2004

Dueling CSP representations: Local search in the primal versus dual constraint graph.

Mingyan Huang
University of Windsor

Dueling CSP Representations: Local Search in the Primal versus Dual Constraint Graph

by

Mingyan Huang

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2004

Your file Votre référence
ISBN: 0-612-92447-5
Our file Notre référence
ISBN: 0-612-92447-5

Abstract

Constraint Satisfaction Problems (CSPs) can be used to represent and solve many problems in Artificial Intelligence and the real world. When solving Constraint Satisfaction Problems, many of the methods developed and studied have focused only on the solution of binary CSPs while a large portion of real life problems are naturally modeled as non-binary CSPs. In this thesis we have designed an empirical study to investigate the behaviour of several local search methods in primal and dual constraint graph representations when solving non-binary CSPs. Local search methods tend to find a solution quickly since they generally give up the guarantee of completeness for polynomial time performance. Such local search methods include simple hill-climbing, steepest ascent hill-climbing and min-conflicts heuristics hill-climbing. We evaluate the performance of these three algorithms in each representation for a variety of parameter settings and we compare the search time cost means of two groups to support the comparison.

Our comparison shows that we can use local search to solve a CSP with tight constraints in its dual representation and gain a better performance than using it in its primal representation. When constraints are getting looser, using local search in primal representation is a better choice. Among the three local search methods used in our empirical study, min-conflicts heuristics hill-climbing always gain the best performance while steepest ascent hill-climbing tends to have the worst performance and simple hill climbing is in the middle or sometimes it is the best.

Acknowledgements

The first person I would like to thank is my supervisor Dr. Scott Goodwin, an energetic and hardworking professor with sense of humor. He not only spent countless hours of discussion with me and provided immediate comments, but also gave me guidelines for any of my ideas, even the most immature ones.

I also extend my appreciation to the members of my committee – Dr. S. Ejaz Ahmed and Dr. J. Morrissey, not only for their spending precious time to read through my drafts, but also for their advice and guidance in writing this thesis.

Thanks are also due to Dr. M. Hlynka from Department of Mathematics and Statistics, University of Windsor. He gave me a valuable tutorial in specific field of statistical analysis.

I would also like to thank my fellow students Lucy, Eric and Bob for their help and useful comments during my research. I am grateful for the support and encouragement from my family during my graduate study period in University of Windsor.

Table of Contents

Abstract iii

Acknowledgements iv

List of Figures viii

List of Tables xi

Chapter 1 Introduction 1

1.1 Statement of the Problem 1

1.2 Motivation 3

1.3 Outline 4

Chapter 2 Background 5

2.1 Constraint Satisfaction Problems (CSPs) 5

2.2 Binary CSPs and Non-Binary CSPs 9

2.3 Transform Non-Binary CSP into Binary CSP 11

2.3.1 Dual Graph Method 12

2.3.2 Hidden Variable Method 15

2.4 Search 18

2.4.1 Systematic Search 18

2.4.2 Local Search 23

2.5 Conclusions 26

Chapter 3 Local Search in Primal and Dual Constraint Graphs 27

3.1 Local Search Algorithms 27

3.1.1 General Local Search Strategy 27

3.1.2 Simple Hill-climbing Algorithm 28

3.1.3 Simple Hill-climbing Flowchart 29

3.1.4 Simple Hill-climbing Example 31
3.1.5 Steepest Ascent Hill-climbing 31
3.1.6 Steepest Ascent Hill-climbing Flowchart 33
3.1.7 Steepest Ascent Hill-climbing Example 33
3.1.8 Min-conflicts Heuristics Hill-climbing 36
3.1.9 Min-conflicts Heuristics Flowchart 36
3.1.10 Min-conflicts Heuristics Example 37
3.2 Local Search in Primal and Dual Constraint Graphs 40
3.3 Empirical Study Design 42
3.3.1 Empirical Study Input Design 42
3.3.2 Empirical Study Output Design 44
3.3.3 Empirical Study Comparisons 46
3.4 Conclusions 49
Chapter 4 Experiment Result and Analysis 50
4.1 Experiment Results and Analysis on Class I 52
4.1.1 Simple Hill-climbing on Class I 52
4.1.2 Steepest Ascent Hill-climbing on Class I 55
4.1.3 Min-conflicts Heuristics Hill-climbing on Class I 57
4.1.4 Comparisons among Different Hill-climbing algorithms on Class I 59
4.2 Experiment Results and Analysis on Class II 61
4.2.1 Simple Hill-climbing on Class II 61
4.2.2 Steepest Ascent Hill-climbing on Class II 63
4.2.3 Min-conflicts Heuristics Hill-climbing on Class II 65
4.2.4 Comparisons among Different Hill-climbing algorithms on Class II ... 67
4.3 Experiment Results and Analysis on Class III 69
4.3.1 Simple Hill-climbing on Class III 69
4.3.2 Steepest Ascent Hill-climbing on Class III 72
4.3.3 Min-conflicts Heuristics Hill-climbing on Class III 74
4.3.4 Comparisons among Different Hill-climbing algorithms on Class III . 76
4.4 Conclusions 77

Chapter 5 Conclusions **79**

5.1 Future Work 80

Appendix A T-test **81**

Bibliography **87**

Vita Auctoris **101**

List of Figures

Figure 2.1 Map coloring problem – a binary CSP 7

Figure 2.2 An example of non-binary CSPs and its hypergraph 10

Figure 2.3.1 Primal and dual representations of a non-binary CSP P1 15

Figure 2.3.2 An example of hidden variable representation for a non-binary CSP 17

Figure 2.4.1.1 Control of backtracking algorithm 20

Figure 2.4.1.2 Backtracking algorithm (BT) 21

Figure 2.4 Possible problems with Hill-climbing algorithms 25

Figure 3.1.1 General local search algorithm 28

Figure 3.1.2 Simple Hill-climbing algorithm 29

Figure 3.1.3 Simple Hill-climbing flowchart 30

Figure 3.1.4.1 An example of a binary CSP 31

Figure 3.1.4.2 Search tree of Simple Hill-climbing in primal representation 32

Figure 3.1.5 Steepest Ascent Hill-climbing algorithm 33

Figure 3.1.6 Steepest Ascent Hill-climbing flowchart 34

Figure 3.1.7 Search tree of Steepest Ascent Hill-climbing in primal representation 35

Figure 3.1.8 Min-conflicts Heuristics Hill-climbing algorithm 37

Figure 3.1.9 Min-conflicts Heuristics Hill-climbing flowchart 38

Figure 3.1.10.1 Search tree of Min-conflicts Heuristics Hill-climbing in primal representation 39

Figure 3.1.10.2 Different search tree of Min-conflicts Heuristics Hill-climbing in the same primal representation 40

Figure 3.3.3.1 Move cost of Steepest Ascent Hill-climbing 47

Figure 3.3.3.2 Move cost ratio for local search 48

Figure 4.1.1.1 Comparison of Tm Mean of Simple Hill-climbing on Class I 54
Figure 4.1.2.1 Comparison of Tm Mean of Steepest Ascent Hill-climbing on Class I 56
Figure 4.1.3.1 Comparison of Tm Mean of Min-conflicts Heuristics Hill-climbing on Class I 58
Figure 4.1.4.1 Comparisons of Tm Means for LPsim, LPstp, LPmc, LDsim, LDstp and LDmc on Class I 59
Figure 4.1.4.2 Comparisons of Nc for LPsim, LPstp and LPmc on Class I 60
Figure 4.1.4.3 Comparisons of Nc for LDsim, LDstp and LDmc on Class I 60
Figure 4.2.1.1 Comparison of Tm Mean of Simple Hill-climbing on Class II 62
Figure 4.2.2.1 Comparison of Tm Mean of Steepest Ascent Hill-climbing on Class II 64
Figure 4.2.3.1 Comparison of Tm Mean of Min-conflicts Heuristics Hill-climbing on Class II 66
Figure 4.2.4.1 Comparisons of Tm Means for LPsim, LPstp, LPmc, LDsim, LDstp and LDmc on Class II 68
Figure 4.2.4.2 Comparisons of Nc for LPsim, LPstp and LPmc on Class II 68
Figure 4.2.4.3 Comparisons of Nc for LDsim, LDstp and LDmc on Class II 69
Figure 4.3.1.1 Comparison of Tm Mean of Simple Hill-climbing on Class III 71
Figure 4.3.2.1 Comparison of Tm Mean of Steepest Ascent Hill-climbing on Class III 73
Figure 4.3.3.1 Comparison of Tm Mean of Min-conflicts Heuristics Hill-climbing on Class III 75
Figure 4.3.4.1 Comparisons of Tm Means for LPsim, LPstp, LPmc, LDsim, LDstp and LDmc on Class III 76
Figure 4.3.4.2 Comparisons of Nc for LPsim, LPstp and LPmc on Class III 77
Figure 4.3.4.3 Comparisons of Nc for LDsim, LDstp and LDmc on Class III 77
Figure Appendix_1 T-test formula when variances are unequal 83
Figure Appendix_2 Formula for the standard error of the difference between the means when variances are unequal 83

Figure Appendix_3 Formula of confidence interval at a 95% confidence level..... 86

List of Tables

Table 4.1.1.1 Time cost of Simple Hill-climbing on Class I 53

Table 4.1.1.2 Rn, Nd and Cc of Simple Hill-climbing on Class I 54

Table 4.1.2.1 Time cost of Steepest Ascent Hill-climbing on Class I 55

Table 4.1.2.2 Rn, Nd and Cc of Steepest Ascent Hill-climbing on Class I 55

Table 4.1.3.1 Time cost of Min-conflicts Heuristics Hill-climbing on Class I 57

Table 4.1.3.2 Rn, Nd and Cc of Min-conflicts Heuristics Hill-climbing on Class I 58

Table 4.2.1.1 Time cost of Simple Hill-climbing on Class II 61

Table 4.2.1.2 Rn, Nd and Cc of Simple Hill-climbing on Class II 62

Table 4.2.2.1 Time cost of Steepest Ascent Hill-climbing on Class II 64

Table 4.2.2.2 Rn, Nd and Cc of Steepest Ascent Hill-climbing on Class II 65

Table 4.2.3.1 Time cost of Min-conflicts Heuristics Hill-climbing on Class II 66

Table 4.2.3.2 Rn, Nd and Cc of Min-conflicts Heuristics Hill-climbing on Class II 67

Table 4.3.1.1 Time cost of Simple Hill-climbing on Class III 70

Table 4.3.1.2 Rn, Nd and Cc of Simple Hill-climbing on Class III 71

Table 4.3.2.1 Time cost of Steepest Ascent Hill-climbing on Class III 72

Table 4.3.2.2 Rn, Nd and Cc of Steepest Ascent Hill-climbing on Class III 73

Table 4.3.3.1 Time cost of Min-conflicts Heuristics Hill-climbing on Class III 74

Table 4.3.3.2 Rn, Nd and Cc of Min-conflicts Heuristics Hill-climbing on Class III 75

Chapter 1

Introduction

Many problems in Artificial Intelligence (AI) and other areas of computer science can be viewed as special cases of Constraint Satisfaction Problems (CSPs) [Nad90]. CSPs are worth studying in isolation because they are general problems which have unique features that can be exploited to arrive at solutions [Tsa93]. These unique features make CSPs one of the most powerful mechanisms for representing complex relationships in real life problems and AI problems such as computer vision, temporal reasoning and resource allocation in solving AI planning and scheduling problems.

1.1 Statement of the Problem

Basically, a CSP is a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values that the variables can simultaneously take [Tsa93]. There are three factors in a constraint satisfaction problem: variables, a domain for each variable and constraints among these variables. The goal is to find one assignment, all assignments, or the best assignment of values to the variables from their associated domains such that the assignment satisfies all the constraints. Finding the best assignment falls into another category which is called constraint optimization problem (COP) and it is not discussed in this thesis.

Solutions for CSPs can be found by systematic search methods or by local search methods which use randomness to aid in the search [Nag01]. Systematic methods generally search the space of partial solutions by generating consistent assignments to variables with values from their domains and then extending these partial solutions to full solutions one variable at a time. Systematic methods such as chronological backtracking and forward checking are complete search methods which can find all solutions. Local search methods investigate the space of all complete assignments of values to variables for consistent assignments. Local search methods are generally incomplete search methods which aim to find one solution, but may fail to find any solution even if one exists.

A constraint satisfaction problem can be represented as a constraint graph. Algorithms for solving CSPs exploit the search space according to the structure of the constraint graph. Generally, there are two ways of presenting CSPs in a constraint graph. One is the primal constraint graph and the other is the dual constraint graph. A primal constraint graph directly reflects the original constraint satisfaction problem framework while a dual constraint graph is a structural transformation of the primal representation of the given CSP. The dual constraint graph is an equivalent representation of the primal constraint graph where the primal constraints are the dual variables, and the dual constraints are compatibility constraints on the primal variables shared between the primal constraints.

CSPs can be binary or non-binary. A binary CSP is a CSP with unary and binary constraints only. A non-binary CSP is a CSP with constraints not limited to unary and binary constraints. A non-binary constraint involves at least three variables. Local search methods have been used frequently to solve binary CSPs represented as primal constraint graphs. It is possible to use local search to solve binary and non-binary CSPs in their dual representation. In this thesis we are going to solve binary and non-binary CSPs represented as a primal constraint graph or dual constraint graph by using different local search methods and compare their performance. The question we

are interested in is whether and under what circumstances one representation may be preferred to the other.

1.2 Motivation

When solving Constraint Satisfaction Problems, many of the methods developed and studied have focussed only on the resolution of binary CSPs which are limited to constraints involving at most two variables. The justification for this is the fact that any non-binary CSP can be translated into an equivalent binary CSP. [RPD90]. Although binary representation and non-binary representation are equivalent terms of solutions, the latter specifies the CSP in a more natural way. As well, the non-binary CSP constraint graph may contain structural information that can be exploited to make the search process more efficient. With the help of dual constraint graph a lot of existing binary constraint satisfaction algorithms can directly handle non-binary CSPs since the dual representation has a binary structure.

Many real life problems require a solution (not all solutions) to be found quickly. In many situations, a timely response by a CSP solver is crucial. A CSP solver may spend days or years solving some special kinds of CSPs on conventional hardware by using systematic search methods such as backtracking and forwardchecking [Tsa93]. For example, in scheduling transportation airplanes, in a freight airport terminal, one may be allowed very limited time to schedule a lot of airplanes and delays could lead to extremely high cost. In the Hubble Space Telescope scheduling problem [MPJL93], ten of thousands of astronomical observations per year must be scheduled and thus a timely response by a scheduling system is required. In some applications such as allocating resources to emergency rescue teams, solutions should be found in a limited time, otherwise they are useless if they are found too late [Tsa93]. In these cases local search could be useful. Local search methods generally give up the guarantee of completeness for polynomial time performance [Nag01].

1.3 Outline

The remainder of this thesis is organized as follows: Chapter 2 gives some background related to CSP structure and techniques for solving CSPs. Chapter 3 discusses the use of local search methods to solve a CSP in its dual representation versus in its primal representation and give the empirical study design structure. Chapter 4 gives the experiment results according to the approach presented in Chapter 3. Chapter 5 is the conclusion.

Chapter 2

Background

CSPs can be used to represent and solve many problems in AI and the real world. Constraint satisfaction is a term which covers a wide range of methods to solve these problems stated in the form of a set of constraints. In this chapter we will introduce related CSP definitions and search methods.

2.1 Constraint Satisfaction Problems (CSPs)

A constraint satisfaction problem gives a model which describes some requirements for a finite number of variables by using constraints. The set of possible values which is called the domain for each variable is finite. Here we give a formal definition of CSP.

Definition 2.1 A **constraint satisfaction problem (CSP)** is a tuple $P(V,D,C)$ whose components are defined below:

- $V = \{v_1, \ldots, v_n\}$ is a finite set of n variables. In this thesis we also use uppercase V_i to represent a certain subset of V which contains variables $v_{i1}, v_{i2}, \ldots, v_{ik}$.
- $D = \{D_1, \ldots, D_n\}$ is a set of domains. Each variable $v_i \in V$ has a corresponding finite domain of possible values, D_i. We also use $D(v_i)$ to represent the domain of variable v_i.

- $C = \{C_1, \ldots, C_m\}$ is a set of m constraints. A constraint tells which value-combined tuples are allowed for a certain subset V_i of all the variables. A constraint $C_i = < V_i, S_i >$ on an ordered set of variables $V_i = \{v_{i1}, v_{i2}, \ldots, v_{ik}\} \subseteq V$, is defined as a relation on these variables, $S_i \subseteq D(v_{i1}) \times \ldots \times D(v_{ik})$. This relation stands for the set of allowable combined values for the variables in V_i. In this thesis we also use the notation $C_{1,2}$ to represent C_i with $V_i = \{v_1, v_2\}$ and $S_{1,2}$ to represent the combined-value tuples allowed in C_i.

Now we give the following definitions by the above CSP $P(V,D,C)$.

Definition 2.2 The number of variables involved in a constraint is known as the **arity** of the constraint. A **unary constraint** only involves one variable; a **binary constraint** has two variables involved; a non-binary constraint has arity greater than two. The **problem arity** of a CSP is defined as the maximum constraint arity in this CSP.

Definition 2.3 Given a set of variables $V_i = \{v_{i1}, v_{i2}, \ldots, v_{ik}\}$, a value assignment from domain $D(v_i)$ to variable v_i, for each variable in this set, is called an **instantiation**. For example, $< v_1, 1 >$ is an instantiation for variable v_1. A **solution** is an assignment of values to all the variables, so that each variable in P is assigned a value from its domain, and all the constraints in P are satisfied simultaneously. A CSP is solvable if it has at least one solution, otherwise it is unsolvable or over constrained.

Definition 2.4 A CSP which only contains unary and binary constraints is called a **binary CSP**. A CSP which has one or more non-binary constraints is called a **non-binary CSP**.

Constraint satisfaction problems can be characterized by **their tightness**, which could be measured under the following definitions.

Definition 2.5 **The tightness of a constraint** $T(C_i)$ is measured by the number of tuples that satisfy the constraint over all possible combined-value tuples in C_i. $T(C_i) = S / T$ where S is the number of S_i (combined-value tuples allowed in C_i) and T is the number of all possible combined-value tuples in C_i according to the domains of each variable involved in C_i.

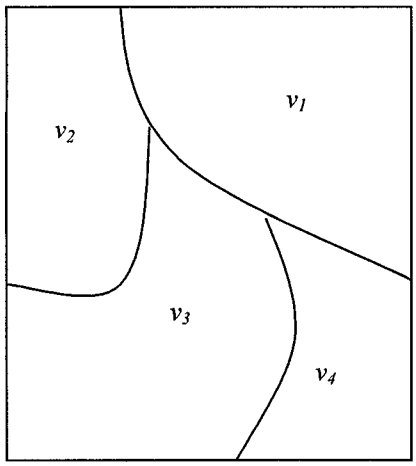
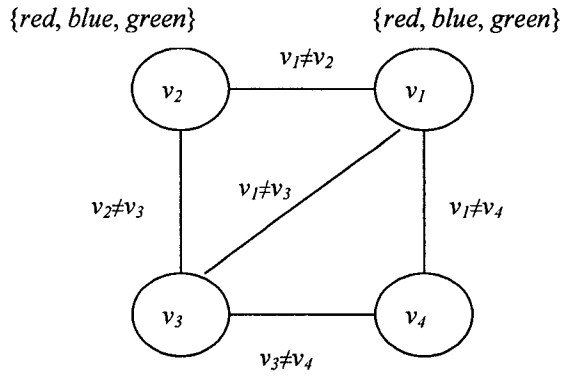


Figure 2.1(a) map to be colored



Variables: v_1, v_2, v_3, v_4

Domains:

Domain of v_1 $D(v_1)$: {*red, blue, green*}
Domain of v_2 $D(v_2)$: {*red, blue, green*}
Domain of v_3 $D(v_3)$: {*red, blue, green*}
Domain of v_4 $D(v_4)$: {*red, blue, green*}

Constraints:

C_1: $v_1 \neq v_2$ C_2: $v_1 \neq v_3$ C_3: $v_1 \neq v_4$
C_4: $v_2 \neq v_3$ C_5: $v_3 \neq v_4$

Figure 2.1(b) a constraint graph of the CSP in 2.1(a)

Figure 2.1 Map coloring problem - a binary CSP

Definition 2.6 **The tightness of a CSP** $T(P)$ is measured by the number of solution tuples over the number of all distinct combined-value tuples over all variables in P.

Tightness is a relative measure. Some CSPs solving techniques are more suitable for tighter problems, while others are suitable for looser problems.

Here we use the map coloring problem [Kum92] to explain concepts for CSP. In the map coloring problem in Figure 2.1(a), we need to assign a color to each area of the map from a set of colors such that no two adjacent areas have the same color. Figure 2.1(a) shows an example of a map to be colored. The map has four areas which are to be colored red, blue or green. Figure 2.1(b) is the CSP model which describes the problem. In the map coloring problem, each area is a variable and the domain of each variable is the given set of colors. For each pair of areas that are adjacent on the map, there is a constraint between the corresponding variables which disallows the same value to be assigned to these two variables. For this map coloring problem in Figure 2.1, there are four variables $\{v_1, v_2, v_3, v_4\}$ and each variable has the same domain {red, blue, green}. There are five constraints $\{C_1: v_1 \neq v_2, C_2: v_1 \neq v_3, C_3: v_1 \neq v_4, C_4: v_2 \neq v_3, C_5: v_3 \neq v_4\}$. The number of satisfied tuples of C_1 is 6 and these tuples are {(*red, blue*), (*red, green*), (*blue, red*), (*blue, green*), (*green, red*), (*green, blue*)}. The number of all possible combined-value tuples for C_1 is $3 \times 3 = 9$. Therefore the tightness of constraint C_1 is $T(C_1) = 6/9$. There are total 6 solution tuples for this map coloring problem which are {(*red, blue, green, blue*), (*red, green, blue, green*), (*blue, red, green, red*), (*blue, green, red, green*), (*green, red, blue, red*), (*green, blue, red, blue*)}. The number of all distinct combined-value tuples over all 4 variables is $3 \times 3 \times 3 \times 3 = 81$. Thus we get the tightness of this map coloring problem $T(P) = 6/81$.

A constraint in a CSP can be given either explicitly, by enumerating the tuples allowed, or implicitly, e.g., by an algebraic expression. When constraints are given explicitly, they are known as **extensional constraints**, and when constraints are given implicitly, they are known as **intensional constraints**. In Figure 2.1(b) the constraints are given in an intensional form. We can also enumerate constraint $C_1: v_1 \neq v_2$ in its extensional form as $S_{1,2}$ = {(*red, blue*), (*red, green*), (*blue, red*), (*blue, green*), (*green, red*), (*green, blue*)}.

2.2 Binary CSPs and Non-Binary CSPs

Constraint Satisfaction Problems can be divided into binary CSPs and non-binary CSPs which are also called **general CSPs**. A binary CSP is a CSP with unary and binary constraints only, which means, each constraint of this CSP is either a constraint which restricts a single variable or a constraint between two variables. The map coloring problem in Figure 2.1 is a binary CSP since each constraint is only between two variables. In Figure 2.1b we can see that only two variables are involved in each of the five constraints C_1, C_2, C_3, C_4 and C_5, which means that the adjacent areas in that map can not take the same color. A CSP with constraints not limited to unary and binary will be referred to as a non-binary CSP.

Before giving an example of non-binary CSPs, we now present some definitions from graph theory in [Nag01].

Definition 2.7 A **graph** G is a structure <V, E>, where $V = \{v_1, v_2, \ldots, v_n\}$ is a finite set of elements called **vertices** (also referred to as **nodes**), and $E = \{e_1, e_2, \ldots, e_n\}$, is a finite set of elements of called **edges**, such that every element of E is a pair of distinct elements from V. V is called the **vertex set** of G, while E is called in the **edge set**. An edge in a graph can only connect two nodes.

Definition 2.8 The edges of a graph may be assigned specific values or labels, in which case the graph is called a **labelled graph**.

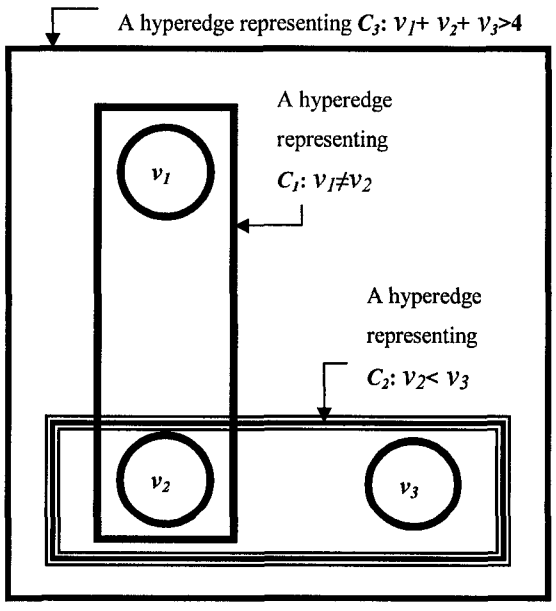
Definition 2.9 A binary CSP can be associated with a **constraint graph** G. N(G), which is the set of nodes(vertices) in G, corresponds to the set of variables and E(G), the set of edges in G, corresponds to the set of binary constraints [Mac77].

