100**	**266**
frb53-24-5	1219	10	1955	95	423	100	125	100	**40**
frb56-25-1	1344	1	1993	24	1698	56	1268	**100**	**470**
frb56-25-2	1344	0	n/a	17	1598	52	1387	**97**	**659**
frb56-25-3	1344	0	n/a	97	537	100	285	100	**121**
frb56-25-4	1344	11	1915	93	476	100	183	100	**50**
frb56-25-5	1344	27	1719	100	168	100	80	100	**27**
frb59-26-1	1475	0	n/a	16	1607	21	1778	**88**	**843**
frb59-26-2	1475	0	n/a	9	1881	7	1930	**37**	**1677**
frb59-26-3	1475	3	1978	21	1768	64	1294	**96**	**636**
frb59-26-4	1475	0	n/a	3	1980	20	1745	**79**	**1004**
frb59-26-5	1475	30	1708	98	431	100	174	100	**62**

表 5.4　NuMVC和其他算法在BHOSLIB实例的比较

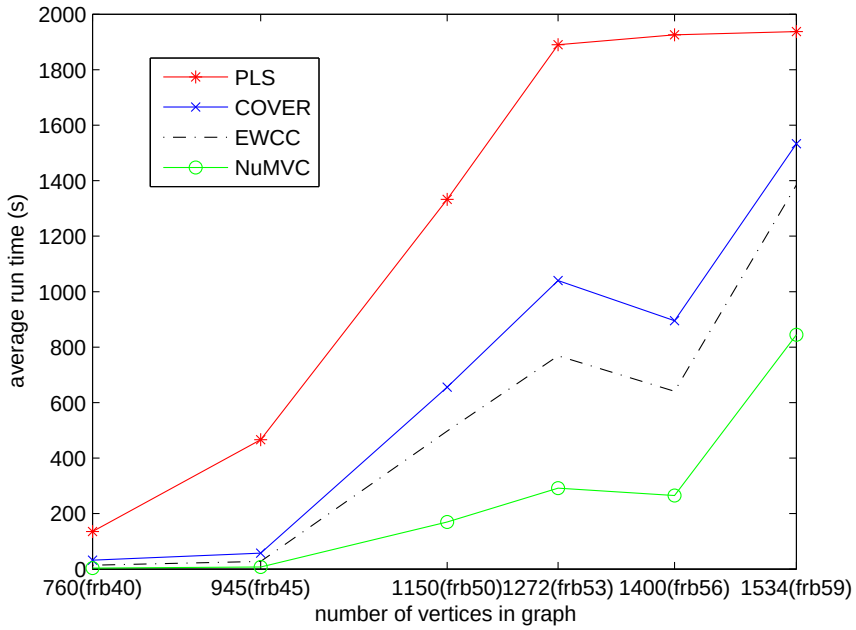

图 5.1　NuMVC与其他算法在BHOSLIB实例上的平均运行时间比较

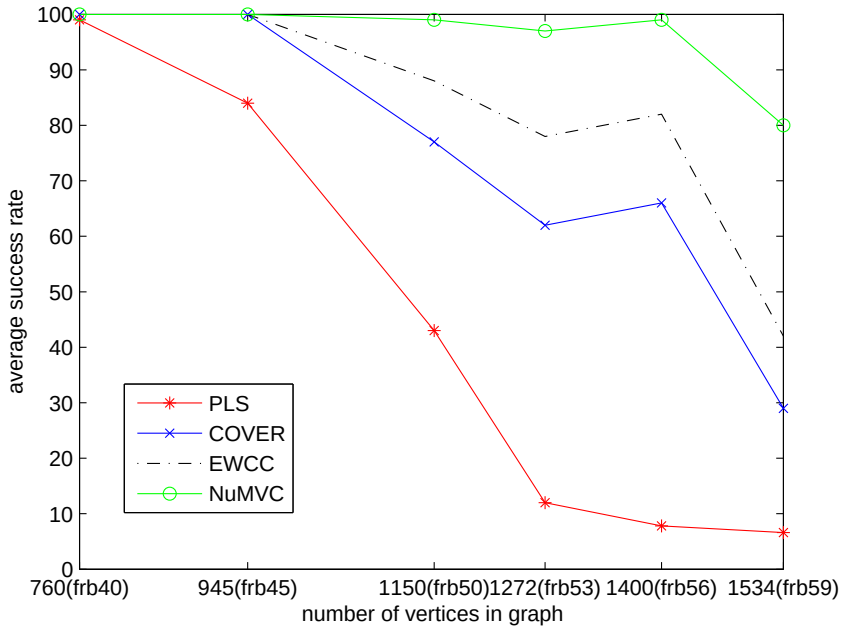

图 5.2　NuMVC与其他算法在BHOSLIB实例上的平均成功率比较

Solver	3902		≤ 3903	
	suc	time	suc	time
COVER	0	n/a	33	2768
EWCC	1	2856	79	2025
NuMVC	4	2955	93	1473

表 5.5　NuMVC与其他算法在frb100-40上的比较

NuMVC的卓越性能在困难实例上更明显地凸现出来。对于那些没有一个算法能达到100%成功率的实例，也就是可以认为是比其他实例都难的实例，在这些实例上，NuMVC的平均成功率为82.57%，远远地超过其他算法的平均成功率：PLS，COVER和EWCC的平均成功率分别为0.85%，17.43%和35.71%。NuMVC在这些实例的平均成功率分别是PLS，COVER和EWCC的97倍，4.7倍和2.3倍。可以说，NuMVC在求解BHOSLIB实例上取得了一次突破。

另外，我们还注意到，NuMVC在BHOSLIB实例上的性能比最近的一个并行算法CLS [84]的4核版本还好，即使我们不把NuMVC的运行时间除以4。如果我们考虑了机器速度比，然后把NuMVC的运行时间除以4，那么NuMVC 则是远远地优于CLS算法。实际上，简单地对NuMVC同时运行4个线程就能达到很好的效果，比CLS好许多。

对于挑战实例frb100-40，我们对COVER，EWCC和NuMVC各自运行了100次，时间限制为4000秒。PLS因为在BHOSLIB实例上表现太差，因此没有在此实例上运行。表5.5 显示NuMVC在此挑战实例上明显优于COVER和EWCC。在NuMVC的100次运行中，有4次找到了3902个点的顶点覆盖，平均时间为2955秒，总共有93此次找到了3903点或更优的解，平均时间为1473秒。这样的结果也是我们知道的对frb100-40 最好的求解结果了。

5.4.4 对两阶段交换策略的实验分析

相对于之前的MVC局部搜索算法而言，NuMVC算法采用的两阶段策略大大降低了算法的单步复杂度。这一节我们通过在几个典型实例上比较NuMVC和其他算法的单位时间运行步数进行比较，以此支持该观点。

从表格5.6可以看出，NuMVC算法在单位时间内执行的步数远远多于

Graph Instance	PLS #steps/sec	COVER #steps/sec	EWCC #steps/sec	NuMVC #steps/sec
brock400_2	1,344,161	310,881	402,819	759,578
C4000.5	85,318	8,699	11,927	513,307
MANN_a45	1546,625	279,514	578,656	991,476
p_hat_1500-1	170,511	19,473	34,111	297,220
frb53-24-5	841,346	128,971	219,038	570,425
frb56-25-5	801,282	116,618	199,441	522,561
frb59-26-5	706,436	108,534	189,536	511,014

表 5.6 Complexity per step on selected instances

其他两个MVC局部搜索算法COVER和EWCC。单位时间内NuMVC运行步数是COVER和EWCC的几倍到几十倍。这说明两阶段交换策略确实明显地降低了MVC局部搜索算法的单步复杂度。另一方面，虽然PLS在单位时间内可以执行比NuMVC更多的步数，然而PLS是MC局部搜索算法，采取的算法框架和MVC局部搜索算法本质上不一样，因此不具备可比性。实际上，虽然PLS的单步执行效率比较高，但由于算法的收敛到最优解需要的步数远远大于MVC局部搜索算法，所以PLS的性能仍然大幅度落后于NuMVC算法。

5.4.5 对遗忘机制的实验分析

为了说明遗忘机制在NuMVC算法中的作用，我们把NuMVC和NuMVC-在一些典型实例上做了比较。从表5.7可以看出，没有遗忘机制的NuMVC-在这些实例上明显落后于NuMVC算法。这尤其可以从成功率上的差距看出来。NuMVC-的成功率明显小于NuMVC的，比如在3个frb59实例上，NuMVC-的成功率仅为40%，10% 和90%，而NuMVC则达到了88%，37% 和96%。在那些成功率都为100%的实例上，NuMVC的平均求解时间则比NuMVC-的更快。