An edge in a constraint graph only connects two nodes since a binary constraint only involves two variables. For example, Figure 2.1b is a constraint graph for the binary

CSP in Figure 2.1a. In Figure 2.1b, the set of nodes N(G) ,which includes v_1, v_2, v_3 and v_4, corresponds to the set of variables in the map coloring problem. The set of edges E(G), which includes C_1, C_2, C_3, C_4 and C_5, corresponds to the set of binary constraints in the map coloring problem. Constraint graphs are also referred to as **constraint networks**.

Definition 2.10 A **hypergraph** is a generalisation of a graph where the set of edges is replaced by a set of **hyperedges**. A hyperedge extends the notion of an edge by allowing more than two nodes to be connected by a hyperedge. A hypergraph is a structure <V,E^h>, where V is a set of nodes and E^h is a set of hyperedges, with each hyperedge is a subset of the node set V.

In a constraint graph, an edge is only allowed to connect two nodes. This representation is good for binary CSPs, but is limited when representing non-binary CSPs. Thus we use hypergraph for non-binary CSPs.



Variables: v_1, v_2, v_3

Domains:
Domain of v_1 $D(v_1)$: {1, 2, 3}
Domain of v_2 $D(v_2)$: {1, 2, 3}
Domain of v_3 $D(v_3)$: {1, 2, 3}

Constraints:
C_1: v_1≠v_2
C_2: v_2< v_3
C_3: v_1+ v_2+ v_3>4

Figure 2.2 An example of non-binary CSPs and its hypergraph

Figure 2.2 is an example of non-binary CSPs. Here we can see that the graph is a hypergraph since there is one hyperedge connects three nodes v_1, v_2 and v_3. This hyperedge represents constraint C_3: $v_1 + v_2 + v_3 > 4$, which involves three variables v_1, v_2 and v_3.

2.3 Transform Non-Binary CSP into Binary CSP

In the early research of constraint satisfaction problems, many of the methods developed and studied focussed only on solving binary CSPs which are limited to constraints involving at most two variables. The justification for this has been the fact that the non-binary and binary representations are equivalent in terms of solutions [RPD90]. But many real life problems contain non-binary constraints and the most natural way to model such real life problems is to construct non-binary CSPs. For example a non-binary constraint which specifies that a set of n variables needs to be assigned different values (called an ***all_different*** constraint [Nag01]) can also be specified by a set of binary constraints which restricts any two variables in the variable set can only be assigned different values from their domains. Although these two formulations are equivalent in terms of the solutions that they admit, the former is clearly the one that specifies the requirement in a more natural way. As well, it may be more efficient to solve a non-binary CSP directly.

From the above we can find that there are two good reasons for looking carefully at the issue of translating non-binary CSPs into binary CSPs. First, non-binary CSPs appear quite frequently when modeling real life problems. The second reason is that, as noted above, a common justification for focusing solely on binary CSPs is the fact that a non-binary CSP can be translated into an equivalent binary representation.

There are two well known modeling techniques which can be used to transform a general (non-binary) CSP model into an equivalent binary CSP: **the dual graph method** and **the hidden variable method**.

2.3.1 Dual Graph Method

A hypergraph for a non-binary CSP is also called a **primal representation**, or **primal constraint graph**, since it directly represents this non-binary CSP. In [DP89] Dechter and Pearl introduced the **dual representation** to CSP researchers which originally comes from the relational database community. They propose the transformation of any non-binary CSP into its dual representation. The main idea of transforming a non-binary CSP into its dual representation is to construct a new CSP where constraints in the original non-binary CSP are now variables with structured domains and variables in the original non-binary CSP are now the constraints. The dual graph method for transforming a non-binary CSP into a binary CSP is also known as **dual encoding**.

Definition 2.11 Given a CSP, the **dual constraint graph** associated with it is a labelled graph, where $N=C$. For every pair of constraints C_i, $C_j \in C$, such that $V_i \cap V_j \neq \emptyset$, there is an edge in the dual constraint graph, connecting nodes C_i and C_j. A dual constraint graph is the dual representation of a CSP.

The following example illustrates how dual graph method converts a non-binary CSP into a binary CSP. First consider the following non-binary CSP *P1*:

Variables: v_1, v_2, v_3, v_4
Domains:
Domain of v_1: $D(v_1) = \{1, 2\}$
Domain of v_2: $D(v_2) = \{0, 1\}$
Domain of v_3: $D(v_3) = \{1, 2, 3\}$
Domain of v_4: $D(v_4) = \{1, 2, 3\}$
Constraints:
$C_{1,2,3}$: $v_1 + v_2 < v_3$

$C_{1,4}$: $v_1 < v_4$

$C_{2,3}$: $v_2 \neq v_3$

For any CSP model there are three factors: variables, domain for each variable and constraints among these variables. According to these three factors there are three main steps in the dual encoding for constructing the dual graph for a non-binary CSP from its primal representation. Here we use *P1* to represent the original non-binary CSP and use *P2* to represent newly constructed binary CSP:

a) Construct the variables:

For each primal constraint in the original CSP *P1* we construct a corresponding **dual variable** in *P2*. Thus the constraints in the primal representation become the variables in the dual representation.

1) For constraint $C_{1,2,3}$ in *P1*: changed to the dual node $C_{1,2,3}$;

2) For constraint $C_{1,4}$ in *P1*: changed to the dual node $C_{1,4}$;

3) For constraint $C_{2,3}$ in *P1*: changed to the dual node $C_{2,3}$.

There are three dual variables in *P2* (also called **dual nodes** in the dual graph), which correspond to the three constraints in *P1*.

b) Construct the domains:

Since every dual variable is a constraint in the original CSP, the domain of each dual variable is the set of tuples that satisfy the constraint. The following table illustrates how to get the **dual domain** for a dual node:

Dual Node	Corresponding constraint in *P1*	Related domains in *P1*	Tuples satisfying the constraint	Dual Node's domain in *P2*
$C_{1,2,3}$	$C_{1,2,3}$: $v_1 + v_2 < v_3$	$D(v_1) = \{1, 2\}$ $D(v_2) = \{0, 1\}$ $D(v_3) = \{1, 2, 3\}$	1. $v_1=1, v_2=0, v_3=2$ 2. $v_1=1, v_2=0, v_3=3$ 3. $v_1=1, v_2=1, v_3=3$ 4. $v_1=2, v_2=0, v_3=3$	{(1, 0, 2), (1, 0, 3), (1, 1, 3), (2, 0, 3)}
$C_{1,4}$	$C_{1,4}$: $v_1 < v_4$	$D(v_1) = \{1, 2\}$ $D(v_4) = \{1, 2, 3\}$	1. $v_1=1, v_4=2$ 2. $v_1=1, v_4=3$ 3. $v_1=2, v_4=3$	{(1, 2), (1, 3), (2, 3)}
$C_{2,3}$	$C_{2,3}$: $v_2 \neq v_3$	$D(v_2) = \{0, 1\}$ $D(v_3) = \{1, 2, 3\}$	1. $v_2=0, v_3=1$ 2. $v_2=0, v_3=2$ 3. $v_2=0, v_3=3$ 4. $v_2=1, v_3=2$ 5. $v_2=1, v_3=3$	{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3)}

c) Construct the constraints:

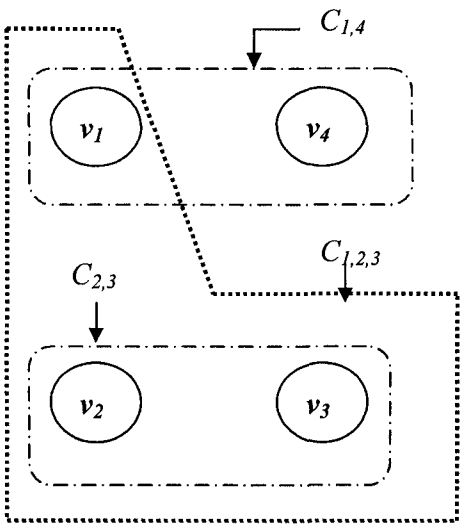
Check all the constraints in the original CSP *P1*. If two constraints in the original CSP share any variables, then there is an edge connecting the two nodes in the new binary representation *P2*. This constraint is a compatibility binary constraint which restricts same values should be assigned to the shared variable between the two dual nodes. The follow table shows how to get the new **dual constraints** in *P2*:

Dual Nodes	Corresponding constraint in *P1*	Shared variables	New constraints in *P2*
$C_{1,2,3}$ $C_{1,4}$	$C_{1,2,3}$: $v_1 + v_2 < v_3$ $C_{1,4}$: $v_1 < v_4$	v_1	1. v_1 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$ and $D(C_{1,4})$
$C_{1,2,3}$ $C_{2,3}$	$C_{1,2,3}$: $v_1 + v_2 < v_3$ $C_{2,3}$: $v_2 \neq v_3$	v_2	2. V_2 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$ and $D(C_{2,3})$
		v_3	3. V_3 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$and $D(C_{2,3})$
$C_{1,4}$ $C_{2,3}$	$C_{1,4}$: $v_1 < v_4$ $C_{2,3}$: $v_2 \neq v_3$		

Following the above three steps now we get the new binary CSP *P2*, which is transformed from the non-binary CSP by the dual graph method. The new CSP *P2* is a binary CSP since each of its dual constraints only involves two dual variables. Figure 2.3.1 is the primal representation and the dual representation of the non-binary CSP *P1*.

As CSP constraints can be represented either intensionally or extensionally, in primal graph either representation is allowed. But for the dual graph method that converts the primal constraints into dual variables, the dual domains need to be stored explicitly. In the above example the dual graph method gets the dual domains which are enumerated as tuples while the primal constraints are given implicitly. When modeling many real life problems the primal constraints are frequently expressed

extensionally. In that way the dual graph method can get the dual domains without a conversion.



Variables: $\{ v_1, v_2, v_3, v_4\}$

Constraints:

$C_{1,2,3}$: $v_1 + v_2 < v_3$

$C_{1,4}$: $v_1 < v_4$

$C_{2,3}$: $v_2 \neq v_3$

Domains:

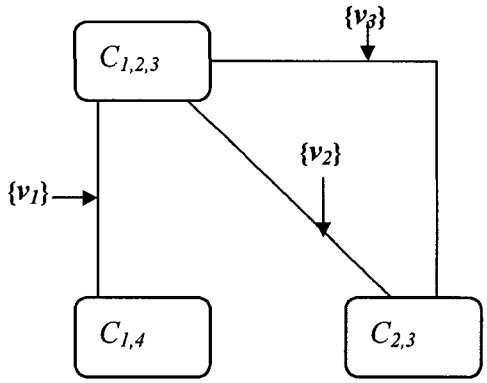
$D(v_1) = \{1, 2\}$

$D(v_2) = \{0, 1\}$

$D(v_3) = \{1, 2, 3\}$

$D(v_4) = \{1, 2, 3\}$

Primal Representation of *P1*



Dual Variables: $\{ C_{1,2,3}, C_{1,4}, C_{2,3}\}$

Dual Domains:

$D(C_{1,2,3}) = \{(1, 0, 2), (1, 0, 3), (1, 1, 3), (2, 0, 3)\}$

$D(C_{1,4}) = \{(1, 2), (1, 3), (2, 3)\}$

$D(C_{2,3}) = \{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3)\}$

Dual Constraints:

1. v_1 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$ and $D(C_{1,4})$
2. v_2 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$ and $D(C_{2,3})$
3. v_3 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$ and $D(C_{2,3})$

Dual Representation of *P1*

Figure 2.3.1 Primal and dual representations of a non-binary CSP *P1*

2.3.2 Hidden Variable Method

In [Dec90], Dechter shows how to represent any non-binary relation with binary relations using hidden variable method. Unlike the dual graph method which throws away the original variables and introduces new dual variables into the dual graph, the hidden variable method keeps all the primal nodes (variables) of the original CSP and adds new nodes which represent the primal constraints to the hidden representation. The hidden variable method is also known as the **hidden encoding**.

In the hidden variable representation, the set of variables includes all of the variables of the original problem with no changes to their domains plus a new set of "hidden" variables which were called h-variables.

These "hidden" variables are constructed as follows. For each constraint C_i in the original problem we add an h-variable H_i. The domain of H_i consists of a unique identifier for every satisfying tuple in the constraint C_i. For every h-variable H_i we add a binary constraint between H_i and each of the variables involved in the constraint C_i, In this way the "hidden" variable H_i and an original variable v_k are thus constrained. Every value of H_i corresponds to a tuple of values for the variables in the constraint C_i and thus defines a unique value for v_k. Hence, the binary constraint between H_i and v_k consists of a unique value for v_k for every value of H_i.

Consider the following non-binary CSP from [Nag01] which has 6 variables and 4 constraints. Each variable has the same domain $\{0, 1\}$. The constraints are:

$$C_{1,2,6} : v_1 + v_2 + v_6 = 1$$

$$C_{1,3,4} : v_1 - v_3 + v_4 = 1$$

$$C_{4,5,6} : v_4 + v_5 - v_6 \geq 1$$

$$C_{2,5,6} : v_2 + v_5 - v_6 = 0$$

Given below, Figure 2.3.2 is the hidden variable representation for the CSP above.

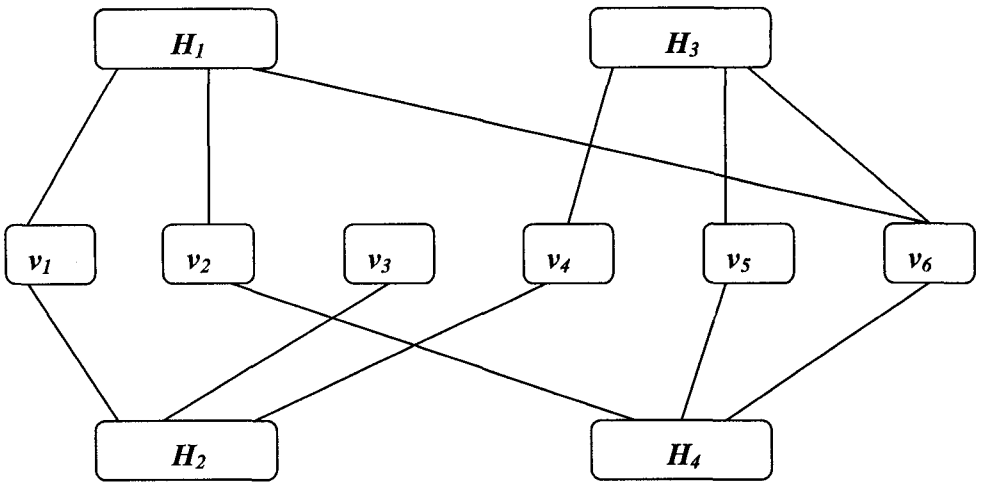


Figure 2.3.2 An example of hidden variable representation for a non-binary CSP

In Figure 2.3.2, there are ten variables: the six original variables v_1, v_2, v_3, v_4, v_5, v_6 and four "hidden" variables H_1, H_2, H_3, H_4, one for each constraint in the original problem $C_{1,2,6}$, $C_{1,3,4}$, $C_{4,5,6}$, $C_{2,5,6}$. For example, the constraint $C_{1,2,6}$ has a corresponding h-variable H_1, whose domain can be the set$\{1, 2, 3\}$ (a unique identifier for each of the seven tuples in the constraint). We can define a correspondence between the values of H_1, and the tuples in $C_{1,2,6}$ as follows:

$$1 \rightarrow (0, 0, 1), 2 \rightarrow (0, 1, 0), 3 \rightarrow (1, 0, 0)$$

Then, we add a constraint between the pairs of variables $\{v_1, H_1\}$, $\{v_2, H_1\}$ and $\{v_6, H_1\}$, giving the binary constraints,

$$Cv_{1,H1} = \{(0, 1), (0, 2), (1, 3)\}$$

$$Cv_{2,H1} = \{(0, 1), (1, 2), (0, 3)\}$$

$$Cv_{6,H1} = \{(1, 1), (0, 2), (0, 3)\}$$

For example, for binary constraint $Cv_{1,H1}$, the value 1 for H_1 corresponds to the tuple $(0, 0, 1)$ in which $v_1 = 0$. Hence, $H_1 = 1$ is only compatible with $v_1 = 0$.

After the two well known methods, dual graph method and hidden variable method, were proposed in [DP89] and [Dec90], some research which is based on systematic

search and problem reduction techniques has been done. In [BB98], the dual graph method and the hidden variable method are compared under forward checking which is a backtracking-based algorithm. In [BB98] Bacchus and van Beek also give some guidance for when one should consider translating between non-binary and binary representations. In [SW99] Stergiou and Walsh extend the above results and compare the dual encoding to the hidden encoding, and they also give transformations between the dual encoding and hidden encodings. [BCBW02] is an extension of [BB98] and [SW99], which performs a detailed formal comparison of the dual encoding and hidden variable encoding under forward checking and maintaining arc consistency algorithms. In [Nag01] Nagarajan presents new encodings based on dual encodings for non-binary constraint satisfaction problems and extends the standard forms of local consistency defined in the dual representation.

2.4 Search

In CSP research more effort probably has been spent on searching than in other approaches. Since different constraint satisfaction problems have different problem characters and solution requirement, a large amount of search methods are developed to solve CSPs. Search methods can be roughly classified into two categories: systematic and local search.

2.4.1 Systematic Search

Often systematic search method for solving CSPs is a combination of a standard backtracking procedure, along with problem reduction techniques before and interleaved during search. Problem reduction techniques transform CSPs to equivalent but hopefully easier problems by reducing the size of the domains and constraints in the original problems [Tsa93]. The basic idea behind problem reduction involves the

removal of redundant values from the domains of the variables and the tightening of the constraints so that the size of the search space decreases. For example, given two variables v_1 and v_2, each of which has the same domain $\{1, \ldots, 10\}$, and a binary constraint between v_1 and v_2 is given as $v_1 + v_2 < 5$. It is possible to see that the domain of each variable can be easily tightened with a number of redundant values removed from both of the domains, so that they are changed to $\{1, 2, 3\}$. Problem reduction normally does not produce solutions, but can be done as pre-processing step for another algorithm, or step by step, interwoven with the exploration of the search space by a search algorithm. In the latter case, subsets of the search space are cut off, saving the search algorithm the effort of systematically investigating the eliminated elements, which otherwise would happen, even repeatedly. In [Mac77] Mackworth defines three local consistencies which are node, arc and path consistency to characterize the property of binary constraint networks. In [Fe78] Freuder generalizes this to k-consistency.

Many systematic search algorithms such as forward checking [HE80], back-jumping [Gas78], and constraint-directed backtracking (CDBT) [PG97] have been proposed, most of which are variations of the basic backtracking method. These search methods are capable to investigate the entire search space in a systematic manner which guarantees that eventually either all the solutions are found, or, if no solution exists, this fact is determined with certainty. This typical property of algorithms based on systematic search is called **completeness**.

The basic backtracking algorithm was first generalized by Bitner and Reingold in [BR75]. The backtracking algorithm (BT) includes a recursive procedure which explores the search space under certain variable order and domain value order. In algorithm BT, variables are instantiated sequentially, i.e., variables are assigned values according to a kind of variable order. Once all the variables relevant to a constraint are instantiated, the recursive procedure will check the validity of the constraint. If a partial instantiation violates any of the constraints, backtracking is performed to the most

recently instantiated variable which still has alternative values available. Since the BT algorithm will always backtrack to the last decision when it becomes unable to proceed, it is also called chronological backtracking. Figure 2.4.1.1 [Tsa93] shows the control of BT and Figure 2.4.1.2 gives the pseudo code which describes the BT algorithm in detail.

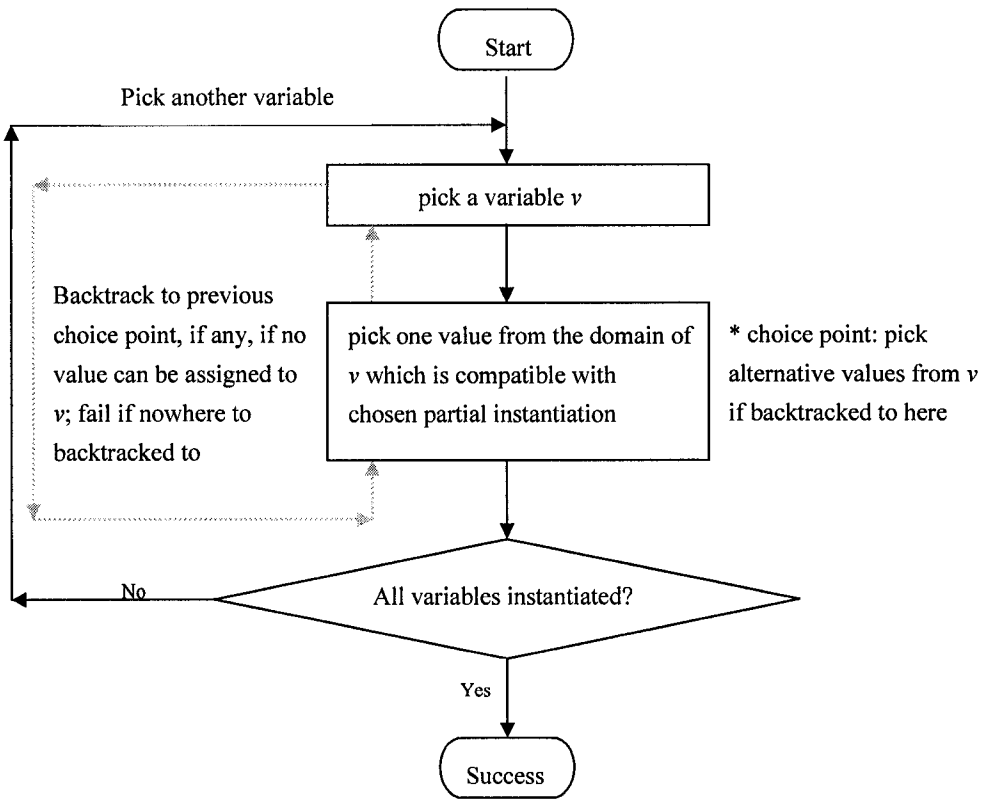


Figure 2.4.1.1 Control of backtracking algorithm

Clearly, whenever a partial instantiation violates a constraint, backtracking is able to prune off a subspace from the Cartesian product of all variable domains. Kumar points out in [Kum87] that the backtracking method essentially performs a depth-firth search of the space of potential solutions of the CSP.

```
procedure backtracking(V, D, C)
begin
bt(V, { }, D, C)
end

prodedure bt(VARS, ENV, D, C)
/* VARS is a set of variables which have not been instantiated */
/* ENV is a partial instantiation */
begin
1.  if VARS = { } then
2.      return ENV
3.  else
4.      pick one variable v from VARS
5.      repeat
6.          pick one value x from Dv
7.          delete x from Dv
8.          if ENV + {< v, x >} violates no constraints in C then
9.              RESULT := bt( VARS-{v}, ENV+{< v, x >}, D, C)
10.             if RESULT <> { } then
11.                 return RESULT
12.             endif
13.         endif
14.     until Dv = { }
15.     return { }
16. endif
end
```

Figure 2.4.1.2 Backtracking algorithm (BT)

Consider the following map coloring problem as a binary CSP:

Variables: v_1, v_2, v_3

Domains:

Domain of v_1: $Dv_1 = \{red, blue, green\}$

Domain of v_2: $Dv_2 = \{red, blue\}$

Domain of v_3: $Dv_3 = \{red, green\}$

Constraints:

C_1: $v_1 \neq v_2$

C_2: $v_1 \neq v_3$

C_3: $v_2 \neq v_3$

Now we launch algorithm BT under the variable order $\{v_1, v_2, v_3\}$ to get the solutions of the above map coloring problem. In this example we use BT to find all the solutions, so if one solution is found and there are other possible instantiations haven't been explored, BT will continue the search procedure.

1) For v_1=*red*, it violates no constraints, go on to assign value for v_2;
2) For v_1=*red*, v_2=*red*, it violates the constraint $v_1 \neq v_2$, do backtracking;
3) For v_1=*red*, v_2=*blue*, it violates no constraints, go on to assign value for v_3;
4) For v_1=*red*, v_2=*blue*, v_3=*red*, it violates the constraint $v_1 \neq v_3$, do backtracking;
5) For v_1=*red*, v_2=*blue*, v_3=*green*, it satisfies all the constraints, thus, it is a solution, then, do backtracking.
6) For v_1=*blue*, it violates no constraints, go on to assign value for v_2;
7) For v_1=*blue*, v_2=*red*, it violates no constraints, go on to assign value for v_3;
8) For v_1=*blue*, v_2=*red*, v_3=*red*, it violates the constraint $v_2 \neq v_3$, do backtracking;
9) For v_1=*blue*, v_2=*red*, v_3=*green*, it satisfies all the constraints, thus, it is a solution, then, do backtracking.
10) For v_1=*blue*, v_2=*blue*, it violates the constraint $v_1 \neq v_2$, do backtracking;
11) For v_1=*green*, it violates no constraints, go on to assign value for v_2;
12) For v_1=*green*, v_2=*red*, it violates no constraints, go on to assign value for V_3;
13) For v_1=*green*, v_2=*red*, v_3=*red*, it violates the constraint $v_2 \neq v_3$, do backtracking;
14) For v_1=*green*, v_2=*red*, v_3=*green*, it violates the constraint $v_1 \neq v_3$, do backtracking;
15) For v_1=*green*, v_2=*blue*, it violates no constraints, go on to assign value for V_3;
16) For v_1=*green*, v_2=*blue*, v_3=*red*, it satisfies all the constraints, thus, it is a solution, then, do backtracking.
17) For v_1=*green*, v_2=*blue*, v_3=*green*, it violates the constraint $v_1 \neq v_3$, since all the possible values have been assigned to v_1, v_2 and v_3, BT terminates.

In Step 5, 9 and 16 we get the solutions for this map coloring problem:

Solution 1: $\{< v_1, red>, < v_2, blue>, < v_3, green>\}$

Solution 2: $\{< v_1, blue >, < v_2, red >, < v_3, green >\}$

Solution 3: $\{< v_1, green >, < v_2, blue >, < v_3, blue >\}$

The time complexity of BT is exponential. If a CSP has n variables, each of which has a domain with size a, and there are e constraints in this problem. Since there are altogether a^n possible candidate solutions and for each of the n-tuples (candidate solution) all the constraints must be checked once in the worse case, the time complexity of algorithm BT is $O(a^n e)$ [Tsa93]. The search efficiency could be improved if the domain size can be reduced. This could be achieved by problem reduction techniques.

2.4.2 Local Search

Local search launches the search process at some random state which is an instantiation including all variables and then continues by iteratively moving from one state to another in the search space in a non-deterministic manner, guided by heuristics. The next move is partly determined by the outcome of the previous move. Typically local search methods are incomplete which means even if the given CSP has a solution, they are not guaranteed to find it eventually. They are also not guaranteed to report there is no solution if the given CSP has no solution. But local search have always been attractive as will be shown below in [Hoo98]: First, many constraint satisfaction problem are constructive by nature, it is known that they are solvable and what is required is actually the generation of a solution rather than just deciding whether a solutions does exist. Secondly, in many real-world applications often the time to find a solution is limited. In these situations systematic methods often have to be aborted after the given time has been exhausted and none of the solutions have been found while in the same situation local search methods may offer a solution within the time limit.

The local search methodology often uses the following terms some of which originally appear in [Bar98]:

- **state** (node): one possible assignment of all variables from their domains; the number of states is equal to the product of each domain's size.
- **evaluation value**: the number of satisfied constraints of the state.
- **neighbor**: the state which is obtained from the current state by changing one variable's value from its domain.
- **move**: one move means to pick a neighbour state from the current state's neighbourhood and make this neighbour state as the next current state.
- **strict local optimum**: the state that is not a solution and the evaluation values of all of its neighbors are smaller than the evaluation value of this state.
- **plateau**: the state that is not a solution and the evaluation values of all of its neighbors are equal to the evaluation value of this state.
- **local optimum**: the state that is not a solution and the evaluation values of all of its neighbors are smaller than or equal to the evaluation value of this state. Local optimum can be seen as a state which is either a plateau or a strict local optimum.
- **global maximum**: the state is a solution.

Hill-climbing methods are probably the most known strategies of local search [Bar98]. These hill-climbing methods use heuristics to incrementally alter inconsistent value assignments of all the variables and move towards a solution. Their stochastic nature generally gives up the guarantee of completeness which is provided by systematic search methods [Bar98].

The problem with Hill Climbing algorithms in general is that they do not guarantee to find a solution or report no solution. They may settle in strict local optima, where all neighbors are worse than the current state, though the current state is not a solution. They may also loop in plateaus, where all the neighbors have the same evaluation value as the current state (see Figure 2.4 [Tsa93]). In these situations local search algorithms

terminates the current loop and randomly pickup an initial state again.

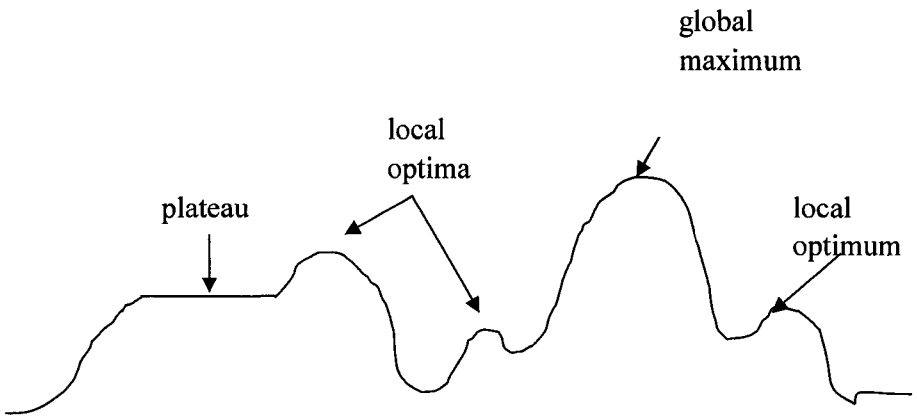


Figure 2.4 Possible problems with hill climbing algorithms: the algorithms may stay in plateaus or local optima

Recently local search has been attractive in solving constraint satisfaction problems. In [SLM92] GSAT was introduced as a greedy local search method for solving propositional satisfiability problems. GSAT can also be extended to solve constraint satisfaction problems. [MPJL93] proposes the min-conflicts heuristic repair method which can be used in hill-climbing search. One major problem of basic local search algorithms is that they may get stuck in local optima. To this aim one general method is restarting from a new randomly generated initial state. Another common extension to prevent getting stuck in local optima is the application of random walk [SKC94] which modifies the value of a variable involved in a violated constraint randomly by choosing some other value than the current one. Another heuristic that allows escaping from local optima is Tabu search [Glo89] which can leave local optima by forbidding moves to recently visited states. Tabu search and random walk heuristics are compared in [SSS97]. An empirical study of min-conflicts heuristics for binary CSPs is presented in [PR95]. Hoos and Stützle propose an empirical methodology [HS98] which is based on characterising run-time distributions of stochastic local search algorithms on single problem instances.

2.5 Conclusions

In this chapter we gave a brief introduction to constraint satisfaction problems and briefly discussed different problem solving techniques. Among the different approaches, we will focus on observing behaviours of several local search methods in primal and dual constraint graphs in Chapter 3.

Chapter 3

Local Search in Primal and Dual Constraint Graphs

In this chapter we describe several local search algorithms such as simple hill climbing, steepest ascent hill climbing and min-conflicts heuristics. We then illustrate how these local search methods are applied in primal constraint graph and dual constraint graph. Finally we give the empirical study design and discuss a statistical analysis method we used in this thesis for comparing the means of two groups.

3.1 Local Search Algorithms

3.1.1 General Local Search Strategy

All the hill climbing algorithms described in this thesis are based on a common idea known as *local search.* In local search, an initial state (valuation of variables) is generated and the algorithm moves from the current state to a neighbouring state until a solution has been found or the resources available such as maximum number of moves and maximum number of iterations are exhausted. This idea is expressed in the following general local search algorithm (Figure 3.1.1) that enables implementation of many particular local search algorithms via definitions of specific procedures. In the procedure we presented here, the evaluation value means how many constraints are

satisfied. The more constraints are satisfied, the larger is the evaluation value. When all the constraints are satisfied, the evaluation value of such a state equals to the number of constraints in the original CSP.

```
Procedure LocalSearch (Max_Moves, Max_Iteration)
begin
1.    s ← random valuation of variables
2.    for i:=1 to Max_Moves do
3.       for j:=1 to Max_Iteration do
4.          if evaluation(s)= the number of all constraints then
5.             return s
6.          endif
7.          select n in neighborhood(s)
8.          if acceptable(n) then
9.             s ← n
10.         endif
11.      endfor
12.      s ← restartState(s);
13.   endfor
14.   return s
end
```

Figure 3.1.1 General local search algorithm

3.1.2 Simple Hill-climbing Algorithm

Hill-climbing methods are probably the best known strategies of local search. First we look at the simple hill-climbing algorithm which is presented in Figure 3.1.2. The idea of simple hill-climbing is:

1. Start at randomly generated state.
2. Move to the neighbor with a better evaluation value.
3. If a local optimum is reached then restart at other randomly generated state.

This procedure repeats till the solution is found. The simple hill-climbing algorithm does not need to explore all the neighbors of the current state. But the order of the neighborhood states may make a difference since the simple hill-climbing method will choose the neighbor which has a better evaluation value by such order. Thus it determines which part of the search space will be investigated next.

```
Procedure SimpleHillClimbing(Max_Restarts)
begin
1.  for i:=1 to Max_Restarts do
2.     s ← random instantiation of all variables
3.     while evaluation(s)<the number of all constraints do
4.        findNeighbor:
5.        if no neighbor left in neighborhood of s then
6.           goto restart;   /* a local optimum is reached */
7.        else select n in neighborhood(s)
8.        endif
9.        remove n from neighborhood(s)
10.       if evaluation(n)=the number of all constraints then
11.          return n;
12.       endif;
13.       if evaluation(n)> evaluation(s) then
14.          s ← n
15.       else goto findNeighbor
16.       endif
17.    endwhile
18.    return s
19.    restart:
20. endfor
end
```

Figure 3.1.2 Simple hill-climbing algorithm

3.1.3 Simple Hill-climbing Flowchart

Below we give the flowchart (Figure 3.1.3) for simple hill-climbing according to the algorithm introduced in 3.1.2.

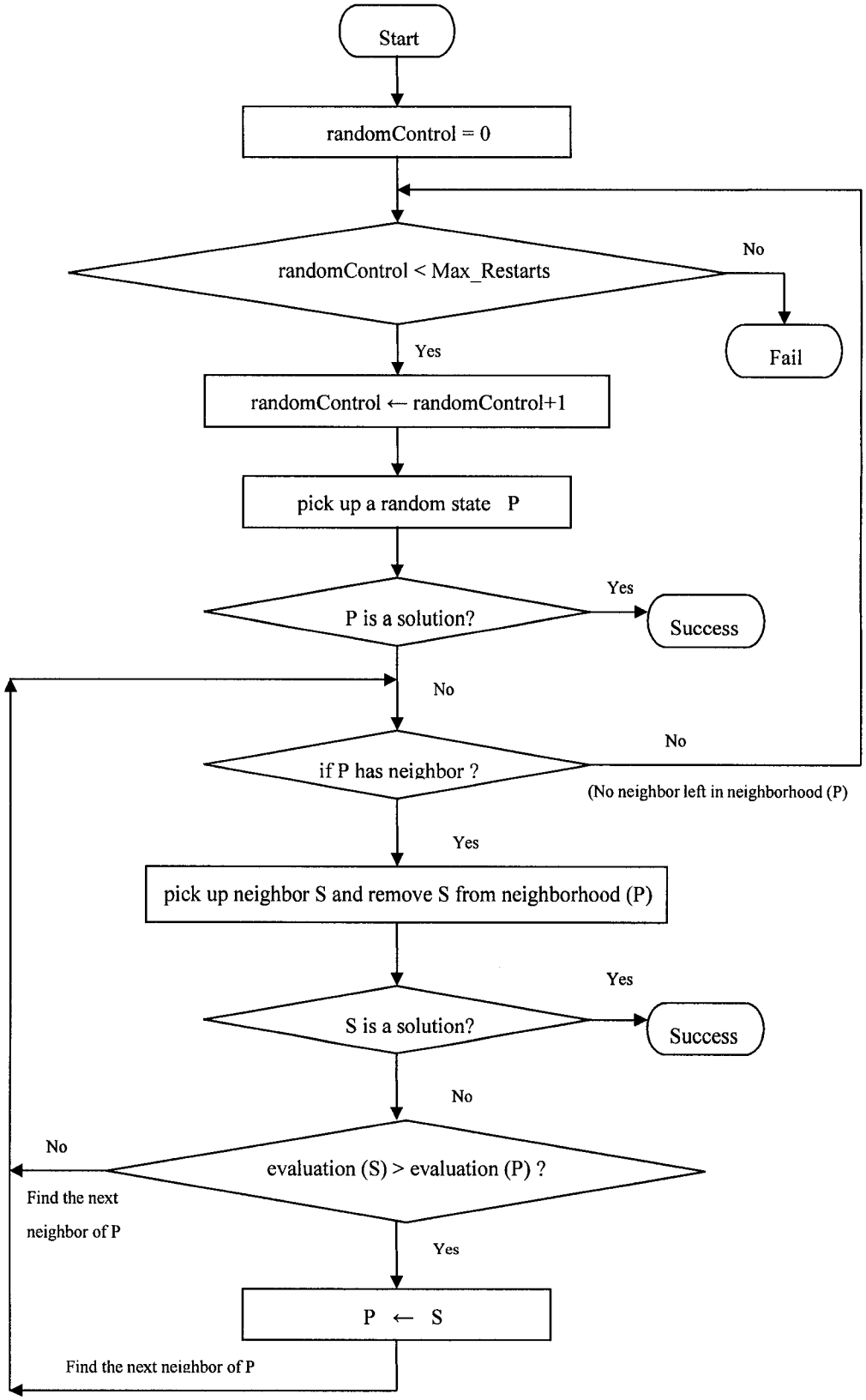


Figure 3.1.3 Simple hill-climbing flowchart

3.1.4 Simple Hill-climbing Example

Now we use an example to illustrate how simple hill-climbing works to solve the map coloring problem (Figure 2.1) in its primal representation. This time we use domain $\{1, 2, 3\}$ instead of using $\{red, blue, green\}$ in the following CSP (Figure 3.1.4.1):

Variables: v_1, v_2, v_3, v_4
Domains:
Domain of v_1 $D(v_1)$: $\{1, 2, 3\}$
Domain of v_2 $D(v_2)$: $\{1, 2, 3\}$
Domain of v_3 $D(v_3)$: $\{1, 2, 3\}$
Domain of v_4 $D(v_4)$: $\{1, 2, 3\}$
Constraints:
C_1: $v_1 \neq v_2$
C_2: $v_1 \neq v_3$
C_3: $v_1 \neq v_4$
C_4: $v_2 \neq v_3$
C_5: $v_3 \neq v_4$

Figure 3.1.4.1 An example of a binary CSP

Figure 3.1.4.2 is the search tree of simple hill-climbing working in primal representation of the binary CSP in Figure 3.1.4.1. Note that each node of the search tree has the format: $v_1v_2v_3v_4$ (evaluation value). In this example, simple hill-climbing uses 3 moves and visits 11 search nodes to find the solution $\{< v_1, 2>, < v_2, 3>, < v_3, 1>, < v_4, 3>\}$.

3.1.5 Steepest Ascent Hill-climbing

Steepest ascent hill-climbing algorithm which is presented in Figure 3.1.5 differs with the simple hill-climbing method in that the former evaluates all the neighbors of the current state and chooses the best one while the latter only explores part of the neighborhood states and select a better one to move. The steepest ascent hill-climbing algorithm has to explore all the neighbors of the current state before choosing the move and such a choosing process may take a lot of time.

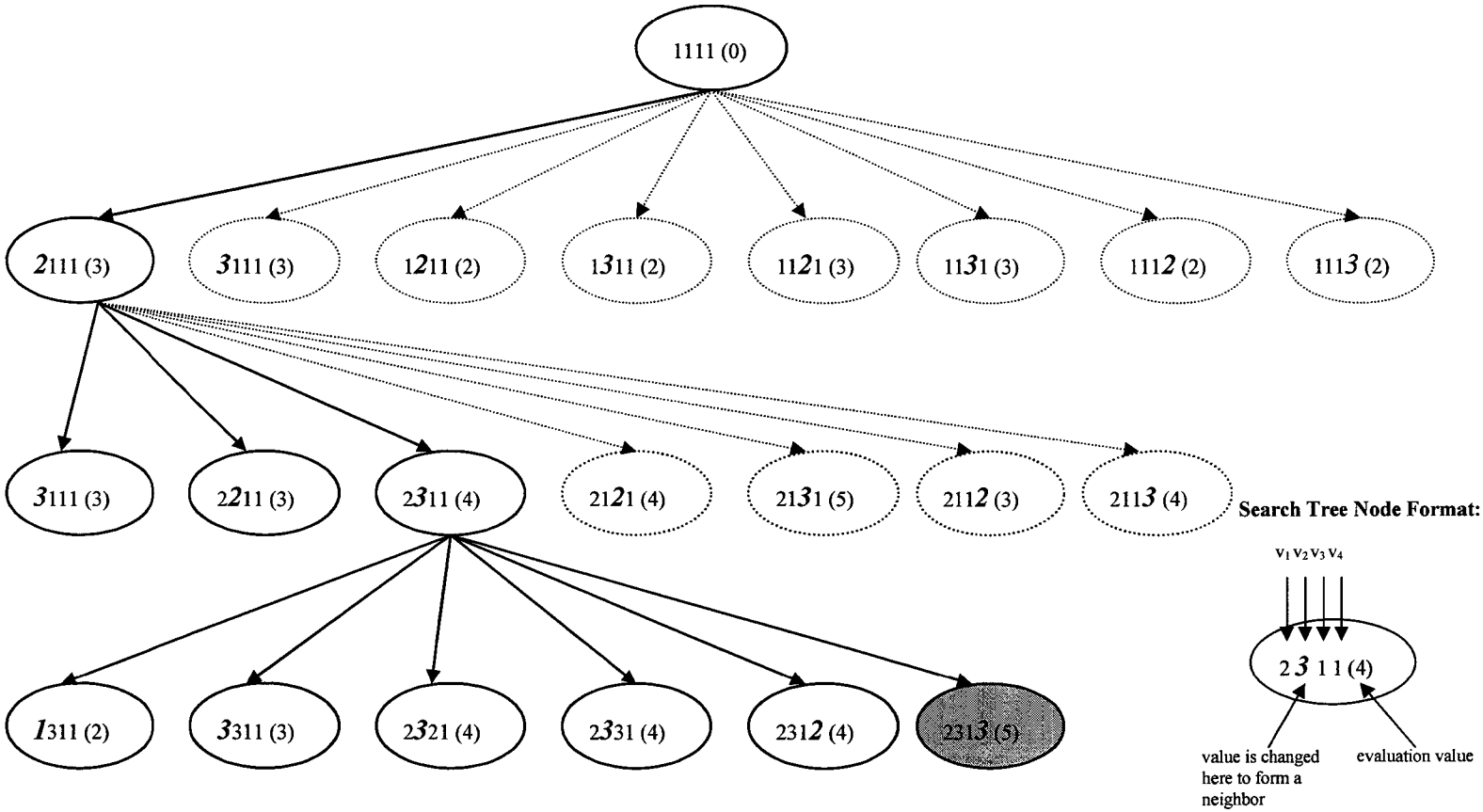


Figure 3.1.4.2 Search tree of simple hill-climbing in primal representation

```
Procedure SteepestAscentHillClimbing(Max_Restarts)
begin
1.  for i:=1 to Max_Restarts do
2.      s ← random instantiation of all variables
3.      while evaluation(s)<the number of all constraints do
4.          find the best neighbor n which has the largest evaluation value
5.          if evaluation(n)=the number of all constraints then
6.              return n;
7.          endif;
8.          if evaluation(n)> evaluation(s) then
9.              s ← n
10.         else goto restart   /* a local optimum is reached */
11.         endif
12.     endwhile
13.     return s
14.     restart:
15. endfor
end
```

Figure 3.1.5 Steepest ascent hill-climbing algorithm

3.1.6 Steepest Ascent Hill-climbing Flowchart

Figure 3.1.6 is the flowchart for steepest ascent hill-climbing according to the algorithm introduced in 3.1.5.

3.1.7 Steepest Ascent Hill-climbing Example

Figure 3.1.7 is the search tree of steepest ascent hill-climbing working in primal representation of the binary CSP in Figure 3.1.4.1. In this example, steepest ascent hill-climbing uses 2 moves and visits 13 search nodes to find the solution $\{< v_1, 2>, < v_2, 1>, < v_3, 3>, < v_4, 1>\}$.

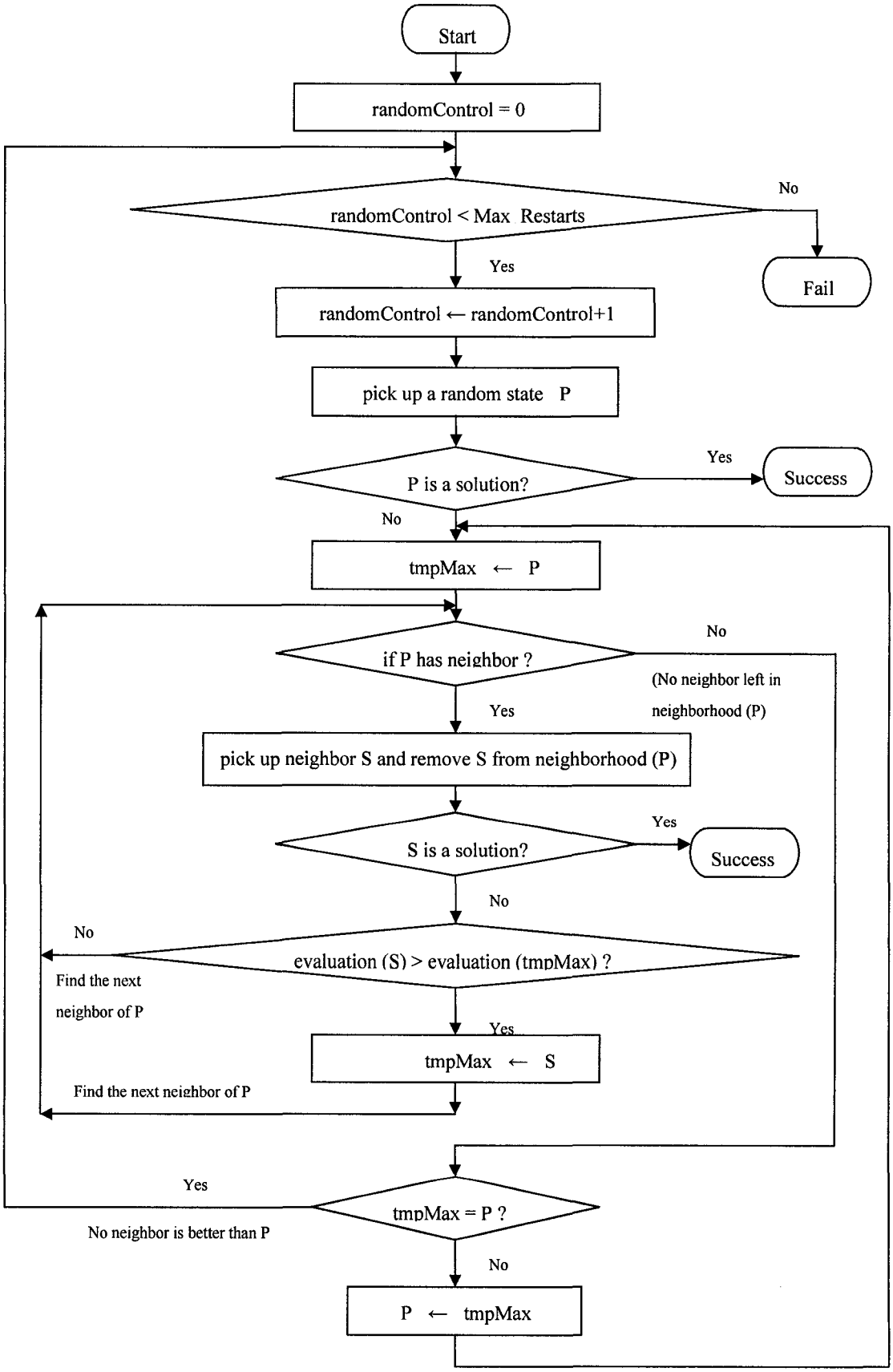


Figure 3.1.6 Steepest ascent hill-climbing flowchart

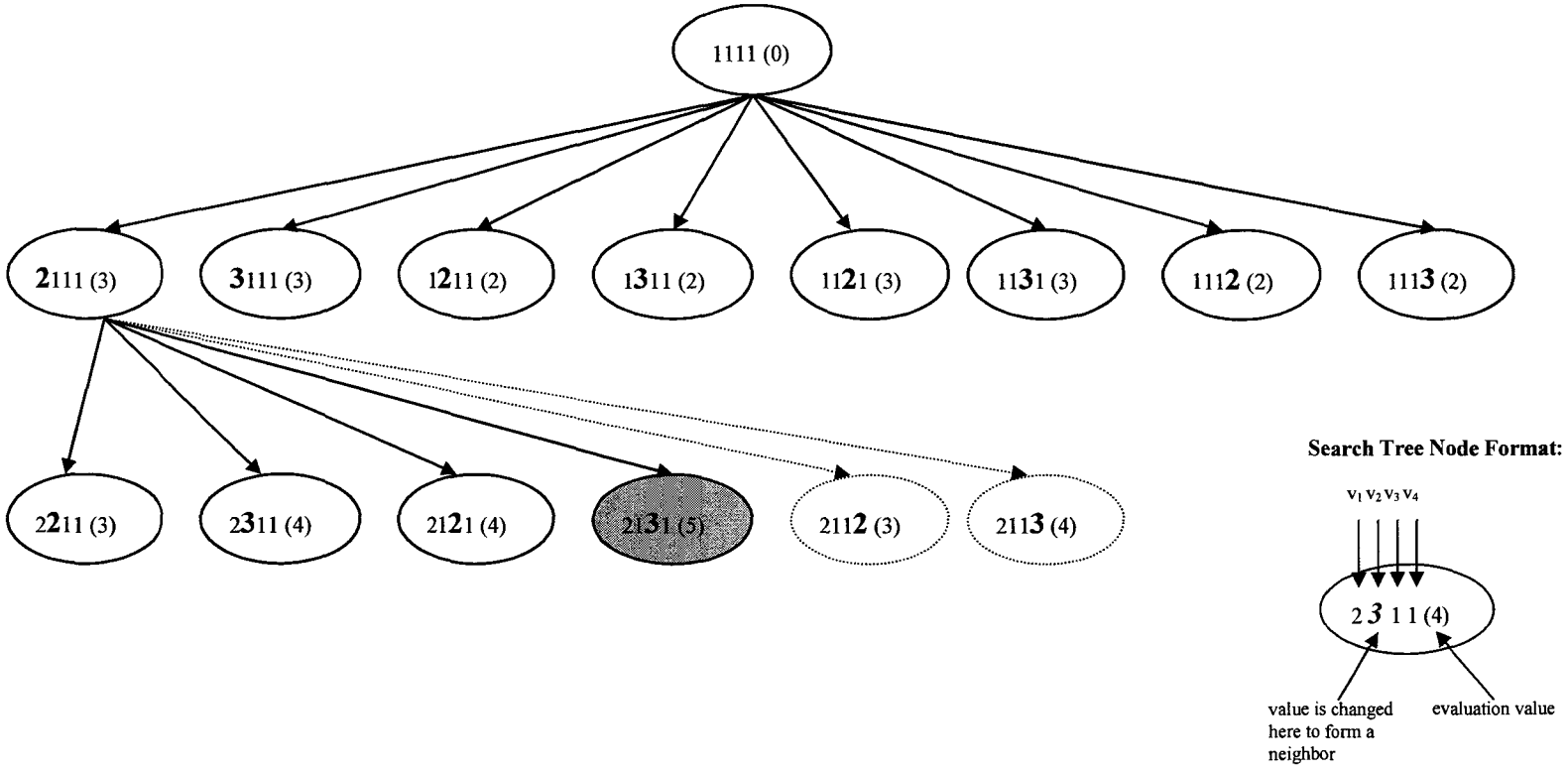


Figure 3.1.7 Search tree of steepest ascent hill-climbing in primal representation

3.1.8 Min-conflicts Heuristics Hill-climbing

To avoid exploring all the neighbors of the current state some heuristics were proposed to find a next move. Min-conflicts heuristics is a heuristic repair method which attempts to minimize the number of constraint violations after each step. Min-conflicts heuristics was first introduced in [MPJL93]. The min-conflicts heuristics can be used with a variety of different search strategies such as backtracking-based search and local search.