5.5 讨论

NuMVC使用的边加权的遗忘机制是受SAT局部搜索算法的子句加权的平滑机制所启发而提出的。然而，它和SAT局部搜索算法的平滑技术是不同的。

根据子句权值被平滑的方法，就我们所知SAT平滑技术可以主要分为三类：第一种是通过公式$w_i := \rho \cdot w_i + (1-\rho) \cdot \overline{w}$(其中$\overline{w}$是平均子句权值)把所有

Graph		NuMVC		NuMVC-	
Instance	k^*	suc rate	time	suc rate	time
brock400_2	371	**96%**	**572**	20%	1780
C4000.5	3982	100%	**152**	100%	270
MANN_a45	690	**100%**	**86**	50%	1287
p_hat_1500-1	1488	100%	3.75	100%	5.35
frb50-23-1	1100	100%	**38**	100%	52
frb50-23-2	1100	**100%**	**177**	100%	284
frb50-23-3	1100	**95%**	**606**	70%	1021
frb53-24-1	1219	**86%**	**895**	70%	1259
frb53-24-2	1219	**100%**	**205**	100%	227
frb53-24-3	1219	100%	**51**	100%	87
frb56-25-1	1344	**100%**	**470**	90%	781
frb56-25-2	1344	**97%**	**659**	60%	1283
frb56-25-3	1344	100%	**121**	100%	153
frb59-26-1	1475	**88%**	**843**	40%	1600
frb59-26-2	1475	**37%**	**1677**	10%	1952
frb59-26-3	1475	**96%**	**636**	90%	957

表 5.7　NuMVC和NuMVC-在BHOSLIB实例的比较,NuMVC-是NuMVC去掉遗忘机制的版本

子句权值都拉回到平均权值附近，比如ESG [85], SDF [71]和SPAS等算法 [72]；第二种是对所有权值大于1的子句，其权值都减去1，比如DLM [70]和PAWS算法 [73]；最后一种比较少见，即从邻居的满足子句权值转移到不满足子句的权值去，如DDWF算法 [86]。

NuMVC算法中的遗忘机制和上述的这些平滑机制不一样，它是通过公式$w(e) := \rho \cdot w(e)$(其中$\rho < 1$)定期地减小每个条边的权值，从而达到遗忘的效果。另外，这个遗忘是在所有边权平的均值达到一个阈值时执行遗忘机制。这都是我们在文献中没有见过的，所以这个遗忘机制是一个新的策略。实际上，我们也做过一些实验，把NuMVC算法的遗忘机制换成我们讨论的那些平滑机制，得到的算法性能都比NuMVC明显差。

5.6 本章总结

本章提出了两个新的局部搜索策略：两阶段点对交换和带遗忘边加权，并基于这两个技术设计了一个MVC局部算法，即NuMVC。这里对这两个新策略

做一个简单的回顾：

- 两阶段点对交换：以前的算法都是根据某启发式同时挑选两个（待交换的）点，本策略在不同阶段基于不同启发式挑选移除的点和加入的点。该策略大大地降低了算法的单步复杂度，使得算法在给定时间内可以执行更多的搜索，从而提高算法的时间性能。
- 带遗忘边加权：以前的边加权技术只增加权值而不减少，这样以前的加权决定会误导搜索。带遗忘边加权策略在每一步对所有未被覆盖的边权值加1，并且当所有边的平均权值达到一个阈值的时候，所有边权都会根据公式$w(e) := \rho \cdot w(e)$(其中$\rho < 1$)来减去一些权值，以此遗忘早期的加权决定。通过这个策略，近期的加权决定对搜索的方向起到更重要的引导，使得算法免受早期加权决定的影响。

我们在标准测试实例DIMACS和BHOSLIB上评估了NuMVC的性能，并且和已知的世界上最好的MVC (MC, MIS) 局部搜索算法做比较。实验结果表明，NuMVC在大多数的DIMACS难解实例和所有的BHOSLIB实例，都比已知的最好算法更优。而且，在难解的随机实例集BHOSLIB上，NuMVC算法的性能比其他算法都好几倍到100倍。基于实验结果，我们可以说NuMVC算法是MVC问题求解的一个突破，代表了求解MVC (MC, MIS)问题的一个新高度。