When applying min-conflicts heuristics in local search method, min-conflicts heuristics hill-climbing chooses randomly any conflicting variable, i.e., the variable that is involved in any unsatisfied constraint, and then picks a value which maximizes the number of satisfied constraints (break ties randomly). If no such value exists, it picks randomly one value which can form a neighbor that has the same number of satisfied constraints as the current state does. Min-conflicts heuristics hill-climbing does not explore all the neighbors of the current state, but it explores all those neighbors which are related with the randomly chosen conflicting variable by changing that variable's value. If all the neighbors have less number of satisfied constraints than the current state, min-conflicts heuristics hill-climbing will restart the search procedure. The min-conflicts heuristics algorithm for hill-climbing is showed in Figure 3.1.8.

In the following parts of this thesis we also use min-conflicts heuristics to represent using this heuristic repair method in hill-climbing algorithm.

3.1.9 Min-conflicts Heuristics Flowchart

Figure 3.1.9 is the flowchart for min-conflicts heuristics according to the algorithm introduced in 3.1.8.

3.1.10 Min-conflicts Heuristics Example

Figure 3.1.10.1 is the search tree of min-conflicts heuristics working in primal representation of the binary CSP in Figure 3.1.4.1. In this example, min-conflicts heuristics uses 5 moves and visits 10 search nodes to find the solution $\{< v_1, 3>, < v_2, 2>, < v_3, 1>, < v_4, 2>\}$.

```
Procedure MinConflicsHeuristicsHillClimbing (Max_Restarts)
begin
1.  for i:=1 to Max_Restarts do
2.     s ← random instantiation of all variables
3.     while evaluation(s)<the number of all constraints do
4.        randomly pick a variable V which is currently in conflict
5.        neighborhood(s)=change V's value from its domain
6.        choose the best neighbor n which has the largest evaluation value
           and evaluation(n)> evaluation(s)
7.        if evaluation(n)=the number of all constraints then
8.           return n;
9.        endif;
10.       if no such neighbor in Step 6 exists, then
11.          randomly choose a neighbor n which evaluation(n)= evaluation(s)
12.          s ← n
13.       endif
14.       if no neighbor' evaluation value >= evaluation(s) then
15.          goto restart
16.       endif;
17.    endwhile
18.    return s
19.    restart:
20. endfor
end
```

Figure 3.1.8 Min-conflicts heuristics hill-climbing algorithm

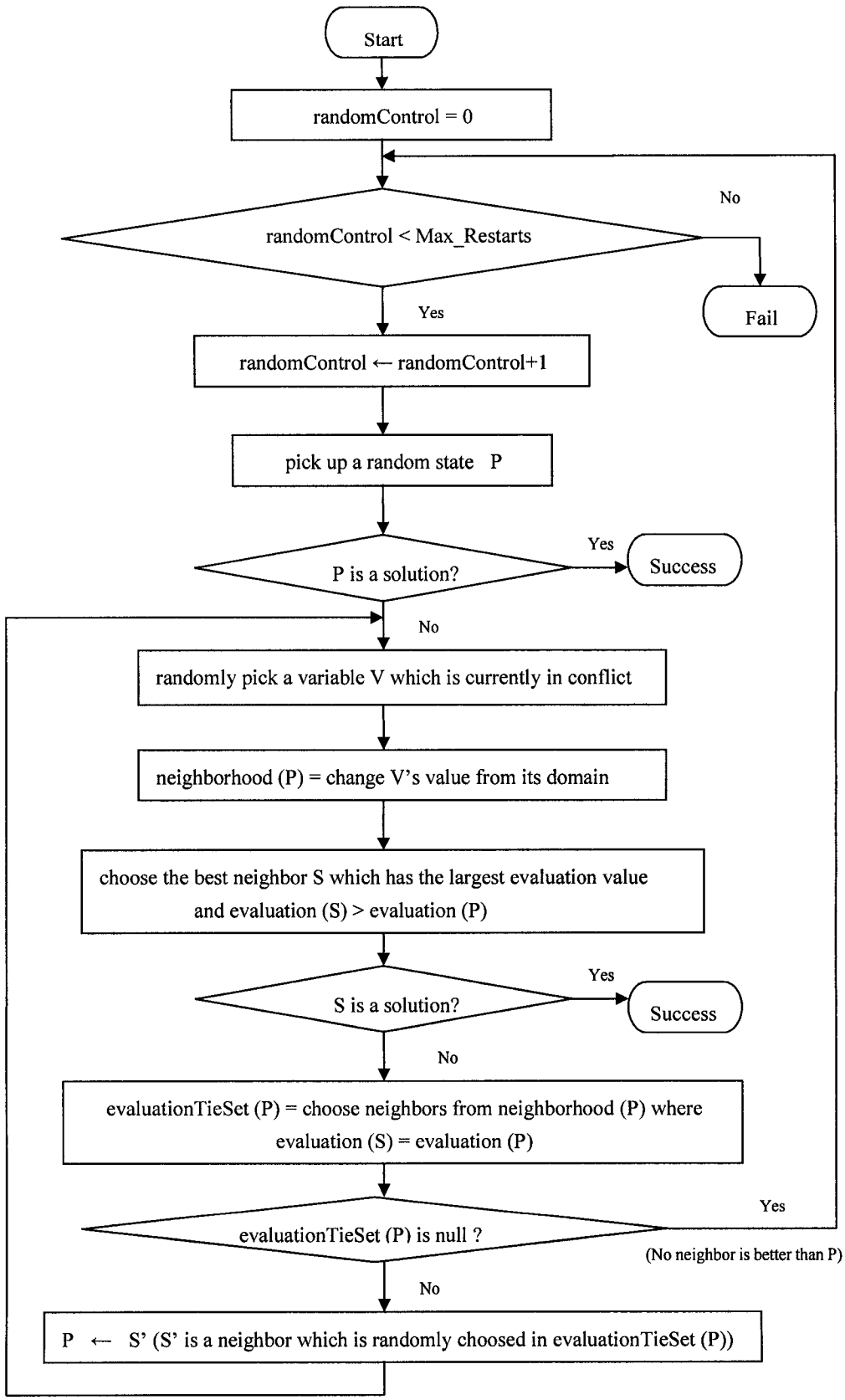


Figure 3.1.9 Min-conflicts heuristics hill-climbing flowchart

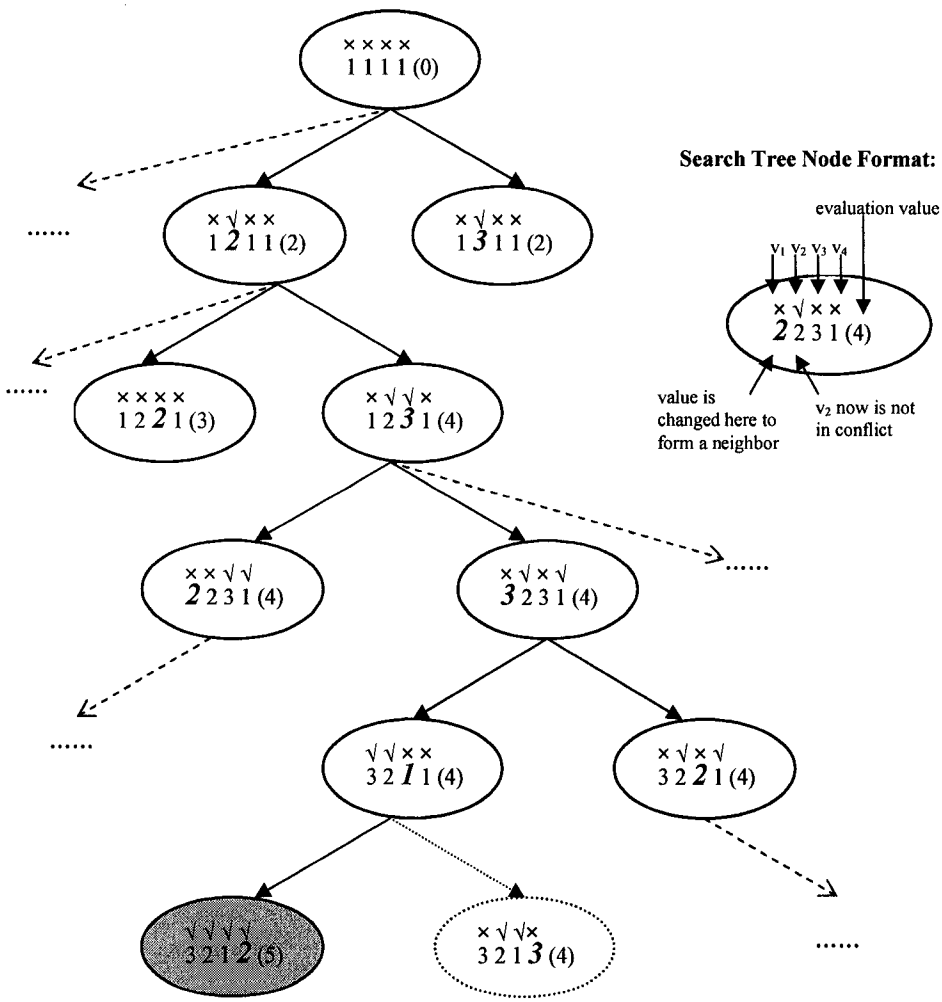


Figure 3.1.10.1 Search tree of min-conflicts heuristics hill-climbing in primal representation

The search trees of min-conflicts heuristics can be different even when the algorithm begins with the same start state. For example, in Figure 3.1.10.1, when the algorithm has the node (1231) as the current state after 2 moves, it randomly chooses a conflicting variable v_1 to get its neighbors which are (2231) and (3231). But if the algorithm chooses another conflicting variable v_4 to get its neighbors, it will find a solution $\{< v_1, 1>, < v_2, 2>, < v_3, 3>, < v_4, 2>\}$ immediately (See Figure 3.1.10.2). Since min-conflicts heuristics randomly chooses a variable which is in conflict to get

neighbors, the search trees are different. Another reason to cause a different search tree is that min-conflicts heuristics will randomly chooses a neighbor which has the same evaluation value as the current state to continue the move when all the neighbors are not better than the current state (break ties randomly). In Figure 3.1.10.1 and 3.1.10.2, those dashed lines with arrow on one end indicate there are different ways to continue the search.

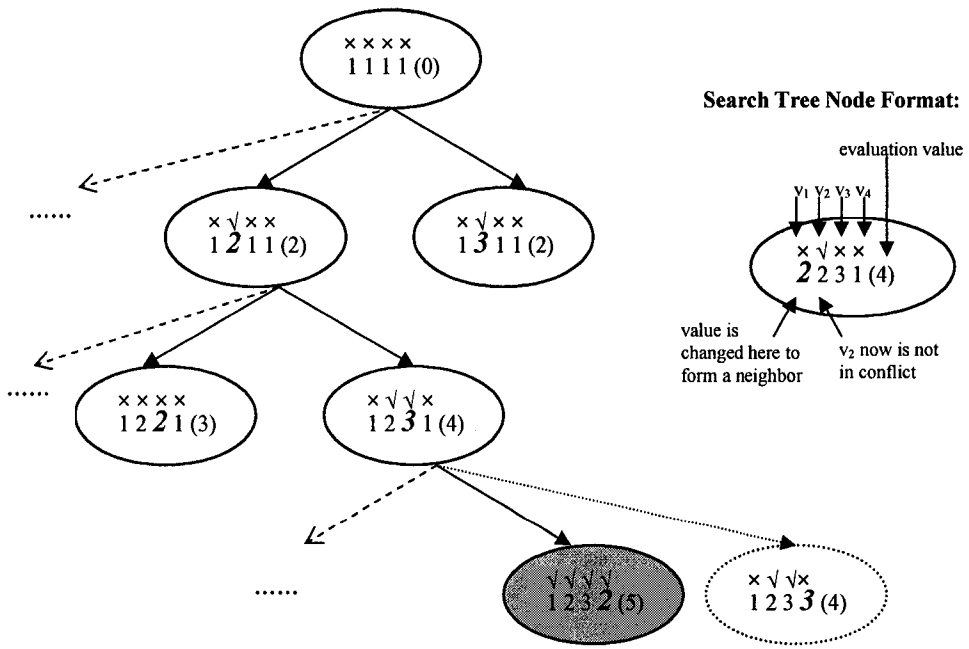


Figure 3.1.10.2 Different search tree of min-conflicts heuristics hill-climbing in the same primal representation

3.2 Local Search in Primal and Dual Constraint Graphs

The dual encoding which was first introduced to solve CSPs by Dechter and Pearl in [DP89] gives a way to transform a non-binary CSP into a binary CSP. [Nag01] has a thorough study on the dual encodings. In the last decade the dual encoding techniques

for CSP solving have been mainly concentrated on systematic search coupled with various problem reduction methods before and interleaved during the search procedure. Almost in the same period the local search methods are frequently used in CSP solving, but most of the research only exploited local search in binary CSPs. Empirical study of local search on non-binary CSPs has seldom been mentioned. Thus we propose our approach to investigate local search behaviour on non-binary CSPs both in primal and dual representations. The local search methods which are studied in this thesis for solving non-binary CSPs in our research are simple hill-climbing, steepest ascent hill-climbing and min-conflicts heuristics hill-climbing [MPJL93].

Different local search methods visit different number of nodes during the search procedure. During one move from one current state to the next current state, steepest hill climbing will investigate all the possible neighbours while simple hill climbing and min-conflicts heuristics hill-climbing only explore part of them. For each node in the search tree the three local search methods will check all the constraints. So steepest hill climbing may take longer time to find a solution since it may visit more search tree nodes and check more constraints. But steepest hill climbing may use the least moves to get a solution if it does not get stuck in a local optimum.

Since in the dual representation the dual domains are consistent value combinations which have satisfied the primal constraints inside the dual variables, if there are tight constraints in the original CSP, local search on dual constraint graph may find a solution within a shorter time while local search on primal constraint graph need more moves to gradually get such tight constraints satisfied. But when the constraints are looser, local search on primal constraint graph may get a better performance than local search on dual constraint graph since under such situation the dual representation has big dual domain size which indicates a large number of neighbors need to be visited during each move.

In dual encodings the dual domains need to be explicitly stored. To fairly evaluate the performance of local search in primal and dual representations we store the primal constraints and dual domains both in an extensional form.

3.3 Empirical Study Design

In order to make the comparison objective and do the experiment efficiently, we design the requirement for empirical study as follows:

- Can deal with both binary and non-binary CSPs
- Represent primal constraints and dual domains extensionally
- Can change the number of variables
- Can change the domain size
- Can change the constraint tightness
- Provide enough information to evaluate the performance

According to the above requirement the local search procedure gets the original CSP described in a flat file as an input. In our design the original CSPs can be very flexible so that we can make a deep empirical study to investigate the behavior of local search on both primal and dual constraint graphs.

Experiment environment for empirical study:
The experiments are run on a P4 2.4G PC with Windows 2000. The programming language is Java.

3.3.1 Empirical Study Input Design

The input file is generated by a CSP problem generator which is described in Chapter 4. The content of the input file is as follows:

- Name of the problem instance
- Number of variables
- Size of each domain
- Number of constraints
- Variables involved in each constraint
- Tightness of each constraint

Below is an example of input flat file:

ProblemB01,4,3
1,3
2,3
3,3
4,5
1,1,2,90
2,2,3,4,50
3,1,4,10

The format of the above flat file:

Line 1:

ProblemB01,4,3

Line 1	ProblemB01	4	3
Description:	Problem Instance Name	Number of Variables	Number of constraints

Line 2 to 5:

1,3
2,3
3,3
4,5

Line 2	1	3
Description	Variable v_1	Domain size of v_1 is 3.

Line 3	2	3
Description	Variable v_2	Domain size of v_2 is 3.

Line 4	3	3
Description	Variable v_3	Domain size of v_2 is 3.

Line: 5	4	5
Description	Variable v_4	Domain size of v_4 is 5.

Line 6 to 8:

1,1,2,90
2,2,3,4,50
3,1,4,10

Line 6	1	1	2	90
Description	Constraint C_1	Variable v_1	Variable v_2	Tightness: 90%

Line 7	2	2	3	4	50
Description	Constraint C_2	Variable v_2	Variable v_3	Variable v_4	Tightness: 50%

Line 8	3	1	4	10
Description	Constraint C_3	Variable v_1	Variable v_4	Tightness: 10%

3.3.2 Empirical Study Output Design

To avoid getting stuck at local optima, local search often need to regenerate the initial state till it finds a solution and since local search investigates the search space in a non-deterministic manner, in each round the solution found generally is different. Below is the output in each round which finds a solution for the specific problem instance:

- The solution.
- How many times does the local search randomly generate the initial state?
- How many search tree nodes are visited during the search period?
- How long does it take to get the solution (time cost)?
- How many constraints are checked during the search period?

Note that "the nodes visited" for local search in primal constraint graph has a different meaning from it in dual constraint graph. For local search in primal, a node is a value assignment to all primal variables. For local search in dual, a node is a value

assignment to all dual variables. For example, we consider the CSP *P1* in Figure 2.3.1:

Variables: $\{v_1, v_2, v_3, v_4\}$

Constraints:

$C_{1,2,3}$: $v_1 + v_2 < v_3$

$C_{1,4}$: $v_1 < v_4$

$C_{2,3}$: $v_2 \neq v_3$

Domains:

$D(v_1) = \{1, 2\}$

$D(v_2) = \{0, 1\}$

$D(v_3) = \{1, 2, 3\}$

$D(v_4) = \{1, 2, 3\}$

Primal Representation of *P1*

Dual Variables: $\{C_{1,2,3}, C_{1,4}, C_{2,3}\}$

Dual Domains:

$D(C_{1,2,3}) = \{(1, 0, 2), (1, 0, 3), (1, 1, 3), (2, 0, 3)\}$

$D(C_{1,4}) = \{(1, 2), (1, 3), (2, 3)\}$

$D(C_{2,3}) = \{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3)\}$

Dual Constraints:

1. v_1 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$ and $D(C_{1,4})$
2. v_2 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$ and $D(C_{2,3})$
3. v_3 should be assigned the same values from the domains of the dual nodes: $D(C_{1,2,3})$ and $D(C_{2,3})$

Dual Representation of *P1*

When we assign value to each primal variable as $\{<v_1, 1>, <v_2, 0>, <v_3, 1>, <v_4, 1>\}$, such value assignment is a node in local search on *P1*'s primal representation. When we assign value to each dual variable as $\{<C_{1,2,3}, (1, 0, 2)>, <C_{1,4}, (1, 2)>, <C_{2,3}, (0, 1)>\}$, such value assignment is a node in local search on *P1*'s dual representation. Such difference also exists when we mention "the constraints checked" for local search on primal and dual representations.

For evaluating the behavior of local search both in primal and dual representations, each problem instance will be run for 100 rounds. For each problem instance, we keep the average values and their variances for each output parameter and there is a summary result as the following:

- The average number of times that local search randomly generates the initial state and its variance.

- The average number of search nodes visited during the search period and its variance.
- The average CPU time to get the solution (time cost) and its variance.
- The average number of constraints checked during the search period and its variance.

3.3.3 Empirical Study Comparisons

In this empirical study we intend to investigate the behaviour of local search on primal and dual representations over different problem instances. The local search characters are compared under a ratio which is the **move cost** of local search in dual representation comparing with the move cost of local search in primal representation. First we illustrate what the move cost is in steepest ascent hill-climbing. One move for local search is from one current state to the next current state. For each move in steepest ascent hill-climbing, the search procedure will investigate all the neighbors of the current state. For each neighbor, it will check all the constraints to get the evaluation value. The move cost in steepest ascent hill-climbing is the number of possible neighbors multiplies the number of constraints. In Figure 3.3.3.1 we give the move cost of steepest ascent hill-climbing in primal representation and dual representation. We use the notion $|SET|$ to represent the size of a set, i.e., the number of elements in this set. In Figure 3.3.3.1, $|D(v_i)|$ is the domain size of variable v_i and $|C|$ is the number of primal constraints in a CSP, $|D(C_j)|$ is the domain size of dual variable C_j i.e., $|D(C_j)|$ is the number of satisfied tuples in primal constraint C_j, and $|C_{dual}|$ is the number of dual constraints in a CSP.

1. The number of neighbors in each move for steepest ascent hill-climbing in primal representation:

$$\#\ neighbors\ of\ SteepestPrimal = \sum_{i=1}^{n} (|D(v_i)|-1)$$

2. For each neighbor it will check all the primal constraints: $|C|$

3. The move cost of steepest ascent hill-climbing in primal representation:

$$MV_{SteepestPrimal} = \sum_{i=1}^{n} (|D(v_i)|-1)\times|C|$$

4. The number of neighbors in each move for steepest ascent hill-climbing in dual representation:

$$\#\ neighbors\ of\ SteepestDual = \sum_{j=1}^{m} (|D(C_j)|-1)$$

5. For each neighbor it will check all the dual constraints: $|C_{dual}|$

6. The move cost of steepest ascent hill-climbing in dual representation:

$$MV_{SteepestDual} = \sum_{j=1}^{m} (|D(C_j)|-1)\times|C_{dual}|$$

Figure 3.3.3.1 Move cost of steepest ascent hill-climbing

For each neighbor in simple hill-climbing and min-conflicts heuristics, they also need to check all the constraints to get the evaluation value, but they do not need to explore all the neighbors as steepest ascent hill-climbing does. Thus the move cost of simple hill-climbing and min-conflicts heuristics will be less than the move cost of steepest ascent hill-climbing. But the move cost of steepest ascent hill-climbing in Figure 3.3.3.1 can be seen as the worst case for simple hill-climbing and min-conflicts heuristics. In Figure 3.3.3.2, we give the **move cost ratio** between move cost for local search in dual representation and move cost for local search in primal representation. We use the move cost in Figure 3.3.3.1 to represent the move cost of local search methods in our empirical study.

1. The move cost of local search in primal representation:

$$MV_{LP} = \sum_{i=1}^{n} (|D(v_i)|-1)\times|C|$$

2. The move cost of local search in dual representation:

$$MV_{LD} = \sum_{j=1}^{m} (|D(C_j)|-1)\times|C_{dual}|$$

3. **Move Cost Ratio** $= \dfrac{MV_{LD}}{MV_{LP}}$

Figure 3.3.3.2 Move cost ratio for local search

From Figure 3.3.3.2 we can find that the constraint tightness, domain size and constraint arity in the original given CSP are possible factors which can affect the ratio. A higher ratio means a higher move cost for local search in dual representation than in primal representation while a lower ratio means a lower move cost for local search in dual representation than in primal representation. For example, there is a CSP with 5 variables and 3 constraints. Each variable has the same domain size 10, the same constraint tightness 10%, each constraint has the same arity 3 and it has 5 dual constraints. The number of satisfied tuples in each primal constraint is (10*10*10)*10% = 100. The move cost for local search in dual representation is (100-1)*(100-1)*(100-1)*5 = 4,851,495. The move cost for local search in primal representation is (10-1)*(10-1)*(10-1) *(10-1)*(10-1)*3 = 177,147. Then the ratio will be 27.39.

In the empirical study, we use different move cost ratios to represent different problem instances. We compare the time cost of local search in both representations as the ratio increases. When move cost ratio increases in the same experiment result table, it means the move cost for local search in dual representation also increases. Thus we can find when local search in one representation is prior to another. Since the

"node visited" and "constraint" have different meaning in primal and dual representations, time cost is the most significant comparison of local search in the two representations. In this kind of comparison we use T-test (See Appendix A) to tell whether the difference between the means of time cost is significant or not.

We also compare the performance of different local search methods on the same constraint graph representation. In this situation we briefly compare the number of search nodes visited during the search period and the time cost to find a solution.

3.4 Conclusions

In this chapter we reviewed the three local search methods used in our empirical study and we also gave our approaches which focus on observing behaviours of the three local search methods in primal and dual constraint graphs.

Chapter 4

Experiment Result and Analysis

In this chapter we provide the empirical evaluation for local search methods characters on primal and dual representations presented in Chapter 3. Random CSP problem instances were generated based on a four parameter model in [Nag01] which extends the standard four parameter binary CSP model in [Smi94]. The four parameter model in [Nag01] for generating non-binary CSP is described as the following:

1. Number of variables: n
2. Size of each variable's domain: m
3. Constraint density: p_1
4. Constraint tightness: p_2

p_1 is the probability that there is a constraint among the variables in a CSP. The CSP generator used in our empirical study will generate problem instances in three problem classes. The number of constraints for each problem class was determined by the problem arity, the number of variables and the constraint density. For example, given a CSP with problem arity 5, 10 variables and constraint density $p_1 = 0.05$, the CSP generator will generate a CSP with $((10 \times 9 \times 8 \times 7 \times 6) / 5!) \times 0.05 = 12$ constraints.

In our empirical study we generate 57 problems in 3 classes given blow. Each class is given by $<n, a, m, p_1>$, where n is the number of variables, a is the problem arity, m is size of the domains and p_1 is the constraint density.

Class I: <9, 3, 10, 0.06>

Class II: <9, 3, 20, 0.06>

Class III: <12, 5, 10, 0.006>

Each class contains a set of problem instances with constraint tightness p_2 increasing from low value to higher values which indicates the problems in the class are getting easier. Such changing of constraint tightness also affects the move cost ratio from low to high which means the move cost for local search in dual representation is getting greater. Problem Class II is based on Problem Class I which enlarges the domain size from 10 to 20. Problem Class III is also based on Problem Class I but it enlarges the number of variables from 9 to 12.

Section 4.1, 4.2 and 4.3 present the results on Problem Class I, II and III. Each problem is run 100 times by each algorithm on a certain constraint graph, i.e., primal constraint graph or dual constraint graph. The following notation is used to represent the three local search algorithms on a certain constraint graph:

LPsim: simple hill-climbing on primal constraint graph

LDsim: simple hill-climbing on dual constraint graph

LPstp: steepest ascent hill-climbing on primal constraint graph

LDstp: steepest ascent hill-climbing on dual constraint graph

LPmc: min-conflicts heuristics hill-climbing on primal constraint graph

LDmc: min-conflicts heuristics hill-climbing on dual constraint graph

We also use the following notation when we measure the performance of different algorithms:

MoR: move cost ratio

Rn: number of times that local search randomly generates the initial state

Nd: number of nodes visited in the search procedure

Cc: number of constraints checked in the search procedure

Tm: CPU time cost by the search procedure to find a solution

CI: confidence interval in T-test

4.1 Experiment Results and Analysis on Class I

In this section we present experiment results based on the problem instances in Class I. This class includes 19 problems each of which has 9 variables, domain size 10, problem arity 3 and constraint density 0.06. Constraint tightness changes from 0.02 to 0.2 in steps of 0.01 which leads the move cost ratio increases from 2.11 to 22.11. A problem with smaller tightness value is a problem having tighter constraints.

4.1.1 Simple Hill-climbing on Class I

Table 4.1.1.1 is the time cost (Tm) result of simple hill-climbing on Class I on both two kinds of the constraint graphs. Here we use T-test (See Appendix A) to compare the two means of simple hill-climbing on both representations. In this thesis we launch all T-tests by a given alpha level $\alpha=0.05$, T_cv =1.645. CI is given as a 95% confidence interval on the difference of means. If |T_value| > 1.645, we reject H_0 which sets the hypothesis that the two means of the two groups have no significant difference. . If |T_value| > 1.645 and T_value is positive, we can conclude that the mean of time cost for simple hill-climbing in primal representation is greater than the mean of time cost for simple hill-climbing in dual representation, thus, LDsim generally can find a solution faster than LPsim in this problem instance. If |T_value| > 1.645 and T_value is negative, we can conclude that the mean of time cost for simple hill-climbing in primal representation is less than the mean of time cost for simple

hill-climbing in dual representation, thus, LPsim generally can find a solution faster than LDsim in this problem instance. If |T_value| < 1.645, we accept H_0 and conclude that these two means have no significant difference. The CPU time is measured on millisecond.

Move Cost Ratio	LPsim Tm Mean	LDsim Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
2.11	348.28	5.94	9.813	LPsim Mean > LDsim Mean and Reject H_0	273.96	410.71
3.22	247.03	5.47	9.787	LPsim Mean > LDsim Mean and Reject H_0	193.18	289.93
4.33	56.73	5.78	9.012	LPsim Mean > LDsim Mean and Reject H_0	39.86	62.03
5.44	42.97	6.09	8.488	LPsim Mean > LDsim Mean and Reject H_0	28.36	45.39
6.55	30.00	7.50	7.215	LPsim Mean > LDsim Mean and Reject H_0	16.38	28.61
7.66	23.75	7.35	6.708	LPsim Mean > LDsim Mean and Reject H_0	11.61	21.19
8.77	17.03	7.03	5.466	LPsim Mean > LDsim Mean and Reject H_0	6.41	13.58
9.88	16.87	6.87	5.418	LPsim Mean > LDsim Mean and Reject H_0	6.38	13.61
11.00	9.68	9.06	0.462	Accept H_0	-2.01	3.24
12.11	9.38	8.43	0.741	Accept H_0	-1.56	3.46
13.22	7.04	10.14	-2.507	LPsim Mean < LDsim Mean and Reject H_0	-5.52	-0.67
14.33	6.72	10.00	-2.688	LPsim Mean < LDsim Mean and Reject H_0	-5.67	-0.88
15.44	5.94	8.75	-2.624	LPsim Mean < LDsim Mean and Reject H_0	-4.91	-0.71
16.55	4.38	8.75	-4.411	LPsim Mean < LDsim Mean and Reject H_0	-6.31	-2.42
17.66	4.21	10.93	-5.761	LPsim Mean < LDsim Mean and Reject H_0	-9.01	-4.43
18.77	4.06	9.68	-5.335	LPsim Mean < LDsim Mean and Reject H_0	-7.68	-3.55
19.88	3.90	9.85	-5.601	LPsim Mean < LDsim Mean and Reject H_0	-8.03	-3.86
21.00	2.97	10.47	-6.911	LPsim Mean < LDsim Mean and Reject H_0	-9.62	-5.37
22.11	2.97	10.94	-7.028	LPsim Mean < LDsim Mean and Reject H_0	-10.19	-5.74

Table 4.1.1.1 Time Cost of Simple Hill-climbing on Class I

The result of Table 4.1.1.1 is also presented as a graph in Figure 4.1.1.1. From this figure we can see that simple hill-climbing can find a solution faster in the dual representation when move cost ratio is low (MoR from 2.11 to 9.88). As the move cost ratio is increasing which indicates the problem is getting looser, simple hill-climbing will get a better performance in the primal representation (MoR from 13.22 to 22.11) than in the dual representation. The value on Y axis is given in logarithmic scale.

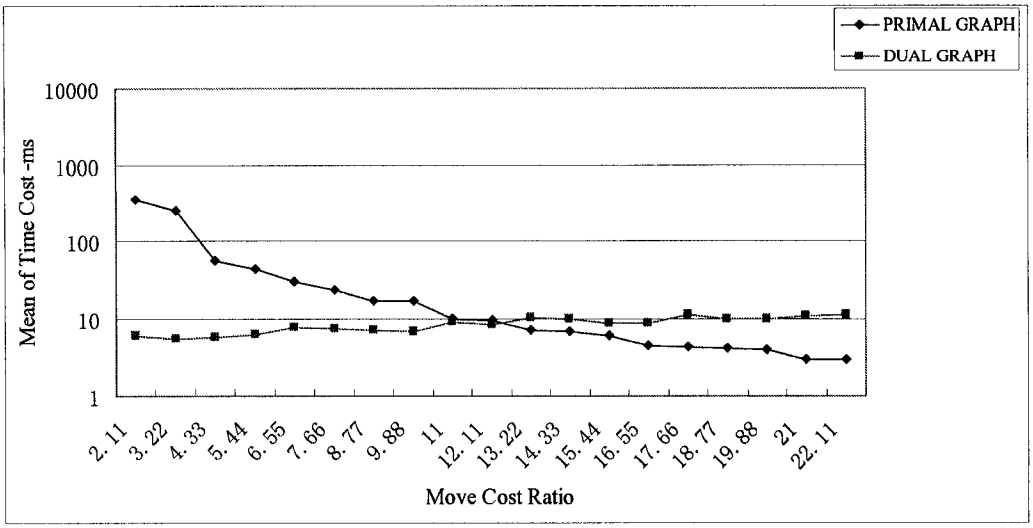


Figure 4.1.1.1 Comparison of Tm Mean of Simple Hill-climbing on Class I

Move Cost Ratio	LPsim Rn	LPsim Nd	LPsim Cc	LDsim Rn	LDsim Nd	LDsim Cc
2.11	117.41	16116.03	80580.15	4.59	968.44	8715.96
3.22	77.37	11366.28	56831.40	2.63	903.49	8131.41
4.33	16.89	2596.60	12983.00	2.19	957.76	8619.84
5.44	13.03	1986.61	9933.05	1.90	1014.70	9132.30
6.55	8.85	1365.12	6825.60	2.05	1295.84	11662.56
7.66	7.24	1081.39	5406.95	1.76	1312.91	11816.19
8.77	5.16	767.06	3835.30	1.44	1222.49	11002.41
9.88	5.06	723.84	3619.20	1.39	1170.61	10535.49
11.00	3.04	419.92	2099.60	1.62	1603.32	14429.88
12.11	3.04	421.32	2106.60	1.36	1499.03	13491.27
13.22	2.29	311.65	1558.25	1.44	1806.49	16258.41
14.33	2.28	283.34	1416.70	1.32	1748.07	15732.63
15.44	1.94	242.61	1213.05	1.12	1504.60	13541.4
16.55	1.66	185.18	925.90	1.04	1516.75	13650.75
17.66	1.63	174.10	870.50	1.17	1897.88	17080.92
18.77	1.51	164.49	822.45	1.07	1734.06	15606.54
19.88	1.46	152.87	764.35	1.08	1753.19	15778.71
21.00	1.31	122.28	611.40	1.08	1811.20	16300.80
22.11	1.23	115.83	579.15	1.10	1959.58	17636.22

Table 4.1.1.2 Rn, Nd and Cc of Simple Hill-climbing on Class I

In Table 4.1.1.2 we give the other three output parameters of simple hill-climbing on Class I which includes number of times that local search randomly generates the initial state (Rn), number of nodes visited (Nd) and number of constraints checked (Cc). For those problem instances with very tight constraints, simple hill-climbing will spend a lot of time to regenerate an initial state in the primal representation, thus, it cost longer time to find a solution as more nodes are visited in the search tree and more constraints are checked. The Nc and Cc can also reflect the time cost.

4.1.2 Steepest Ascent Hill-climbing on Class I

Move Cost Ratio	LPstp Tm Mean	LDstp Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
2.11	1567.51	22.18	10.202	LPstp Mean > LDstp Mean and Reject H_0	1248.46	1842.19
3.22	307.18	9.54	9.792	LPstp Mean > LDstp Mean and Reject H_0	238.06	357.21
4.33	77.33	9.53	8.821	LPstp Mean > LDstp Mean and Reject H_0	52.73	82.86
5.44	70.00	11.25	8.229	LPstp Mean > LDstp Mean and Reject H_0	44.75	72.74
6.55	36.87	10.94	6.817	LPstp Mean > LDstp Mean and Reject H_0	18.47	33.38
7.66	29.84	8.43	6.892	LPstp Mean > LDstp Mean and Reject H_0	15.32	27.49
8.77	23.12	8.43	6.069	LPstp Mean > LDstp Mean and Reject H_0	9.94	19.43
9.88	22.19	9.53	5.326	LPstp Mean > LDstp Mean and Reject H_0	8.01	17.31
11.00	12.81	11.88	0.536	Accept H_0	-2.46	4.32
12.11	12.18	10.63	0.955	Accept H_0	-1.63	4.73
13.22	10.31	13.91	-2.101	LPstp Mean < LDstp Mean and Reject H_0	-6.95	-0.24
14.33	10.31	12.66	-1.451	Accept H_0	-5.52	0.82
15.44	9.06	11.72	-1.811	LPstp Mean < LDstp Mean and Reject H_0	-5.53	0.21
16.55	8.28	14.53	-3.811	LPstp Mean < LDstp Mean and Reject H_0	-9.46	-3.03
17.66	7.34	12.97	-3.829	LPstp Mean < LDstp Mean and Reject H_0	-8.51	-2.74
18.77	6.10	13.28	-5.006	LPstp Mean < LDstp Mean and Reject H_0	-9.99	-4.36
19.88	5.62	15.31	-6.095	LPstp Mean < LDstp Mean and Reject H_0	-12.81	-6.57
21.00	5.31	13.59	-5.809	LPstp Mean < LDstp Mean and Reject H_0	-11.07	-5.48
22.11	5.78	15.94	-6.151	LPstp Mean < LDstp Mean and Reject H_0	-13.39	-6.92

Table 4.1.2.1 Time Cost of Steepest Ascent Hill-climbing on Class I

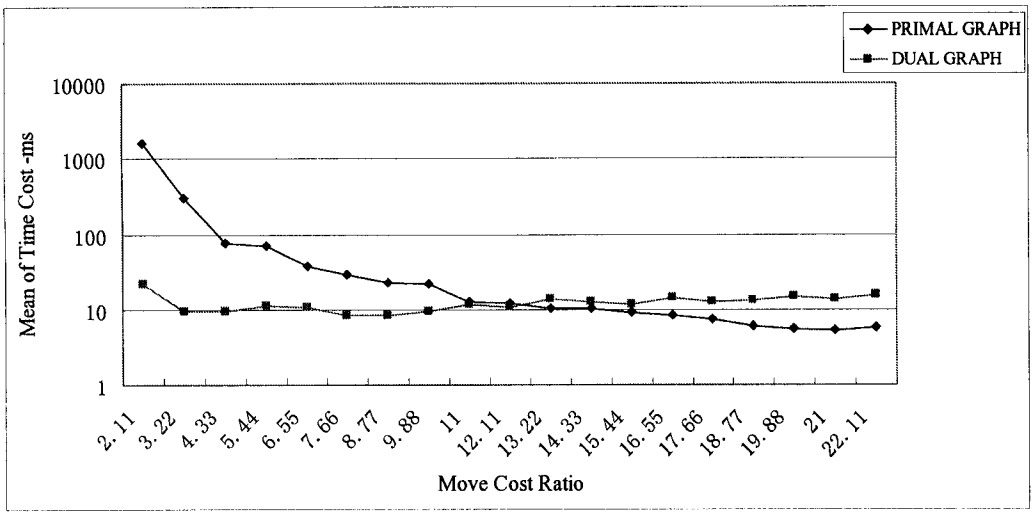


Figure 4.1.2.1 Comparison of Tm Mean of Steepest Ascent Hill-climbing on Class I

Move Cost Ratio	LPstp Rn	LPstp Nd	LPstp Cc	LDstp Rn	LDstp Nd	LDstp Cc
2.11	335.91	73377.97	366889.84	12.02	3925.98	35333.82
3.22	57.41	13835.15	69175.75	3.15	1617.67	14559.03
4.33	13.23	3554.18	17770.90	2.35	1626.57	14639.13
5.44	12.08	3228.24	16141.20	2.33	1901.65	17114.85
6.55	6.03	1707.28	8536.40	1.84	1792.47	16132.23
7.66	4.81	1339.21	6696.05	1.38	1472.06	13248.54
8.77	3.82	1060.51	5302.55	1.30	1456.58	13109.22
9.88	3.74	993.92	4969.60	1.23	1635.47	14719.23
11.00	2.21	579.12	2895.60	1.35	2013.10	18117.90
12.11	2.09	551.71	2758.60	1.23	1860.76	16746.84
13.22	1.83	468.26	2341.30	1.34	2359.98	21239.82
14.33	1.95	468.38	2341.89	1.28	2239.70	20157.30
15.44	1.66	396.15	1980.75	1.05	2018.93	18170.36
16.55	1.56	364.07	1820.35	1.08	2138.05	19242.44
17.66	1.38	313.70	1568.50	1.09	2286.00	20574.00
18.77	1.29	273.46	1367.30	1.04	2273.56	20462.03
19.88	1.25	244.25	1221.25	1.08	2662.71	23964.39
21.00	1.16	230.95	1154.75	1.02	2413.82	21724.38
22.11	1.26	250.11	1250.55	12.02	3925.98	35333.82

Table 4.1.2.2 Rn, Nd and Cc of Steepest Ascent Hill-climbing on Class I

Table 4.1.2.1 is the time cost (Tm) result of steepest ascent hill-climbing on Class I on both primal and dual constraint graphs. The result of Table 4.1.2.1 is also showed as a graph in Figure 4.1.2.1. From this figure we can see that steepest ascent hill-climbing can get a better performance in the dual representation when move cost ratio is low (MoR from 2.11 to 9.88). As the move cost ratio is increasing, steepest ascent hill-climbing will find a solution faster in the primal representation (MoR from 15.44 to 22.11) than in the dual representation. We give Rn, Nd and Cc of steepest ascent hill-climbing on Class I in Table 4.1.2.2.

4.1.3 Min-conflicts Heuristics Hill-climbing on Class I

Move Cost Ratio	LPmc Tm Mean	LDmc Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
2.11	151.16	9.37	9.037	LPmc Mean > LDmc Mean and Reject H_0	111.03	172.54
3.22	137.77	9.10	10.108	LPmc Mean > LDmc Mean and Reject H_0	103.72	153.61
4.33	121.87	5.14	10.156	LPmc Mean > LDmc Mean and Reject H_0	94.21	139.25
5.44	76.10	6.57	9.308	LPmc Mean > LDmc Mean and Reject H_0	54.88	84.17
6.55	53.28	7.05	8.745	LPmc Mean > LDmc Mean and Reject H_0	35.86	56.59
7.66	29.84	7.04	7.534	LPmc Mean > LDmc Mean and Reject H_0	16.86	28.73
8.77	22.97	7.19	6.528	LPmc Mean > LDmc Mean and Reject H_0	11.04	20.51
9.88	13.28	7.81	3.548	LPmc Mean > LDmc Mean and Reject H_0	2.44	8.49
11.00	10.16	8.28	1.431	Accept H_0	-0.69	4.45
12.11	9.38	8.52	0.625	Accept H_0	-1.82	3.53
13.22	7.34	9.38	-1.621	Accept H_0	-4.51	0.42
14.33	7.35	9.84	-1.821	LPmc Mean < LDmc Mean and Reject H_0	-5.17	0.19
15.44	4.69	10.00	-4.342	LPmc Mean < LDmc Mean and Reject H_0	-7.71	-2.91
16.55	5.15	10.16	-4.114	LPmc Mean < LDmc Mean and Reject H_0	-.39	-2.62
17.66	3.59	12.19	-6.017	LPmc Mean < LDmc Mean and Reject H_0	-11.41	-5.79
18.77	3.43	11.88	-7.115	LPmc Mean < LDmc Mean and Reject H_0	-10.77	-6.12
19.88	3.28	11.41	-5.854	LPmc Mean < LDmc Mean and Reject H_0	-10.85	-5.41
21.00	3.13	13.91	-8.029	LPmc Mean < LDmc Mean and Reject H_0	-13.41	-8.14
22.11	2.65	12.97	-9.222	LPmc Mean < LDmc Mean and Reject H_0	-12.51	-8.12

Table 4.1.3.1 Time Cost of Min-conflicts Heuristics Hill-climbing on Class I

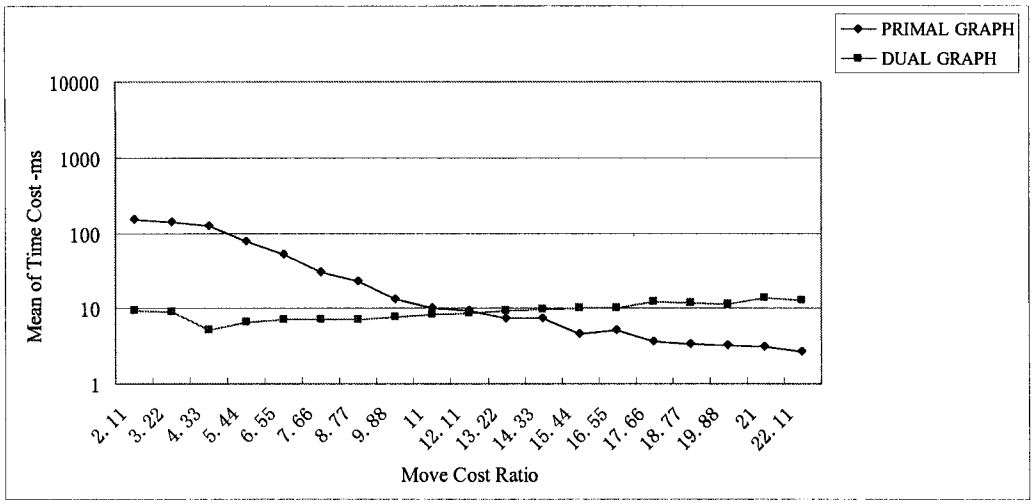


Figure 4.1.3.1 Comparison of Tm Mean of Min-conflicts Heuristics Hill-climbing on Class I

Move Cost Ratio	LPmc Rn	LPmc Nd	LPmc Cc	LDmc Rn	LDmc Nd	LDmc Cc
2.11	335.91	73377.97	366889.84	12.02	3925.98	35333.82
3.22	57.41	13835.15	69175.75	3.15	1617.67	14559.03
4.33	13.23	3554.18	17770.90	2.35	1626.57	14639.13
5.44	12.08	3228.24	16141.20	2.33	1901.65	17114.85
6.55	6.03	1707.28	8536.40	1.84	1792.47	16132.23
7.66	4.81	1339.21	6696.05	1.38	1472.06	13248.54
8.77	3.82	1060.51	5302.55	1.30	1456.58	13109.22
9.88	3.74	993.92	4969.60	1.23	1635.47	14719.23
11.00	2.21	579.12	2895.60	1.35	2013.10	18117.90
12.11	2.09	551.71	2758.60	1.23	1860.76	16746.84
13.22	1.83	468.26	2341.30	1.34	2359.98	21239.82
14.33	1.95	468.38	2341.89	1.28	2239.70	20157.30
15.44	1.66	396.15	1980.75	1.05	2018.93	18170.36
16.55	1.56	364.07	1820.35	1.08	2138.05	19242.44
17.66	1.38	313.70	1568.50	1.09	2286.00	20574.00
18.77	1.29	273.46	1367.30	1.04	2273.56	20462.03
19.88	1.25	244.25	1221.25	1.08	2662.71	23964.39
21.00	1.16	230.95	1154.75	1.02	2413.82	21724.38
22.11	1.26	250.11	1250.55	12.02	3925.98	35333.82

Table 4.1.3.2 Rn, Nd and Cc of Min-conflicts Heuristics Hill-climbing on Class I

Table 4.1.3.1 is the time cost (Tm) result of min-conflicts heuristics hill-climbing on Class I on both primal and dual constraint graphs. The result of Table 4.1.3.1 is also showed as a graph in Figure 4.1.3.1. From this figure we can see that min-conflicts heuristics hill-climbing can get a better performance in the dual representation when move cost ratio is low (MoR from 2.11 to 9.88). As the move cost ratio is increasing, min-conflicts heuristics hill-climbing will find a solution faster in the primal representation (MoR from 14.33 to 22.11) than in the dual representation. We give Rn, Nd and Cc of min-conflicts heuristics hill-climbing on Class I in Table 4.1.3.2.