本章最后还分析了NuMVC算法中两个新策略即两阶段交换和遗忘机制对算法性能的影响，肯定了它们的作用，并讨论了本章提出的遗忘机制和SAT局部搜索中的平滑机制的相似和区别。

第六章　总结与进一步工作

6.1 本文内容总结

最小顶点覆盖(MVC)问题是经典的NP难组合优化问题，有着重要的理论和现实意义。局部搜索算法是现实求解MVC问题的一个有效途径。本文研究最小顶点覆盖(MVC)问题的局部搜索算法。我们提出了新的MVC局部搜索算法策略，并基于这些策略设计了高效的MVC局部搜索算法。具体而言，本文的创新之处体现在以下几个方面。

(1) 提出了扩展部分顶点覆盖的策略，首先寻找一个好的部分顶点覆盖，然后把它扩展为完整的顶点覆盖。
(2) 提出了新的边加权技术，在遇到局部最优的时候对未覆盖的边权值加1。
(3) 基于(1),(2)策略提出了EWLS算法。该算法刷新了挑战实例frb100-40的求解纪录，找到了一个比以前纪录更小的顶点覆盖。
(4) 提出了可广泛用于提高组合问题局部搜索算法的格局检测策略。利用该策略提高了EWLS和COVER算法，得到新EWCC和COVERcc算法。我们还基于该策略设计了两个高效的SAT局部搜索算法，其中Swcca算法的性能达到国际领先水平。
(5) 提出了两阶段交换点对的框架，在该框架下两个交换顶点是分两个阶段由不同启发式选取的，比以前的同时同启发式选取两个交换顶点更高效。
(6) 提出了带遗忘机制的边加权技术，这是首次把遗忘机制引入到边加权局部搜索算法中。
(7) 基于(4),(5),(6)策略提出NuMVC算法，从实验结果看该算法是MVC问题求解的一个突破。

以上创新性研究成果都已经以会议论文或期刊论文的形式得到了发表。

6.2 进一步工作

本文的进一步工作主要基于两个出发点，一是设计更高效的MVC局部搜索算法；二是注重局部搜索策略在不同问题上的应用以及对这些策略的理解和分析。具体可以从以下几个方面展开。

(1) BHOSLIB实例是一组难解实例，该组实例具有各种问题(皆为NP难问题)的等价版本，是各种算法的求解能力的一个试金石。从该实例组被提出以来，学者们就从不同问题的角度来提高求解它们的能力。我们提出的NuMVC算法站在了本方向的前沿。我们将设计更高效的算法，在合理时间(比如1小时)内以100%的成功率求解该组实例的每一个实例；并争取刷新挑选实例frb100-40的求解纪录。

(2) 近几十年SAT问题的局部搜索算法有很大的进展，而MVC问题和SAT问题有很多类比之处。未来可以考虑借鉴SAT或其他CSP问题的求解技术来设计更高效的MVC算法。例如，提出更好的边加权技术，局部搜索框架，各种不同策略间的切换策略等。

(3) 格局检测是一个一般化的局部搜索策略，可广泛应用于各种组合问题。我们已经把格局检测成功应用于SAT局部搜索算法，所设计的算法Swcca在大量随机实例上优胜于最后的一个"地标式"（landmark）算法Sparrow2011，在结构化实例上也和最后一个"地标式"启发式算法Sattime不相上下，这在SAT算法历史上是相当罕见的 [45]。利用格局检测策略来提高其他问题的局部搜索算法是一个有意义的方向。

(4) 格局检测策略在多个问题上都取得成功应用，而目前却只有直观解释，对于其背后的原理仍然有待探索。可以考虑把格局检测策略加入到一个简单的局部搜索算法中，对格局检测策略在算法中起到的作用做出严格的分析。

参考文献

[1] Reinhard Diestel. *Graph Theory Third Edition*. Springer-Verlag, NewYork, 2006.

[2] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, CA, USA, 1979.

[3] Fernando C. Gomes, Cláudio Nogueira de Meneses, Panos M. Pardalos, and Gerardo Valdisio R. Viana. Experimental analysis of approximation algorithms for the vertex cover and set covering problems. *Computers & OR*, 33(12):3520–3534, 2006.