4.1.4 Comparisons among Different Hill-climbing algorithms on Class I

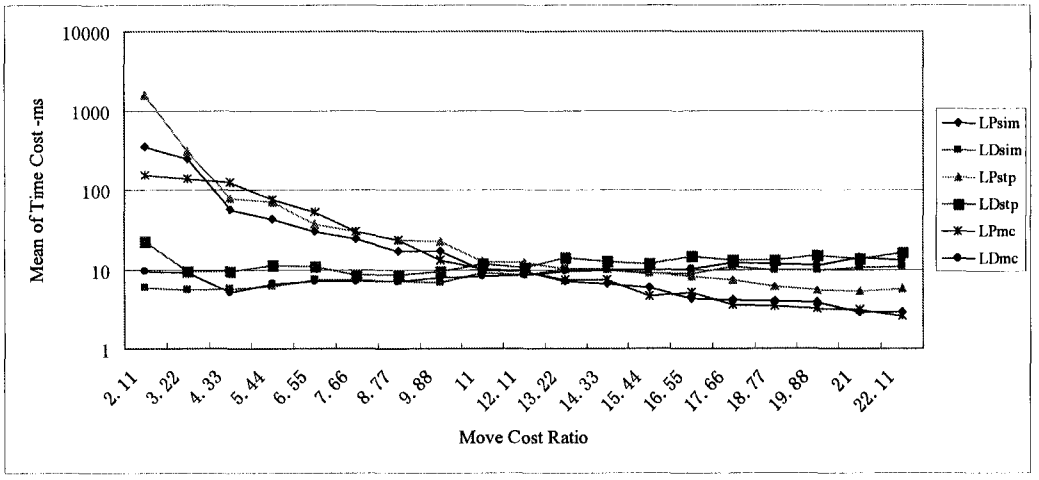


Figure 4.1.4.1 Comparisons of Tm Means for LPsim, LPstp, LPmc, LDsim, LDstp and LDmc on Class I

Based on Table 4.1.1.1, Table 4.1.2.1 and Table 4.1.3.1, we give Figure 4.1.4.1 to compare the Tm means of all the three algorithms on Class I. From Figure 4.1.4.1 we can see that the three hill-climbing methods have similar performance on both primal and dual representations. But steepest ascent hill-climbing does not perform so well as simple hill-climbing and min-conflicts heuristics hill-climbing on both primal and

dual representations that such characters is also showed from Figure 4.1.4.2 and Figure 4.1.4.3. Figure 4.1.4.2 and Figure 4.1.4.3 which compare Nc among these three algorithms on both representations are based on Table 4.1.1.2, Table 4.1.2.2 and Table 4.1.3.2. In Figure 4.1.4.1, 4.1.4.2 and 4.1.4.3, min-conflicts heuristics hill-climbing suggests that it have the best performance among the three algorithms.

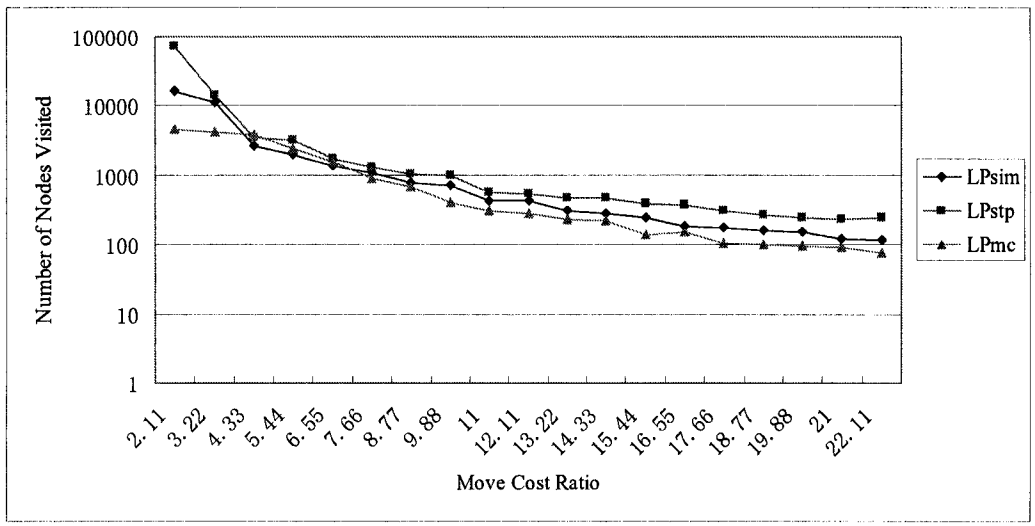


Figure 4.1.4.2 Comparisons of Nc for LPsim, LPstp and LPmc on Class I

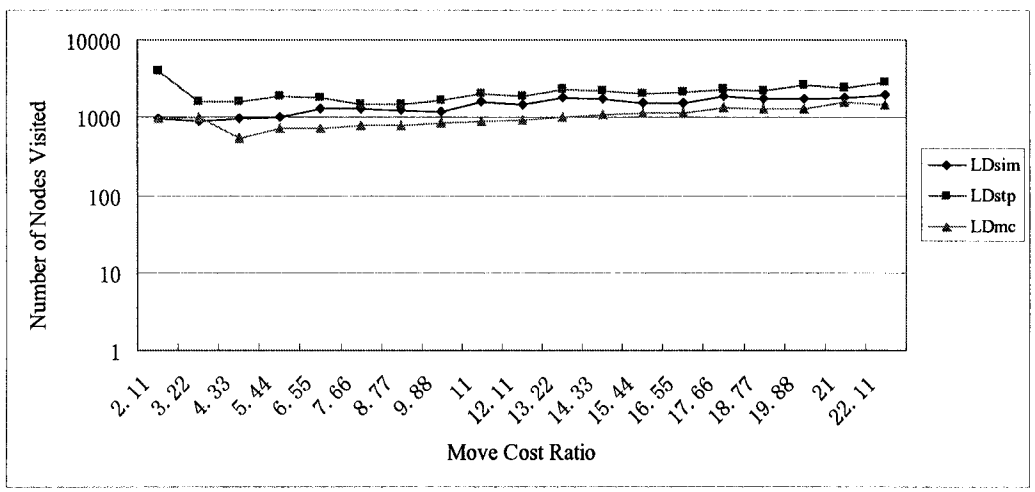


Figure 4.1.4.3 Comparisons of Nc for LDsim, LDstp and LDmc on Class I

4.2 Experiment Results and Analysis on Class II

In this section we present experiment results based on the problem instances in Class II. This class includes 17 problems each of which has 9 variables, domain size 20, problem arity 3 and constraint density 0.06. Constraint tightness changes from 0.0075 to 0.03 in steps of 0.0025, from 0.03 to 0.1 in steps of 0.01.The move cost ratio increases from 3.10 to 42.05. Class II is based on Class I but it enlarges the number of domain size from 10 to 20.

4.2.1 Simple Hill-climbing on Class II

Move Cost Ratio	LPsim Tm Mean	LDsim Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
3.10	2790.05	100.01	11.031	LPsim Mean > LDsim Mean and Reject H_0	2212.03	3168.04
4.15	1249.37	62.96	9.611	LPsim Mean > LDsim Mean and Reject H_0	944.46	1428.36
5.21	1114.21	48.59	9.735	LPsim Mean > LDsim Mean and Reject H_0	851.09	1280.14
6.26	369.84	43.28	8.788	LPsim Mean > LDsim Mean and Reject H_0	253.72	399.39
7.31	322.19	35.93	8.837	LPsim Mean > LDsim Mean and Reject H_0	222.77	349.74
8.36	222.50	30.15	8.602	LPsim Mean > LDsim Mean and Reject H_0	148.52	236.17
9.42	147.17	33.59	7.538	LPsim Mean > LDsim Mean and Reject H_0	84.04	143.11
10.47	122.50	38.59	6.492	LPsim Mean > LDsim Mean and Reject H_0	58.57	109.24
11.52	116.10	45.63	5.752	LPsim Mean > LDsim Mean and Reject H_0	46.46	94.47
12.57	66.08	31.25	4.791	LPsim Mean > LDsim Mean and Reject H_0	20.57	49.08
16.78	37.81	30.31	1.572	Accept H_0	-1.84	16.84
21.00	22.66	35.00	-3.014	LPsim Mean < LDsim Mean and Reject H_0	-20.36	-4.31
25.21	17.65	40.05	-5.277	LPsim Mean < LDsim Mean and Reject H_0	-30.71	-14.08
29.42	13.90	38.13	-6.168	LPsim Mean < LDsim Mean and Reject H_0	-31.92	-16.53
33.63	11.88	51.41	-7.737	LPsim Mean < LDsim Mean and Reject H_0	-49.54	-29.51
37.84	11.87	52.82	-7.852	LPsim Mean < LDsim Mean and Reject H_0	-51.17	-30.72
42.05	9.06	50.47	-8.362	LPsim Mean < LDsim Mean and Reject H_0	-51.11	-31.71

Table 4.2.1.1 Time Cost of Simple Hill-climbing on Class II

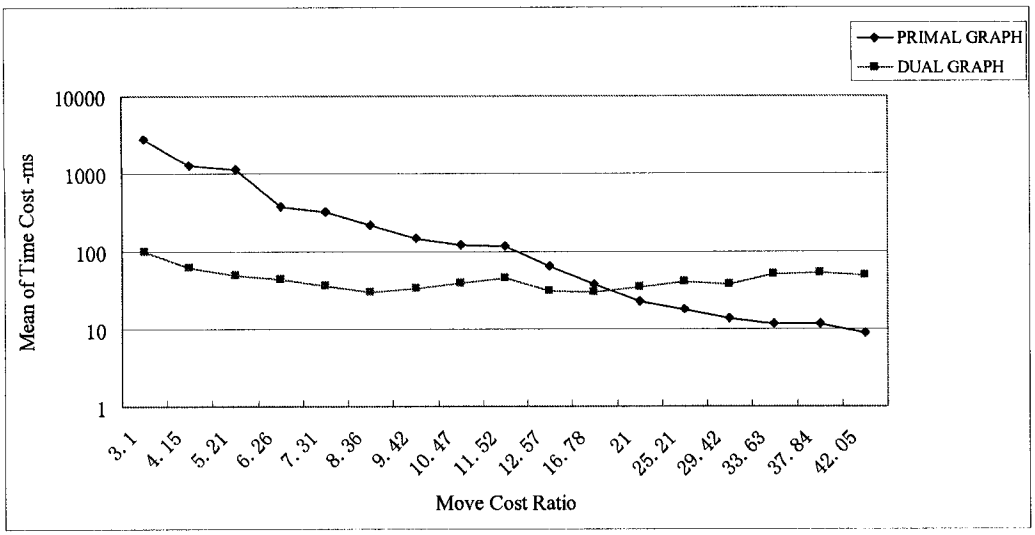


Figure 4.2.1.1 Comparison of Tm Mean of Simple Hill-climbing on Class II

Move Cost Ratio	LPsim Rn	LPsim Nd	LPsim Cc	LDsim Rn	LDsim Nd	LDsim Cc
3.10	475.43	128582.08	642910.44	21.90	16957.69	152619.2
4.15	199.09	57851.64	289258.16	10.22	10367.76	93309.93
5.21	172.99	51667.53	258337.64	6.66	8846.21	79615.89
6.26	55.45	17233.10	86165.50	5.32	7903.72	71133.47
7.31	47.31	14982.82	74914.10	3.61	6311.84	56806.56
8.36	32.34	10419.11	52095.55	2.77	5462.39	49161.51
9.42	20.29	6531.80	32659.03	2.73	6101.02	54909.18
10.47	17.42	5670.87	28354.35	2.87	6955.54	62599.86
11.52	16.71	5396.73	26983.65	2.90	7853.72	70683.47
12.57	9.47	3061.06	15305.30	1.92	5539.44	49854.96
16.78	5.39	1732.79	8663.95	1.56	5463.11	49167.99
21.00	3.20	1011.59	5057.95	1.38	6299.44	56694.96
25.21	2.63	794.15	3970.75	1.30	7097.38	63876.42
29.42	2.10	622.03	3110.15	1.21	6876.21	61885.89
33.63	1.86	497.50	2487.50	1.20	8489.26	76403.34
37.84	1.93	529.90	2649.50	1.20	9171.98	82547.82
42.05	1.62	405.44	2027.2	1.10	9004.26	81038.43

Table 4.2.1.2 Rn, Nd and Cc of Simple Hill-climbing on Class II

Table 4.2.1.1 is the time cost (Tm) result of simple hill-climbing on Class II on both two kinds of the constraint graphs. The result of Table 4.2.1.1 is also presented as a graph in Figure 4.2.1.1. From this figure we can see that simple hill-climbing can find a solution faster in the dual representation when move cost ratio is low (MoR from 3.10 to 12.57). As the move cost ratio is increasing which indicates the problem is getting looser, simple hill-climbing will get a better performance in the primal representation (MoR from 21.00 to 42.05).

We give Rn, Nd and Cc of simple hill-climbing on Class II in Table 4.2.1.2. As the same situation in Class I, for those problem instances with very tight constraints, simple hill-climbing will spend a lot of time to regenerate an initial state in the primal representation, thus, it cost longer time to find a solution as more nodes are visited in the search tree and more constraints are checked.

4.2.2 Steepest Ascent Hill-climbing on Class II

Table 4.2.2.1 is the time cost (Tm) result of steepest ascent hill-climbing on Class II on both primal and dual constraint graphs. The result of Table 4.2.2.1 is also showed as a graph in Figure 4.2.2.1. From this figure we can see that steepest ascent hill-climbing can get a better performance in the dual representation when move cost ratio is low (MoR from 3.10 to 16.78). As the move cost ratio is increasing, steepest ascent hill-climbing will find a solution faster in the primal representation (MoR from 25.21 to 42.05) than in the dual representation. We give Rn, Nd and Cc of steepest ascent hill-climbing on Class II in Table 4.2.2.2.

Move Cost Ratio	LPstp Tm Mean	LDstp Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
3.10	3195.47	94.68	9.157	LPstp Mean > LDstp Mean and Reject H_0	2437.15	3764.42
4.15	3206.58	106.72	9.505	LPstp Mean > LDstp Mean and Reject H_0	2460.68	3739.03
5.21	1114.21	48.59	9.735	LPstp Mean > LDstp Mean and Reject H_0	851.09	1280.14
6.26	749.69	62.19	9.147	LPstp Mean > LDstp Mean and Reject H_0	540.18	834.81
7.31	749.53	67.66	9.065	LPstp Mean > LDstp Mean and Reject H_0	534.45	829.28
8.36	337.03	43.13	8.638	LPstp Mean > LDstp Mean and Reject H_0	227.21	360.58
9.42	313.91	55.15	8.129	LPstp Mean > LDstp Mean and Reject H_0	196.36	321.15
10.47	174.53	50.78	6.822	LPstp Mean > LDstp Mean and Reject H_0	88.21	159.29
11.52	179.85	47.65	6.833	LPstp Mean > LDstp Mean and Reject H_0	94.28	170.11
12.57	119.37	51.56	6.229	LPstp Mean > LDstp Mean and Reject H_0	46.47	89.14
16.78	82.97	45.79	3.947	LPstp Mean > LDstp Mean and Reject H_0	18.71	55.64
21.00	48.91	44.06	0.497	Accept H_0	-14.25	23.95
25.21	35.62	56.40	-3.182	LPstp Mean < LDstp Mean and Reject H_0	-33.57	-7.98
29.42	26.41	55.63	-4.901	LPstp Mean < LDstp Mean and Reject H_0	-40.91	-17.53
33.63	20.93	61.72	-6.491	LPstp Mean < LDstp Mean and Reject H_0	-53.11	-28.47
37.84	20.32	66.25	-6.887	LPstp Mean < LDstp Mean and Reject H_0	-59.00	-32.85
42.05	15.78	65.94	-7.738	LPstp Mean < LDstp Mean and Reject H_0	-62.86	-37.45

Table 4.2.2.1 Time Cost of Steepest Ascent Hill-climbing on Class II

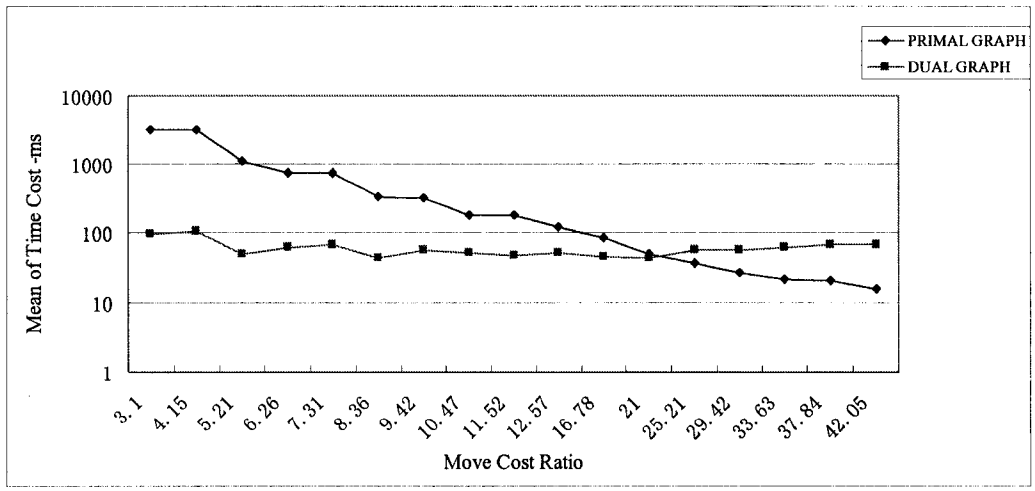


Figure 4.2.2.1 Comparison of Tm Mean of Steepest Ascent Hill-climbing on Class II

Move Cost Ratio	LPstp Rn	LPstp Nd	LPstp Cc	LDstp Rn	LDstp Nd	LDstp Cc
3.10	368.09	150221.52	751107.75	14.72	16568.73	149118.56
4.15	331.01	149641.73	748208.94	12.07	18570.46	167134.14
5.21	172.99	51667.53	258337.64	6.66	8846.21	79615.89
6.26	66.21	34812.32	174061.60	4.92	11183.71	100653.39
7.31	63.21	34898.89	174494.45	3.91	10174.53	91570.77
8.36	27.80	15773.04	78865.20	2.68	7745.12	69706.08
9.42	25.13	14685.85	73429.25	2.90	9697.53	87277.77
10.47	13.48	8132.25	40661.25	2.45	8847.90	79631.10
11.52	13.52	8254.26	41271.30	2.16	8398.32	75584.88
12.57	8.84	5497.38	27486.90	2.15	9321.73	83895.57
16.78	6.02	3826.39	19131.94	1.47	7877.93	70901.36
21.00	3.43	2180.78	10903.90	1.25	7683.34	69150.06
25.21	2.52	1604.37	8021.85	1.33	9876.71	88890.39
29.42	2.03	1209.06	6045.35	1.14	9661.48	86953.32
33.63	1.62	935.39	4676.95	1.15	10886.97	97982.73
37.84	1.62	891.11	4455.54	1.13	11926.19	107335.71
42.05	1.35	718.80	3594.00	1.05	11681.23	105131.07

Table 4.2.2.2 Rn, Nd and Cc of Steepest Ascent Hill-climbing on Class II

4.2.3 Min-conflicts Heuristics Hill-climbing on Class II

Table 4.2.3.1 is the time cost (Tm) result of min-conflicts heuristics on Class II on both primal and dual constraint graphs. The result of Table 4.2.3.1 is also showed as a graph in Figure 4.2.3.1. From this figure we can see that min-conflicts heuristics hill-climbing can get a better performance in the dual representation when move cost ratio is low (MoR from 3.10 to 16.78). As the move cost ratio is increasing, min-conflicts heuristics hill-climbing will find a solution faster in the primal representation (MoR from 25.21 to 42.05) than in the dual representation. We give Rn, Nd and Cc of min-conflicts heuristics hill-climbing on Class III in Table 4.2.3.2.

Move Cost Ratio	LPmc Tm Mean	LDmc Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
3.10	316.29	30.18	9.127	LPmc Mean > LDmc Mean and Reject H_0	224.67	347.54
4.15	289.22	26.27	9.552	LPmc Mean > LDmc Mean and Reject H_0	208.99	316.91
5.21	300.60	52.64	8.316	LPmc Mean > LDmc Mean and Reject H_0	189.52	306.39
6.26	273.44	25.32	9.129	LPmc Mean > LDmc Mean and Reject H_0	194.85	301.38
7.31	251.70	21.22	9.176	LPmc Mean > LDmc Mean and Reject H_0	181.25	279.71
8.36	227.20	28.45	8.721	LPmc Mean > LDmc Mean and Reject H_0	154.08	243.41
9.42	201.89	25.78	8.689	LPmc Mean > LDmc Mean and Reject H_0	136.38	215.83
10.47	142.03	29.06	7.871	LPmc Mean > LDmc Mean and Reject H_0	84.83	141.12
11.52	101.72	28.75	7.036	LPmc Mean > LDmc Mean and Reject H_0	52.64	93.29
12.57	86.25	33.91	5.688	LPmc Mean > LDmc Mean and Reject H_0	34.31	70.37
16.78	51.72	34.68	2.994	LPmc Mean > LDmc Mean and Reject H_0	5.88	28.19
21.00	33.75	37.50	-0.922	Accept H_0	-11.71	4.21
25.21	20.94	40.78	-5.372	LPmc Mean < LDmc Mean and Reject H_0	-27.07	-12.61
29.42	20.95	46.72	-6.262	LPmc Mean < LDmc Mean and Reject H_0	-33.83	-17.71
33.63	16.56	47.66	-8.541	LPmc Mean < LDmc Mean and Reject H_0	-38.23	-23.96
37.84	11.41	52.81	-10.249	LPmc Mean < LDmc Mean and Reject H_0	-49.31	-33.48
42.05	8.60	59.22	-11.791	LPmc Mean < LDmc Mean and Reject H_0	-59.03	-42.21

Table 4.2.3.1 Time Cost of Min-conflicts Heuristics Hill-climbing on Class II

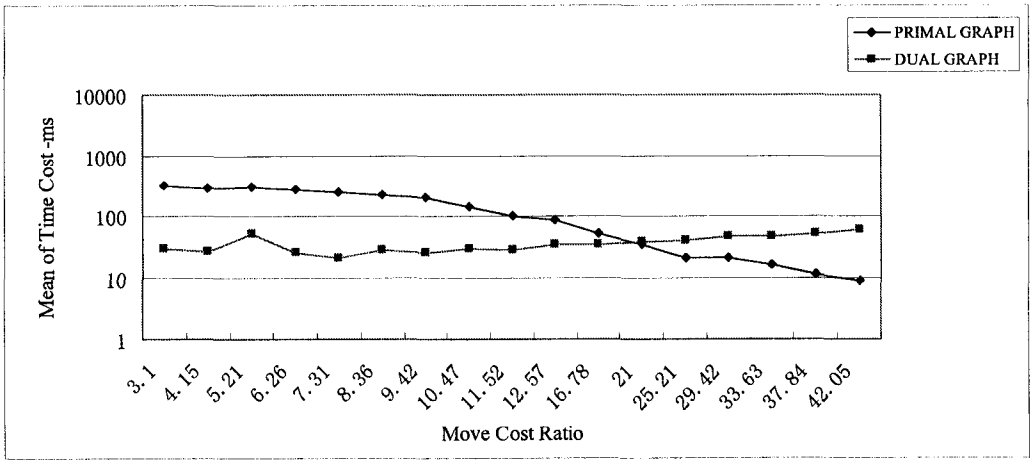


Figure 4.2.3.1 Comparison of Tm Mean of Min-conflicts Heuristics Hill-climbing on Class II

Move Cost Ratio	LPmc Rn	LPmc Nd	LPmc Cc	LDmc Rn	LDmc Nd	LDmc Cc
3.10	48.29	9719.92	48599.60	9.07	3393.21	30538.89
4.15	49.98	8765.79	43828.95	5.59	3055.06	27495.54
5.21	56.28	9435.97	7179.85	7.88	5517.62	49658.58
6.26	55.86	8638.21	43191.05	3.28	2871.23	25841.08
7.31	52.95	7935.67	39678.35	2.53	2463.49	22171.45
8.36	48.11	7091.12	35455.60	2.83	3310.54	29794.86
9.42	43.86	6405.69	32028.45	2.54	2867.86	25810.74
10.47	32.20	4569.75	22848.75	2.58	3276.73	29490.57
11.52	23.55	3272.35	16361.75	2.30	3255.74	29301.66
12.57	19.26	2735.88	13679.40	2.50	3894.00	35046.00
16.78	12.42	1669.74	8348.70	2.02	4007.50	36067.50
21.00	8.49	1087.09	5435.45	1.81	4277.69	38499.21
25.21	5.36	675.23	3376.15	1.65	4659.31	41933.88
29.42	5.35	653.08	3265.40	1.63	5209.89	46889.01
33.63	4.21	493.38	2466.89	1.64	5451.34	49062.06
37.84	3.35	368.36	1841.80	1.55	6013.13	54118.17
42.05	2.47	261.78	1308.90	1.48	6684.93	60164.37

Table 4.2.3.2 Rn, Nd and Cc of Min-conflicts Heuristics Hill-climbing on Class II

4.2.4 Comparisons among Different Hill-climbing algorithms on Class II

Based on Table 4.2.1.1, Table 4.2.2.1 and Table 4.2.3.1, we give Figure 4.2.4.1 to compare the Tm means of all the three algorithms on Class II. Figure 4.2.4.1 shows three hill-climbing methods have similar characters as they are in Class I. Steepest ascent hill-climbing does not perform so well as simple hill-climbing and min-conflicts heuristics hill-climbing on both primal and dual representations (See Figure 4.2.4.2 and Figure 4.2.4.3). These two graphs comparing Nc among the three algorithms on both representations are based on Table 4.2.1.2, Table 4.2.2.2 and Table 4.2.3.2. In Figure 4.2.4.1, 4.2.4.2 and 4.2.4.3, min-conflicts heuristics hill-climbing suggests that it have the best performance among the three algorithms.

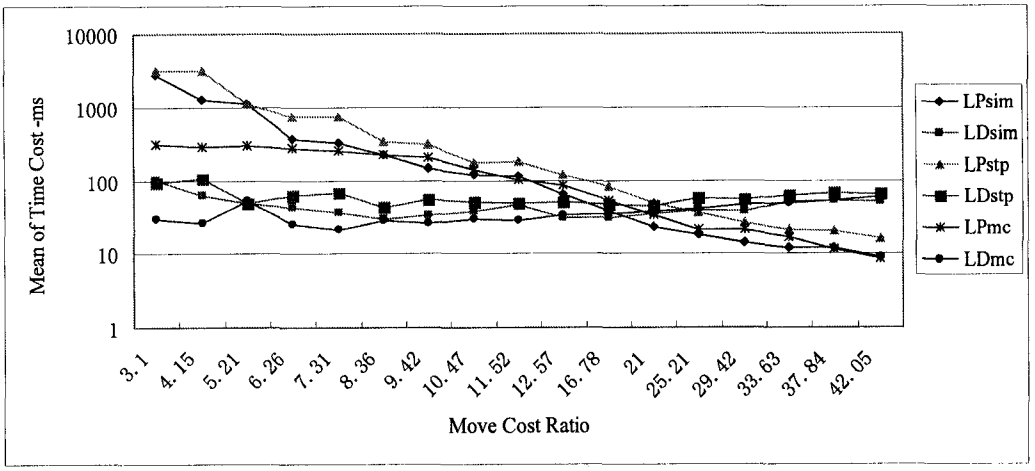


Figure 4.2.4.1 Comparisons of Tm Means for LPsim, LPstp, LPmc, LDsim, LDstp and LDmc on Class II

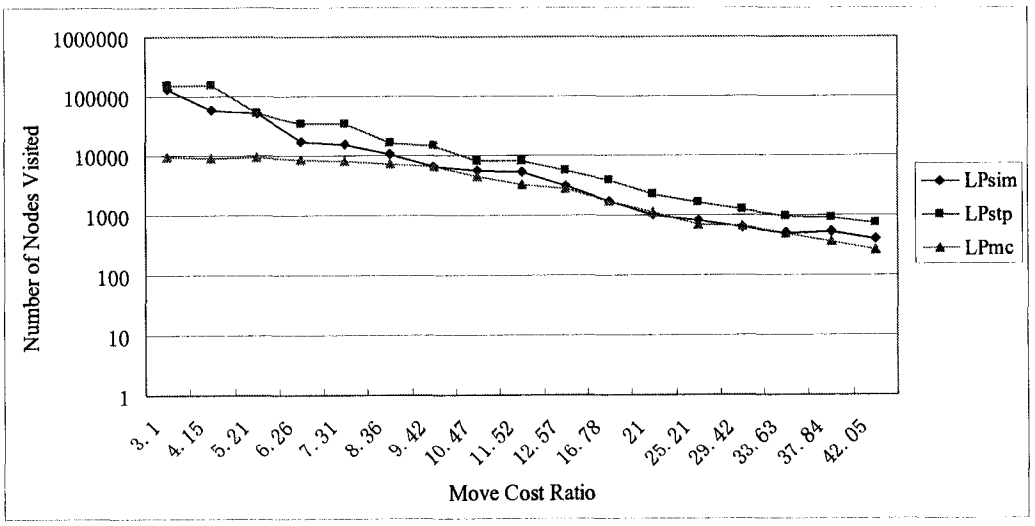


Figure 4.2.4.2 Comparisons of Nc for LPsim, LPstp and LPmc on Class II

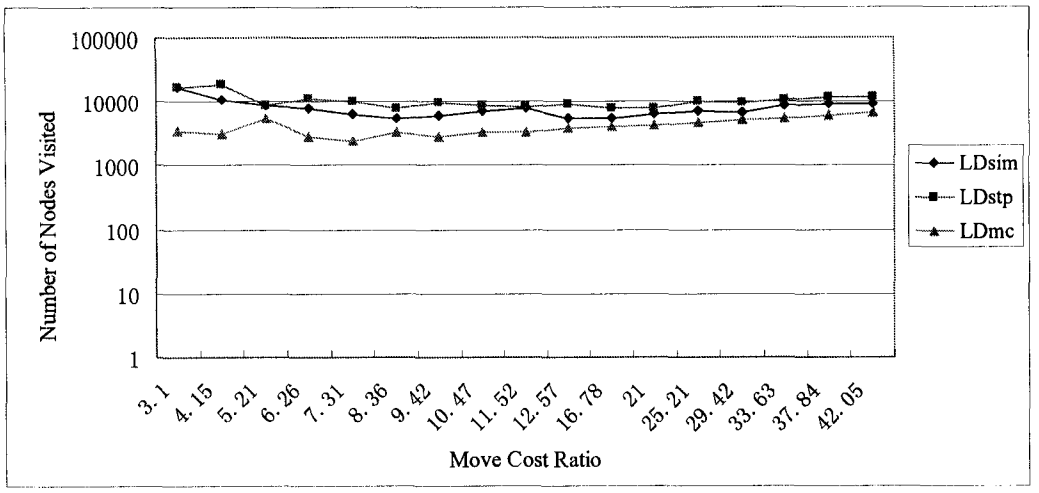


Figure 4.2.4.3 Comparisons of Nc for LDsim, LDstp and LDmc on Class II

4.3 Experiment Results and Analysis on Class III

In this section we present experiment results based on the problem instances in Class III. This class includes 21 problems each of which has 12 variables, domain size 10, problem arity 5 and constraint density 0.006. Constraint tightness changes from 0.01 to 0.02 in steps of 0.01, from 0.02 to 0.025 in steps of 0.0025, from 0.03 to 0.1 in steps of 0.01.The move cost ratio increases from 26.58 to 266.75. Class III is based on Class I but it enlarges the number of variables from 9 to 12 and arity from 3 to 5.

4.3.1 Simple Hill-climbing on Class III

Table 4.3.1.1 is the time cost (Tm) result of simple hill-climbing on Class III on both two kinds of the constraint graphs. The result of Table 4.3.1.1 is also presented as a graph in Figure 4.3.1.1. From this figure we can see that simple hill-climbing can find a solution faster in the dual representation when move cost ratio is low (MoR from 26.58 to 66.61). As the move cost ratio is increasing which indicates the problem is

getting looser, simple hill-climbing will get a better performance in the primal representation (MoR from 106.63 to 266.75) than in the dual representation.

Move Cost Ratio	LPsim Tm Mean	LDsim Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
26.58	2124.86	123.91	9.889	LPsim Mean > LDsim Mean and Reject H_0	1604.39	2397.51
29.25	2247.32	119.53	10.040	LPsim Mean > LDsim Mean and Reject H_0	1712.43	2543.14
31.92	1359.38	118.59	9.271	LPsim Mean > LDsim Mean and Reject H_0	978.47	1503.11
34.58	1425.47	127.50	9.147	LPsim Mean > LDsim Mean and Reject H_0	1019.84	1576.09
37.25	1476.25	128.75	9.189	LPsim Mean > LDsim Mean and Reject H_0	1060.11	1634.89
39.92	982.66	128.91	8.696	LPsim Mean > LDsim Mean and Reject H_0	661.33	1046.16
42.59	864.06	99.37	8.775	LPsim Mean > LDsim Mean and Reject H_0	593.89	935.48
45.26	872.35	135.46	8.394	LPsim Mean > LDsim Mean and Reject H_0	564.82	908.950
47.93	506.40	141.39	6.941	LPsim Mean > LDsim Mean and Reject H_0	261.94	468.07
50.60	490.47	132.03	6.981	LPsim Mean > LDsim Mean and Reject H_0	257.81	459.06
53.26	459.22	138.91	6.715	LPsim Mean > LDsim Mean and Reject H_0	226.81	413.81
59.92	303.59	122.18	5.613	LPsim Mean > LDsim Mean and Reject H_0	118.06	244.75
66.61	282.50	147.64	4.282	LPsim Mean > LDsim Mean and Reject H_0	73.14	196.57
79.95	163.28	158.91	0.195	Accept H_0	-39.51	48.2
106.63	77.18	189.22	-5.656	LPsim Mean < LDsim Mean and Reject H_0	-150.86	-73.21
133.32	46.09	232.98	-8.206	LPsim Mean < LDsim Mean and Reject H_0	-231.52	-142.25
160.00	28.75	255.93	-9.215	LPsim Mean < LDsim Mean and Reject H_0	-275.49	-178.86
186.69	23.91	280.16	-9.529	LPsim Mean < LDsim Mean and Reject H_0	-203.54	-308.95
213.37	17.50	341.57	-9.898	LPsim Mean < LDsim Mean and Reject H_0	-388.23	-259.91
240.06	15.00	398.91	-10.045	LPsim Mean < LDsim Mean and Reject H_0	-458.81	-309.01
266.75	11.25	426.72	-10.168	LPsim Mean < LDsim Mean and Reject H_0	-495.55	-335.38

Table 4.3.1.1 Time Cost of Simple Hill-climbing on Class III

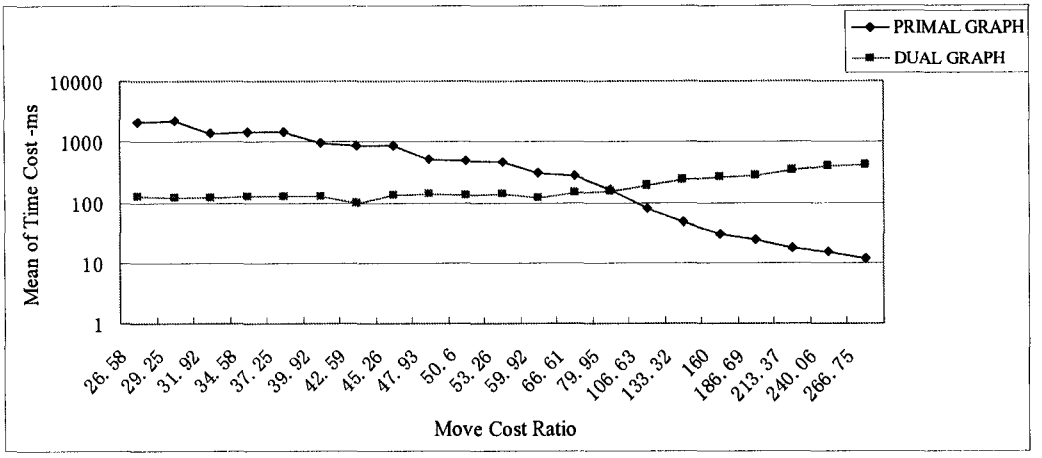


Figure 4.3.1.1 Comparison of Tm Mean of Simple Hill-climbing on Class III

Move Cost Ratio	LPsim Rn	LPsim Nd	LPsim Cc	LDsim Rn	LDsim Nd	LDsim Cc
26.58	405.53	70803.38	354016.88	3.51	19086.69	209953.60
29.25	422.68	74995.75	374978.80	3.08	18278.88	201067.69
31.92	244.77	45028.40	225141.95	2.76	18227.86	200506.60
34.58	256.78	47294.92	236474.64	2.83	20011.91	220130.95
37.25	260.60	49105.26	245526.30	2.59	19785.86	217644.69
39.92	172.56	32716.20	163581.00	2.28	18621.27	204834.05
42.59	148.50	28687.06	143435.30	1.78	15237.18	167608.98
45.26	151.28	29081.20	145406.00	2.13	20854.36	229397.88
47.93	86.17	16845.92	84229.60	2.15	21455.84	236014.23
50.60	82.02	16271.24	81356.20	1.89	19958.81	219546.92
53.26	77.24	15283.55	76417.75	1.93	21541.56	236957.12
59.92	49.64	10056.69	50283.45	1.83	18571.44	204285.94
66.61	45.75	9361.16	46805.80	1.66	23247.17	255718.80
79.95	25.99	5405.01	27025.05	1.42	24240.04	266640.53
106.63	12.33	2541.69	12708.45	1.35	29216.86	321385.40
133.32	7.21	1505.43	7527.15	1.29	36396.42	400360.62
160.00	4.48	920.99	4604.95	1.20	40011.86	440130.40
186.69	3.89	782.15	3910.75	1.14	43114.69	474261.47
213.37	2.91	563.53	2817.70	1.17	53121.55	584337.10
240.06	2.59	483.69	2418.45	1.17	60105.34	661158.75
266.75	2.02	361.63	1808.15	1.19	65107.29	716180.06

Table 4.3.1.2 Rn, Nd and Cc of Simple Hill-climbing on Class HI

We give Rn, Nd and Cc of simple hill-climbing on Class II in Table 4.3.1.2. As the same situations in Class I and Class II, for those problem instances with very tight constraints, simple hill-climbing will spend a lot of time to regenerate an initial state in the primal representation, thus, it cost longer time to find a solution as more nodes are visited in the search tree and more constraints are checked.

4.3.2 Steepest Ascent Hill-climbing on Class III

Move Cost Ratio	LPstp Tm Mean	LDstp Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
26.58	3327.02	167.03	9.537	LPstp Mean > LDstp Mean and Reject H_0	2510.59	3809.38
29.25	2867.67	246.41	9.141	LPstp Mean > LDstp Mean and Reject H_0	2059.19	3183.32
31.92	2597.03	157.64	9.396	LPstp Mean > LDstp Mean and Reject H_0	1930.54	2948.23
34.58	2118.11	103.59	9.502	LPstp Mean > LDstp Mean and Reject H_0	1598.99	2430.04
37.25	1712.18	154.69	9.068	LPstp Mean > LDstp Mean and Reject H_0	1220.86	1894.11
39.92	1570.94	142.81	9.056	LPstp Mean > LDstp Mean and Reject H_0	1119.05	1737.21
42.59	1252.49	137.66	8.826	LPstp Mean > LDstp Mean and Reject H_0	867.27	1362.38
45.26	1131.40	146.56	8.631	LPstp Mean > LDstp Mean and Reject H_0	761.19	1208.48
47.93	945.00	107.66	8.814	LPstp Mean > LDstp Mean and Reject H_0	651.14	1023.53
50.60	591.40	96.88	8.262	LPstp Mean > LDstp Mean and Reject H_0	377.21	611.83
53.26	663.91	115.00	8.301	LPstp Mean > LDstp Mean and Reject H_0	419.31	678.51
59.92	641.71	114.21	8.100	LPstp Mean > LDstp Mean and Reject H	399.86	655.13
66.61	393.75	136.72	6.186	LPstp Mean > LDstp Mean and Reject H_0	175.59	338.46
79.95	219.06	136.09	3.242	LPstp Mean > LDstp Mean and Reject H_0	32.82	133.11
106.63	120.00	169.22	-2.42	LPstp Mean < LDstp Mean and Reject H_0	-88.95	-9.48
133.32	89.85	163.28	-4.071	LPstp Mean < LDstp Mean and Reject H_0	-108.77	-38.08
160.00	55.31	309.69	-8.375	LPstp Mean < LDstp Mean and Reject H_0	-313.91	-194.84
186.69	40.63	196.10	-8.089	LPstp Mean < LDstp Mean and Reject H_0	-193.13	-117.81
213.37	31.25	232.50	-8.953	LPstp Mean < LDstp Mean and Reject H_0	-245.31	-157.19
240.06	24.38	250.16	-9.358	LPstp Mean < LDstp Mean and Reject H_0	-273.06	-178.49
266.75	21.41	251.87	-9.542	LPstp Mean < LDstp Mean and Reject H_0	-277.79	-183.12

Table 4.3.2.1 Time Cost of Steepest Ascent Hill-climbing on Class III

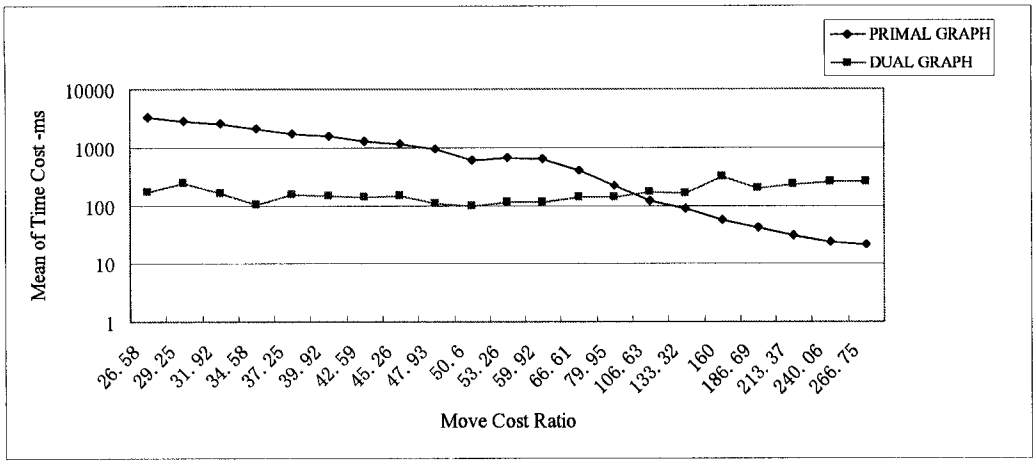


Figure 4.3.2.1 Comparison of Tm Mean of Steepest Ascent Hill-climbing on Class III

Move Cost Ratio	LPstp Rn	LPstp Nd	LPstp Cc	LDstp Rn	LDstp Nd	LDstp Cc
26.58	431.56	110836.45	554182.20	5.72	26480.09	291280.90
29.25	362.75	95797.10	478985.53	4.55	23369.30	257062.22
31.92	316.07	86962.90	434814.47	4.97	22027.07	242297.77
34.58	246.97	70119.46	350597.28	2.70	16048.47	176533.19
37.25	197.45	56907.57	284537.88	3.82	24384.06	268224.70
39.92	177.33	52547.70	262738.47	3.21	21893.27	240826.00
42.59	135.41	41548.94	207744.69	2.91	21133.92	232473.10
45.26	119.06	37236.59	186182.95	2.96	22815.06	250965.60
47.93	99.59	31438.24	157191.20	2.08	16508.06	181588.66
50.60	60.89	19639.98	98199.90	1.78	15392.84	169321.23
53.26	66.97	21913.84	109569.20	1.99	17557.42	193131.60
59.92	63.98	21343.44	106717.20	1.73	17331.28	190644.16
66.61	36.99	12847.99	64239.95	1.80	20173.94	221913.44
79.95	19.52	7110.69	35553.45	1.58	19977.71	219754.81
106.63	10.47	3953.05	19765.25	1.49	25476.95	280246.38
133.32	7.22	2773.24	13866.20	1.24	24779.75	272577.30
160.00	4.56	1783.16	8915.80	1.21	28905.13	317956.40
186.69	3.23	1231.18	6155.95	1.15	29945.87	329404.62
213.37	2.65	999.85	4999.25	1.12	34841.53	383256.88
240.06	2.16	789.05	3945.25	1.14	38444.64	422891.00
266.75	1.92	681.67	3408.35	1.09	38856.80	427424.97

Table 4.3.2.2 Rn, Nd and Cc of Steepest Ascent Hill-climbing on Class III

Table 4.3.2.1 is the time cost (Tm) result of steepest ascent hill-climbing on Class III on both primal and dual constraint graphs. The result of Table 4.3.2.1 is also showed as a graph in Figure 4.3.2.1. From this figure we can see that steepest ascent hill-climbing can get a better performance in the dual representation when move cost ratio is low (MoR from 26.58 to 79.95). As the move cost ratio is increasing, steepest ascent hill-climbing will find a solution faster in the primal representation (MoR from 106.63 to 266.75) than in the dual representation. We give Rn, Nd and Cc of steepest ascent hill-climbing on Class III in Table 4.3.2.2.

4.3.3 Min-conflicts Heuristics Hill-climbing on Class HI

Move Cost Ratio	LPmc Tm Mean	LDmc Tm Mean	T_value	Accept H_0?	CI Lower Bound	CI Upper Bound
26.58	2462.35	85.60	10.113	LPmc Mean > LDmc Mean and Reject H_0	1916.15	2837.34
29.25	2318.19	81.41	9.704	LPmc Mean > LDmc Mean and Reject H_0	1785.02	2688.53
31.92	2173.60	97.98	9.733	LPmc Mean > LDmc Mean and Reject H_0	1657.67	2493.56
34.58	2058.75	72.18	9.806	LPmc Mean > LDmc Mean and Reject H_0	1589.52	2383.61
37.25	2038.75	106.78	9.721	LPmc Mean > LDmc Mean and Reject H_0	1542.45	2321.48
39.92	1735.60	90.15	9.554	LPmc Mean > LDmc Mean and Reject H_0	1307.91	1982.98
42.59	1821.83	145.63	9.273	LPmc Mean > LDmc Mean and Reject H_0	1321.91	2030.46
45.26	1556.29	89.17	9.494	LPmc Mean > LDmc Mean and Reject H_0	1164.23	1769.99
47.93	1457.30	80.93	9.498	LPmc Mean > LDmc Mean and Reject H_0	1092.36	1660.37
50.60	1350.16	93.12	9.341	LPmc Mean > LDmc Mean and Reject H_0	993.26	1520.81
53.26	1337.82	97.50	9.287	LPmc Mean > LDmc Mean and Reject H_0	978.55	1502.08
59.92	1287.49	101.40	9.482	LPmc Mean > LDmc Mean and Reject H_0	940.91	1431.24
66.61	963.13	90.15	9.184	LPmc Mean > LDmc Mean and Reject H_0	686.67	1059.28
79.95	549.21	98.43	8.678	LPmc Mean > LDmc Mean and Reject H_0	348.97	552.58
106.63	172.81	167.35	0.266	Accept H_0	-34.72	45.64
133.32	83.28	148.89	-4.675	LPmc Mean < LDmc Mean and Reject H_0	-93.12	-38.11
160.00	48.44	126.41	-9.901	LPmc Mean < LDmc Mean and Reject H_0	-117.36	-78.57
186.69	35.15	215.78	-11.891	LPmc Mean < LDmc Mean and Reject H_0	-210.41	-10.85
213.37	22.19	214.84	-11.971	LPmc Mean < LDmc Mean and Reject H_0	-224.19	-161.11
240.06	14.85	245.81	-11.467	LPmc Mean < LDmc Mean and Reject H_0	-269.84	-191.07
266.75	12.50	186.57	-11.047	LPmc Mean < LDmc Mean and Reject H_0	-204.95	-143.18

Table 4.3.3.1 Time Cost of Min-conflicts Heuristics Hill-climbing on Class III

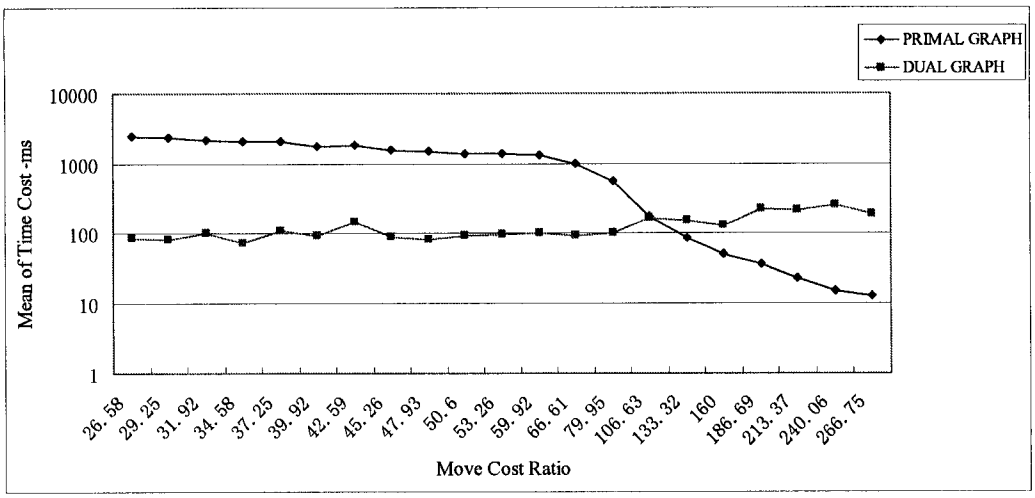


Figure 4.3.3.1 Comparison of Tm Mean of

Min-conflicts Heuristics Hill-climbing on Class HI

Move Cost Ratio	LPmc Rn	LPmc Nd	LPmc Cc	LDmc Rn	LDmc Nd	LDmc Cc
26.58	503.72	50474.98	252374.9	6.97	8383.09	92213.99
29.25	485.95	47376.47	236882.38	6.24	8108.44	89192.84
31.92	489.17	44655.48	223277.36	6.62	9386.93	103256.23
34.58	472.97	42179.20	210896.00	5.19	7268.07	79948.77
37.25	487.91	42241.86	211209.31	6.80	11019.01	121209.11
39.92	438.27	35993.32	179966.69	5.12	8908.09	97988.99
42.59	475.68	37991.71	189958.56	4.59	10541.29	115954.29
45.26	412.5	32551.96	162759.80	4.55	9154.77	100702.49
47.93	401.36	30409.69	152048.45	4.08	8266.28	90929.08
50.60	359.00	27374.82	136874.10	4.35	9555.30	105108.34
53.26	382.61	28069.75	140348.75	4.41	9813.70	107950.70
59.92	373.22	26799.9	133999.50	3.73	9947.34	109420.81
66.61	286.48	19872.24	99361.20	3.24	9072.86	99801.46
79.95	176.27	11717.22	58586.10	2.62	10214.13	112355.54
106.63	58.18	3671.33	18356.65	2.33	11581.12	127392.32
133.32	29.27	1809.22	9046.09	2.02	10291.42	113205.62
160.00	18.11	1061.99	5309.95	1.81	13035.08	143385.88
186.69	13.06	773.80	3869.00	1.58	13387.13	147258.44
213.37	8.77	487.13	2435.64	1.58	14913.96	164053.56
240.06	5.90	323.78	1618.90	1.32	14392.93	158322.23
266.75	5.07	270.84	1354.20	1.24	16565.75	182223.27

Table 4.3.3.2 Rn, Nd and Cc of Min-conflicts Heuristics Hill-climbing on Class III

Table 4.3.3.1 is the time cost (Tm) result of min-conflicts heuristics on Class III on both primal and dual constraint graphs. The result of Table 4.3.3.1 is also showed as a graph in Figure 4.3.3.1. From this figure we can see that min-conflicts heuristics hill-climbing can get a better performance in the dual representation when move cost ratio is low (MoR from 26.58 to 79.95). As the move cost ratio is increasing, min-conflicts heuristics hill-climbing will find a solution faster in the primal representation (MoR from 133.32 to 266.75) than in the dual representation. We give Rn, Nd and Cc of min-conflicts heuristics hill-climbing on Class III in Table 4.3.3.2.