[4] Wayne Pullan and Holger H. Hoos. Dynamic local search for the maximum clique problem. *J. Artif. Intell. Res. (JAIR)*, 25:159–185, 2006.

[5] Yongmei Ji, Xing Xu, and Gary D. Stormo. A graph theoretical approach for predicting common rna secondary structure motifs including pseudoknots in unaligned sequences. *Bioinformatics*, 20(10):1603–1611, 2004.

[6] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(2):439–486, 2005.

[7] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1508–1623, 2002.

[8] G. Karakostas. A better approximation ratio for the vertex cover problem. In *Proc. of ICALP-05*, pages 1043–1050, 2005.

[9] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math*, 182:105–142, 1999.

[10] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[11] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proc. of STOC-06*, pages 681–690, 2006.

[12] Uriel Feige. Approximating maximum clique by removing subgraphs. *SIAM J. Discrete Math.*, 18(2):219–225, 2004.

[13] R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9(6):375–382, 1990.

[14] T. Fahle. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In *Proc. of ESA-02*, pages 485–498, 2002.

[15] Patric R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[16] Jean Charles Régin. Using constraint programming to solve the maximum clique problem. In *Proc. of CP-03*, pages 634–648, 2003.

[17] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. Global Optimization*, 44(2):311, 2009.

[18] Chumin Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxSAT for the maximum clique problem. In *Proc. of AAAI-10*, pages 128–133, 2010.

[19] Chu-Min Li and Zhe Quan. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *ICTAI (1)*, pages 344–351, 2010.

[20] I.k. Evans. An evolutionary heuristic for the minimum vertex cover problem. In *Proceedings of the Seven International Conference on Evolutionary Programming(EP)*, pages 377–386, 1998.

[21] Shyong Jian Shyu, PengYeng Yin, and Bertrand M. T. Lin. An ant colony optimization algorithm for the minimum weight vertex cover problem. *Annals OR*, 131(1-4):283–304, 2004.

[22] Stephen Gilmour and Mark Dras. Kernelization as heuristic structure for the vertex cover problem. In *ANTS Workshop*, pages 452–459, 2006.

[23] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In *Proc. of KI-07*, pages 412–426, 2007.

[24] CC Aggarwal, JB Orlin, and RP Tai. Optimized crossover for the independent set problem. *Operations Research*, 45:226–234, 1997.

[25] Stanislav Busygin, Sergiy Butenko, and Panos M. Pardalos. A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. *J. Comb. Optim.*, 6(3):287–297, 2002.

[26] Valmir C. Barbosa and Luciana C. D. Campos. A novel evolutionary formulation of the maximum independent set problem. *J. Comb. Optim.*, 8(4):419–437, 2004.

[27] Diogo Viera Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. In *Workshop on Experimental Algorithms*, pages 220–234, 2008.

[28] S. Busygin. A new trust region technique for the maximum clique problem, 2002. Internal report, http://www.busygin.dp.ua.

[29] Andrea Grosso, Marco Locatelli, and Federico Della Croce. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *J. Heuristics*, 10(2):135–152, 2004.

[30] Kengo Katayama, Akihiro Hamamoto, and Hiroyuki Narihisa. Solving the maximum clique problem by k-opt local search. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC2004)*, pages 1021–1025, 2004.

[31] Kengo Katayama, Masashi Sadamatsu, and Hiroyuki Narihisa. Iterated k-opt local search for the maximum clique problem. In *Proc. of EvoCOP-07*, pages 84–95, 2007.

[32] Christine Solnon and Serge Fenet. A study of aco capabilities for solving the maximum clique problem. *J. Heuristics*, 12(3):155–180, 2006.

[33] Wayne Pullan. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization*, 6:214–219, 2009.

[34] Shaowei Cai, Kaile Su, and Qingliang Chen. Ewls: A new local search for minimum vertex cover. In *Proc. of AAAI-10*, pages 45–50, 2010.

[35] *BHOSLIB Benchmark*. http://www.nlsde.uaa.edu.cn/~kexu/benchmarks/graphbenchmarks.htm.

[36] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artif. Intell.*, 175(9-10):1672–1696, 2011.

[37] *SAT Competition 2011*. http://satcompetition.org/2011/.

[38] David S. Johnson and Michael Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA, 1996.

[39] Roberto Battiti and Marco Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, 2001.

[40] Wayne Pullan. Phased local search for the maximum clique problem. *J. Comb. Optim.*, 12(3):303–323, 2006.

[41] J. Konc and D. Janezic. An improved branch and bound algorithm for the maximum clique problem. *Communications in Mathematical and in Computer Chemistry*, 58:569–590, 2007.

[42] Shaowei Cai, Kaile Su, and Abdul Sattar. Two new local search strategies for minimum vertex cover. In *Proc. of AAAI-12*, page to appear, 2012.

[43] Shaowei Cai and Kaile Su. Local search with configuration checking for SAT. In *Proc. of ICTAI-11*, pages 59–66, 2011.

[44] Chuan Luo, Kaile Su, and Shaowei Cai. Improving local search for random 3-SAT using quantitative configuration checking. In *Proc. of ECAI-12*, page to appear, 2012.

[45] Shaowei Cai and Kaile Su. Configuration checking with aspiration in local search for SAT. In *Proc. of AAAI-12*, page to appear, 2012.

[46] Shaowei Cai, Kaile Su, and Abdul Sattar. A new local search strategy for SAT. In *Workshop of CSPSAT-11*, 2011.

[47] H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, USA, 2005.

[48] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671– ‑ 680, 1983.

[49] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, NJ, USA, 1996.

[50] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *AAAI*, pages 440–446, 1992.

[51] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *AAAI*, pages 337–343, 1994.

[52] David A. McAllester, Bart Selman, and Henry A. Kautz. Evidence for invariants in local search. In *AAAI/IAAI*, pages 321–326, 1997.

[53] Holger H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proc. of AAAI-02*, pages 655–660, 2002.

[54] Chu Min Li and Wen Qi Huang. Diversification and determinism in local search for satisfiability. In *Proc. of SAT-05*, pages 158–172, 2005.

[55] John Thornton. Clause weighting local search for SAT. *J. Autom. Reasoning*, 35(1-3):97–142, 2005.

[56] Adrian Balint and Andreas Fröhlich. Improving stochastic local search for SAT with a new probability distribution. In *Proc. of SAT-10*, pages 10–15, 2010.

[57] Wil Michiels, Emile H. L. Aarts, and Jan H. M. Korst. *Theoretical aspects of local search*. Springer, 2007.

[58] *DIMACS Benchmark*. ftp://dimacs.rutgers.edu/pub/challeng/.

[59] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. A simple model to generate hard satisfiable instances. In *Proc. of IJCAI-05*, pages 337–342, 2005.

[60] Ke Xu and Wei Li. Many hard examples in exact phase transitions. *Theoretical Computer Science*, 355:291–302, 2006.

[61] Ke Xu and Wei Li. Exact phase transitions in random constraint satisfaction problems. *J. Artif. Intell. Res. (JAIR)*, 12:93–103, 2000.

[62] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artif. Intell.*, 171(8-9):514–534, 2007.

[63] Andrea Grosso, Marco Locatelli, and Wayne J. Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *J. Heuristics*, 14(6):587–612, 2008.

[64] *Papers on BHOSLIB*. http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/list-graph-papers.htm.

[65] Holger H. Hoos and Thomas Stützle. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artif. Intell.*, 112(1-2):213–232, 1999.

[66] K. Smyth and H. H. Hoos T. Stützle. Iterated robust tabu search for max-SAT. In *Proc. of Canadian Conference on AI-03*, pages 129–144, 2003.

[67] J. P. Watson and L. D. Whitley A. E. Howe. Linking search space structure, run-time dynamics, and problem difficulty: A step toward demystifying tabu search. *J. Artif. Intell. Res. (JAIR)*, 24:221–261, 2005.

[68] Dave A. D. Tompkins and Holger H. Hoos. Warped landscapes and random acts of SAT solving. In *Proc. of AMAI-04*, 2004.

[69] Paul Morris. The breakout method for escaping from local minima. In *Proc. of AAAI-93*, pages 40–45, 1993.

[70] Zhe Wu and Benjamin W. Wah. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proc. of AAAI/IAAI-00*, pages 310–315, 2000.

[71] Dale Schuurmans and Finnegan Southey. Local search characteristics of incomplete SAT procedures. *Artif. Intell.*, 132(2):121–150, 2001.

[72] Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. of CP-02*, pages 233–248, 2002.

[73] John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proc. of AAAI-04*, pages 191–196, 2004.

[74] Abdelraouf Ishtaiwi, John Thornton, Anbulagan, Abdul Sattar, and Duc Nghia Pham. Adaptive clause weight redistribution. In *Proc. of CP-06*, pages 229–243, 2006.

[75] *DIMACS Machine Benchmark*. ftp://dimacs.rutgers.edu/pub/dsj/clique/.

[76] Ian P. Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for SAT. In *AAAI*, pages 28–33, 1993.

[77] Fei Yan, Shaowei Cai, Ming Zhang, Guojun Liu, and Zhihong Deng. A clique-superposition model for social networks. *SCIENCE CHINA Information Sciences*, To be appeared, 2011.

[78] Fred Glover. Tabu search – part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[79] Fred Glover. Tabu search – part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[80] Bertrand Mazure, Lakhdar Sais, and Éric Grégoire. Tabu search for SAT. In *Proc. of AAAI-97*, pages 281–285, 1997.

[81] *The SAT Competition Homepage*. http://www.satcompetition.org.

[82] D.L. Berre and L.Simon. Fifty-five solvers in vancouver: the SAT 2004 competition. In *Proc. of SAT-04*, pages 321–344, 2004.

[83] Holger H. Hoos and Thomas Stützle. Local search algorithms for SAT: An empirical evaluation. *J. Autom. Reasoning*, 24(4):421–481, 2000.

[84] Wayne Pullan, Franco Mascia, and Mauro Brunato. Cooperating local search for the maximum clique problem. *J. Heuristics*, 17(2):181–199, 2011.

[85] Dale Schuurmans, Finnegan Southey, and Robert C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proc. of IJCAI-01*, pages 334–341, 2001.

[86] Abdelraouf Ishtaiwi, John Thornton, Abdul Sattar, and Duc Nghia Pham. Neighbourhood clause weight redistribution in local search for SAT. In *Proc. of CP-05*, pages 772–776, 2005.

个人简历、在学期间的研究成果

个人简历

1986年11月生；2004年9月考入华南理工大学计算机技术与工程学院计算机科学与技术专业，2008年7月本科毕业并获得工学学士学位；2008年9月保送到北京大学信息科学技术学院计算机软件与理论专业攻读博士学位至今。

发表/待发表论文

1. Shaowei Cai, Kaile Su: Configuration Checking with Aspiration in Local Search for SAT, to appear in Proc. of AAAI-12.

2. Shaowei Cai, Kaile Su, Abdul Sattar: Two New Local Search Strategies for Minimum Vertex Cover, to appear in Proc. of AAAI-12.

3. Chuan Luo, Kaile Su, Shaowei Cai: Improving Local Search for Random 3-SAT Using Quantitative Configuration Checking, to appear in Proc. of ECAI-12.

4. Fei Yan, Shaowei Cai, Ming Zhang, Guojun Liu, Zhihong Deng: A Clique-superposition Model for Social Networks, SCIENCE CHINA Information Sciences, DOI: 10.1007/s11432-011-4526-y, 2012.

5. Shaowei Cai, Kaile Su: Local Search with Configuration Checking for SAT, Proc. of ICTAI-11, pp. 59-66(2011).

6. Shaowei Cai, Kaile Su, Abdul Sattar: A New Local Search Strategy for SAT, CSPSAT-11 (Affiliated with the SAT-2011 conference), Ann Arbor, Michigan, USA, June 18, 2011.

7. Shaowei Cai, Kaile Su, Abdul Sattar: Local search with edge weighting and configuration checking heuristics for minimum vertex cover, Artif. Intell. 175(9-10), 1672 - 1696 (2011).

8. Shaowei Cai, Kaile Su, Qingliang Chen: EWLS: A New Local Search for Minimum Vertex Cover, Proc. of AAAI-10, pp. 45 - 50 (2010).

致　谢

论文完成之际，我最应该感激的是北京大学这所高等学府。北大古典优雅的校园不仅给我的生活增添了色彩，还赋予了我宁静的心境，伴随我渡过了很多科研的难关。北大浓厚的学术氛围，丰富的课堂讲座，诸多的良师益友，让我如同一颗水珠汇入了大海般自在而幸福。

首先，衷心感谢我的导师苏开乐教授。苏老师为我创造了良好的科研条件，在科研方向，技术改进，论文写作，口头报告等各个方面给予了我精心的指导，让我受益匪浅。苏老师教导我们，做研究不应以论文为目标，而是要做有重要意义的研究，要在本领域最重要的问题上做出世界最先进的研究。他还教导我们做研究要胆大心细，而写论文要客观准确，严肃认真，不得有半点含糊夸大。苏老师不仅带领我在学术上前进，在生活上也给予我无微不至的关怀，令我非常感动。他帮我解决了许多生活上的困难，让我在读博士的这几年可以安心地做研究。苏老师对学生非常亲切，在生活上我们讨论的话题非常广泛，让我感受到了科研之外的精神愉悦。和苏老师相处，如沐春风。苏老师对我做人做事做学问的影响非常深远，他是我一生的导师。

衷心感谢刘田老师和许可老师。刘老师有着踏实的工作作风和丰富的知识储备，对我提出的问题不论巨细都给出了详细的解答。刘老师对学生尽心尽责，不辞劳苦地组织算法讨论班，带领大家学习算法领域各种知识，促进了实验室的学术氛围。许老师有着令人钦佩的学术风范，他杰出的科研成果和低调的处世风格形成了鲜明的对比。许老师对科研有深刻的见解，在长远方向和具体工作上都给予了我很好的指导。他教我明白科研成果的意义不在于技巧的高深，而在于其对世界产生的正面影响力。感谢张健老师对我关于算法应用和软件开源方面的建议。感谢屈婉玲老师特别细心地通读了本论文并指出了论文中的一些笔误。

感谢陪伴我渡过这段难忘岁月的兄弟姐妹们，和大家一起成长的日子是那么快乐。感谢我的室友吕雁飞在科研上和生活上对我的支持和帮助，在我迷茫无助时鼓励我，在我取得成果时为我高兴。感谢我的两位师弟揭忠和罗川。揭忠师弟在我读博士期间非常耐心地帮我做了很多辅助工作，加快了我的科研进度；罗川师弟虽然是最后一年才到北大，不过我们的讨论和合作是愉快的。我们“算法三人组”在科研上是最佳搭档，我们一起攻克难题，并肩作战；在生

活上我们亲如兄弟，不管是调侃娱乐，还是庆祝胜利，都给我留下了许多快乐的回忆。感谢我的师妹刘婵娟，一直默默承担我们小组秘书长兼业务顾问的角色，使大家的科研生活更加顺利。感谢苏飞师兄给我提供各种课程上的材料和有益的讨论。感谢我的论文合作者燕飞，不仅在科研上互相讨论，也带领我体验了各种生活乐趣。感谢孙韬对我在美国期间的照顾，我们的互相交流丰富了我的课余生活。感谢澳大利亚Griffith大学的刘先同学为我提供澳洲签证和生活的各种资讯。感谢实验室和博士班的一众同门，和你们的交流讨论让我获益良多。

感谢Chumin Li，Adrian Balint 和Oliver Gableske，他们和我有着共同的研究兴趣。感谢他们和我分享他们的思想，论文和最新进展。和他们的学术讨论加深了我对相关问题和算法的理解，给我的研究带来了启发。他们是我的良师益友。

我的父母含辛茹苦地养育我，顶着各种巨大的压力培养我读大学读博士，父母这种伟大无私的爱已经不能用感谢来表达了。我的家人多年来给予我的支持和鼓励，让我一直感受到家的温暖，没有你们也就不会有我的今天。亲情一直是我前进的动力。最后，我要向我美丽大方、温柔体贴的女朋友表示由衷的感谢。感谢她在我求学期间一直陪伴我鼓励我，让我的生活充满温馨。感谢她对我忙于科研的理解，几年如一日地替我分担生活上的各种事情，让我可以专心地从事研究工作。

谨以此文献给所有关心和帮助过我的人！

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　月　日

学位论文使用授权说明

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

按照学校要求提交学位论文的印刷本和电子版本；

学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务；

学校可以采用影印、缩印、数字化或其它复制手段保存论文；

在不以赢利为目的的前提下，学校可以公布论文的部分或全部内容。

（保密的论文在解密后应遵守此规定）

论文作者签名：　　导师签名：

日期：　　年　月　日