4.3.4 Comparisons among Different Hill-climbing algorithms on Class III

Based on Table 4.3.1.1, Table 4.3.2.1 and Table 4.3.3.1, we give Figure 4.3.4.1 to compare the Tm means of all the three algorithms on Class III. Figure 4.3.4.1 shows three hill-climbing methods have similar characters as they are in Class I and Class II. Figure 4.3.4.2 and Figure 4.3.4.3 comparing Nc among the three algorithms on both representations are based on Table 4.3.1.2, Table 4.3.2.2 and Table 4.3.3.2.

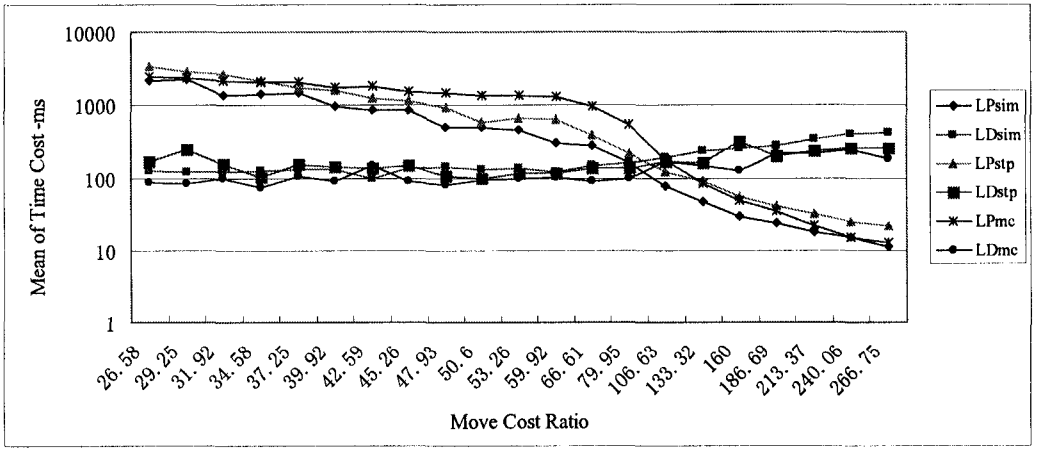


Figure 4.3.4.1 Comparisons of Tm Means for LPsim, LPstp, LPmc, LDsim, LDstp and LDmc on Class III

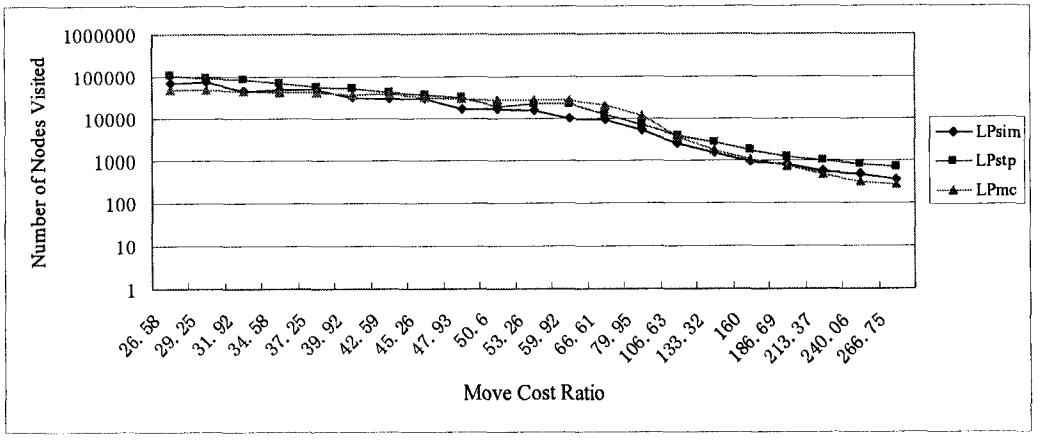


Figure 4.3.4.2 Comparisons of Nc for LPsim, LPstp and LPmc on Class III

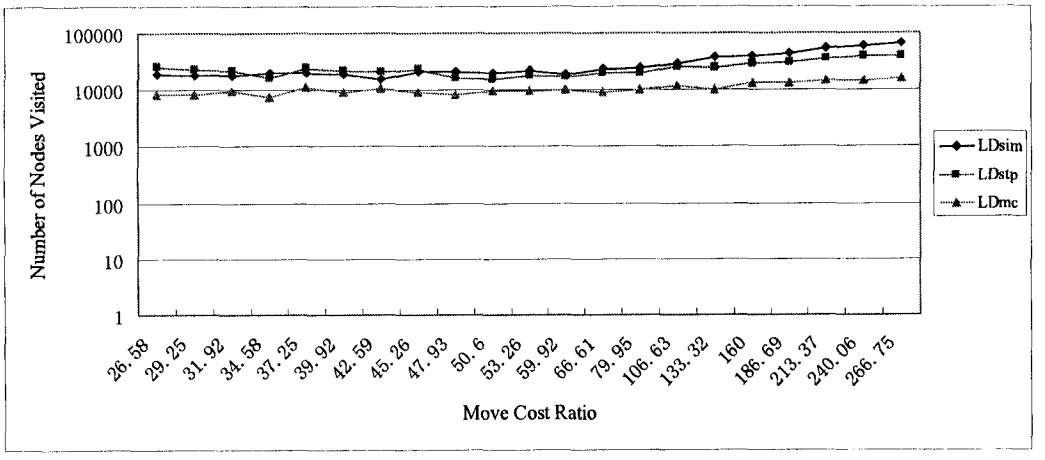


Figure 4.3.4.3 Comparisons of Nc for LDsim, LDstp and LDmc on Class III

4.4 Conclusions

In this chapter we present the experiment result on three problem classes. The comparisons show that local search methods can get a better performance in the dual representation when move cost ratio is low. As the move cost ratio is increasing, local search will find a solution faster in the primal representation than in the dual representation. Among the three local search methods in our empirical study,

min-conflicts heuristics hill-climbing suggests that it have the best performance among the three algorithms, steepest ascent hill-climbing tends to be the worst and simple hill climbing is in the middle or sometimes it is the best.

Chapter 5

Conclusions

In this thesis we designed an empirical study to investigate the behaviour of several local search methods in primal and dual constraint graph representations. Such local search methods used in our empirical study included simple hill-climbing, steepest ascent hill-climbing and min-conflicts heuristics hill-climbing.

Our approach focused on observing behaviours of several local search methods in primal and dual constraint graphs. The measurements we used to characterize the behaviour of the three local search methods were: number of times that local search randomly generates the initial state, number of nodes visited in the search procedure, number of constraints checked in the search procedure and CPU time cost by the search procedure to find a solution. Since search node in primal constraint graph has different meaning from it in dual constraint graph, the number of nodes visited was used to compare different local search methods on the same constraint graph representation. Between the primal and dual representations, we briefly compared time cost to find a solution. We used T-test which is a statistical analysis method to compare the time cost means of two groups to support the comparison in our empirical study. We launched all T-tests by a given risk level $\alpha=0.05$, critical value $T\_cv=1.645$. A 95% confidence interval on the difference of means was also given in the comparison result.

In our comparison result all the three hill-climbing algorithms could find a solution within a shorter time in dual representation than in primal representation when move cost ratio was low. As the move cost ratio was increasing which indicated the problem was getting looser, local search methods got a better performance in the primal representation than in the dual representation. Such results show that we can use local search to solve a CSP with tight constraints in its dual representation and gain a better performance than using it in its primal representation. When constraints are getting looser, using local search in primal representation is a better choice.

Among the three local search methods used in our empirical study, min-conflicts heuristics hill-climbing suggested that it have the best performance among the three algorithms while steepest ascent hill-climbing tended to have the worst performance and simple hill climbing was in the middle or sometimes it was the best.

5.1 Future Work

In our empirical study, move cost of local search is an essential factor affecting the performance of local search. Domain size, number of constraints and constraint tightness, constraint density and constraint arity in both primal and dual representations can affect the move cost. It would be interesting if we pay more attention to all of the above factors on how they affect the move cost in various local search methods. Such research will help us give a clearer view to decide which kind of representation is more suitable to be used in solving CSPs with different characters.

One problem for local search methods is meeting local optima. In our empirical study, we regenerate initial state or break ties randomly (in min-conflicts heuristics) to escape from local optima. There is also other improvement such as random walk [SKC94] or Tabu search [Glo89] can be add to current local search methods to avoid local optima.

Appendix A

T-test

In this appendix we review some issues related to T-test which is used in the empirical study to investigate the behaviour of local search algorithms in CSP's primal and dual representations. Due to the non-deterministic manner of local search algorithms, one can get different solutions if running the algorithm on the same problem instance for several times, each of which costs different time length. The traditional way to get the performance of such algorithms is to run the algorithm on the same problem instance for a number of rounds and then get the mean. For example, one can run simple hill-climbing on a CSP's primal constraint graph for 100 rounds. Then one can get the average search time by dividing the sum of search time consumed in each round by 100. Such average search time is also called the mean of search time for these 100 rounds. Most research on local search methods such as proposing a new local search algorithm or making an improvement for a current algorithm will compare the means of two algorithms' time cost, which is a common form of conducting an empirical study. Most research will conclude that algorithm B is better than algorithm A because the mean of algorithm B's time cost is less than the mean of algorithm A's time cost. Is it always correct to make such a conclusion? If the two means have some difference but do not differ a lot, for instance, the mean of algorithm A's time cost is greater than the mean of algorithm B's time cost with a difference of 0.1 milliseconds, can we say algorithm B is better than algorithm A? Once we have summarized such data as means, how do we decide if the observed differences between the two algorithms are real or just a chance

difference caused by the natural variation within the measurements? In this thesis we use a statistical analysis method called **T-test** which is a common way to approach above questions. Generally when the sample size of each group is larger than 30, such test for assessing the difference of the means between two groups is called a **Z-test**. In the following let us review the related T-test issues.

The T-test assesses whether the means of two groups are statistically different from each other. Figure Apendix_1 is the formula for the T-test when the variances of the two groups are markedly different [PG94]. This formula is a ratio. The top part of the ratio is the difference between the two means or averages. The bottom part is a measure of the variability or dispersion of the measurements which is called the **standard error of the difference**. To compute the standard error of the difference, we take the variance for each group and divide it by the number of rounds of running the algorithm in that group. We add these two values and then take their square root. The specific formula for computing the standard error of the difference is given in Figure Apendix_2.

The result of the formula in Figure Apendix_1 is called a **T_value**. Now we illustrate how to use the T_value to tell whether the difference of the two means is significant or not:

1) The T-test is given under two hypotheses:

   H_0: $\bar{\mu}_A = \bar{\mu}_B$, which sets the hypothesis that the two means of the two groups have no significant difference.

   H_A: $\bar{\mu}_A > \bar{\mu}_B$ (or $\bar{\mu}_A < \bar{\mu}_B$), which sets the hypothesis that the mean of algorithm A's time cost is greater (or less) than the mean of algorithm B's time cost.

2) To test the significance, we need to set a risk level which is called the **alpha level**. For practical purposes, the alpha level is conventionally set at 0.05. This means that five times out of a hundred you would find a statistically

significant difference between the means even if there was none i.e., such difference is gained by chance.

$$\text{T\_value} = \frac{\text{difference between group means}}{\text{variability of groups}}$$

$$= \frac{\bar{X}_A - \bar{X}_B}{SE\,(\bar{X}_A - \bar{X}_B)}$$

$$= \frac{\bar{X}_A - \bar{X}_B}{\sqrt{\frac{S_A^2}{n_A^2} + \frac{S_B^2}{n_B^2}}}$$

Figure Apendix_1 T-test formula when variances are unequal

$$SE\,(\bar{X}_A - \bar{X}_B) = \sqrt{\frac{S_A^2}{n_A^2} + \frac{S_B^2}{n_B^2}}$$

Figure Apendix_2 Formula for the standard error of the difference between the means when variances are

3) We also need to determine the **degrees of freedom (df)** for the test. In the T-test for equal variances, the degrees of freedom is the sum of the number of rounds of running the algorithm in each group minus 2. For example, we run

algorithm A and algorithm B both for 10 rounds, i.e., n_1=10 and n_2=10. Then we can get the degrees of freedom for the T-test is 10 + 10 – 2 = 18. In the T-test for unequal variances, the degrees of freedom is calculated in a very complicated way. But for practical purposes, when $n_1$and n_2 are both larger than 100, we can define df is $\rightarrow \infty$.

Given the alpha level and the degrees of freedom, we can look up the **T_criticalvalue** (**T_cv**) in a standard table of significance which is called a **T-table**. The T-table is used to determine whether the T_value is large enough to be significant. For example, given the alpha level α=0.05 and degrees of freedom df=18, we find T_cv = 1.734. Now we compare T_value with T_cv. If |T_value| > T_cv, we can conclude that the difference between the means of the two groups is significant and such difference between the groups is not likely to have been a chance finding, i.e., we reject the hypothesis H_0: the two means have no significant difference. If |T_value| < T_cv, we will accept the hypothesis H_0: the two means have no significant difference. The T_value will be positive if the first mean is larger than the second and negative if it is smaller. If |T_value| > T_cv and T_value is positive, we can conclude that the mean of algorithm A's time cost is greater than the mean of algorithm B's time cost. If |T_value| > T_cv and T_value is negative, we can conclude that the mean of algorithm B's time cost is greater than the mean of algorithm A's time cost.

There are several issues we need to clarify when applying the T-test method in our empirical study:

1) One sided T-test and two sided T-test: In **one-sided T-test**, it is assumed that before doing the test we had a hypothesis that one mean of the two means was greater (or less) than the other mean. If we did not have such a prior hypothesis, and we only aim to test for a possible difference between the means, we need to do a **two-sided T-test**. The T-table for a one-sided T-test is different from the T-table for a two-sided T-test. In a two-sided T-test one would mostly multiply the alpha level by two. One-sided T-test is also called

one-tailed T-test and two-sided T-test is also called **two-tailed T-test**. In our empirical study, we concentrate on whether the mean of time cost for local search in primal is greater (or less) than the mean of time cost for local search in dual, which is a one-sided T-test.

2) T-test under equal and unequal variances: In Figure Apendix_1 we give the formula for T-test when the variances for two means are unequal. If the two variances are equal, there will be another T-test formula. The way to calculate the standard error of the difference (SE) and degrees of freedom (df) is also different. But in our empirical study, the number of tests in each group is same, i.e., we run local search in primal representation and local search in dual representation on the same problem instance for equivalent times where $n_1 = n_2$. Under such circumstance the T-test formula for equal variances is the same as the T-test formula for unequal variances.

3) T_cv = 1.645 when $df \rightarrow \infty$ and $\alpha = 0.05$ in a one-sided T-test: In our empirical study, we will run each algorithm on one problem instance for 100 times, which means the sample size is large enough to take the critical value (T_cv) as 1.645. So each time after we get the T_value we can compare it with 1.645 to see if there is significant difference between the means.

Some researchers [SC89] recommend reporting **confidence interval** wherever means are estimated in T-test and their difference are reported. Confidence interval for the difference of the means in T-test [SC89] is an interval estimate for the difference of the means. Interval estimates are often desirable because the estimate of the difference of the means may vary from sample to sample. Instead of a single estimate for the difference of the means, a confidence interval generates a lower and upper bound for the difference of the means. The confidence interval estimate indicates how much uncertainty there is in our estimate of the true difference of the means. Confidence interval is expressed under a confidence level. In practice a 95% confidence level is the most commonly used. A 95% confidence level can not be considered that there is a 95% probability that the interval computed from a given sample contains the true difference

of the means. The interval computed from a given sample either contains the true difference of the means or it does not. The confidence level is the proportion of confidence intervals that may be expected to contain the true difference of the means. That is, for a 95% confidence interval, if many samples are collected and for each sample the confidence interval is computed, there are about 95% of these intervals which would contain the true difference of the means. In Figure Appendix_3 we give the formula of confidence interval at a 95% level for the difference of the means in a one-sided T-test when df $\rightarrow \infty$, α=0.05 and variances are unequal. In our empirical study we report the 95% confidence interval for the difference of the means in the T-test.

$$\text{a 95\% confidence interval} = (\bar{X}_A - \bar{X}_B) \pm 1.96 \times SE(\bar{X}_A - \bar{X}_B)$$

$$= (\bar{X}_A - \bar{X}_B) \pm 1.96 \times \sqrt{\frac{S_A^2}{n_A^2} + \frac{S_B^2}{n_B^2}}$$

Figure Appendix_3 Formula of confidence interval at a 95% confidence

Bibliography

[Bar98]Roman Barták. "On-line Guide to Constraint Programming", http://kti.mff.cuni.cz/~bartak/constraints/, Prague, 1998,

[Bar99]Roman Barták. *"Constraint Programming: In Pursuit of the Holy Grail"*, in Proceedings of the Week of Doctoral Students (WDS99), Part IV, pages 555-564, MatFyzPress, Prague, 1999.

[BB98]F. Bacchus and P. van Beek. *"On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems"*, in Proc. National Conference on Artificial Intelligence (AAAI-98), Madison, Wisconsin, 1998.

[BC94]Christian Bessiere and Marie-Odile Cordier. *"Arc-Consistency and Arc-Consistency Again"*, in Proceedings ECAI'94 Workshop on Constraint, 1994.

[BC99]P. van Beek and X. Chen. *"CPlan: A constraint programming approach to planning"*, In Proceedings of the Sixteenth National Conference on Artificial Intelligence, pages 585–590, Orlando, Florida, 1999.

[BCBW02]Fahiem Bacchus, Xinguang Chen, Peter van Beek and Toby Walsh. *"Binary vs. Non-Binary Constraints"*, Artificial Intelligence, 2002.

[BD94]P. van Beek and R. Dechter. *"Constraint Restrictiveness Versus Local and Global Consistency"*, in Principles of Knowledge Representation and Reasoning, (KR-94), pages 572-582, Bonn, Germany, May 1994.

[BD95]Peter van Beek and Rina Dechter. *"On the minimality and global consistency of row-convex constraint networks"*, Journal of the ACM, vol. 42, pages 543--561, 1995.

[Bee94]Peter van Beek. *"On the Inherent Level of Local Consistency in Constraint Networks"*, National Conference on Artificial Intelligence, pages 368-373, 1994.

[Bes94]C. Bessi. *"A fast algorithm to establish arc-consistency in constraint networks"*, Technical Report TR-94-003, January 1994.

[BF00]J. C. Beck and Mark S. Fox. *"Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics"*, Artificial Intelligence, 117(1):31-81, 2000.

[BFML99]C. Bessiere, E. C. Freuder, P. Meseguer and J. Larrosa. *"On forward checking for non-binary constraint satisfaction"*, in Principles and Practice of Constraint Programming, CP-99, volume 1713, pages 88-102, Springer Verlag, 1999.

[BFR95]C. Bressiere, E. C. Freuder and J. C. Regin. *"Using inference to reduce arc-consistency computation"*, in Proceedings of IJCAI, pages 592-598, 1995.

[BFW95]A. Borning, B. N. Freeman-Benson and M. Wilson. *"Constraint hierarchies"*, Lisp and Symbolic Computation, Vol. 5, pages 223-270, 1995.

[BG95]F. Bacchus and A. Grove. *"On the Forward Checking Algorithm"*, Principles and Practice of Constraint Programming (CP-95), pages 292--309, 1995.

[BMR95] S. Bistarelli, U. Montanari and F. Rossi. *"Constraint solving over semirings"*, in Proceedings of IJCAI, pages624- 630, 1995.

[BMRSVF99] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie and H. Fargier. *"Semiring-based CSPs and Valued CSPs: Frameworks, Properties and Comparison"*, Constraints 4(3), 1999.

[BR75]James R. Bitner and Edward M. Reingold. *"Backtrack programming techniques"*, ACM Press, pages: 651 – 656, New York, NY, USA, 1975.

[BR95]F. Bacchus and P. van Run. *"Dynamic variable ordering in CSPs"*, in *Proceedings CP'95*, pages 258–275, Cassis, France, 1995.

[BR96]Christian Bessiere and J. C. Regin. *"MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems"*, Principles and Practice of Constraint Programming, pages 61-75, 1996.

[BR97]C. Bessiere and J. C. Regin. *"Arc consistency for general constraint networks: preliminary results"*, in Proceedings of Internantional Joint Conference on Artificial Intelligence, IJCIA-97, PAGES 398-404, 1997.

[BTW96]J. Borrett, E. Tsang and N. Walsh. *"Adaptive constraint satisfaction: the quickest first principle"*, Proceedings of ECAI, pages160-169, 1996.

[CBP95]J. C. Culberson, A. Beacham, D. Papp. *"Hiding Our Colors"*, CP'95 Workshop on Studying and Solving Really Hard Problems, Cassis, France, 1995.

[DB97]Rina Dechter and Peter van Beek. *"Local and global relational consistency"*, Theoretical Computer Science, Vol. 173, pages 283-308, 1997.

[DB01]R. Debruyne and C. Bessiere. *"Domain filtering consistencies"*, Artificial Intelligence Research, 14:205–230, 2001.

[Dec90]R. Dechter. *"On the expressiveness of networks with hidden variables"*, in Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), pages 556-562, Boston, MA, July 1990.

[Deb96]R. Debruyne. *"Arc-Consistency in Dynamic CSPs is No More Prohibitive"*, Eighth Conference on Tools With Artificial Intelligence, pages 299–306, 1996.

[Dec92]R. Dechter. *"From local to global consistency"*, Artificial Intelligence, 55:87–102, 1992.

[DFGMTT96]David A. Clark, Jeremy Frank, Ian P. Gent, Ewan MacIntyre, Neven Tomov and Toby Walsh. *"Local Search and the Number of Solutions"*, Principles and Practice of Constraint Programming, 1996.

[DJ97]N. W. Dunkin and P. G. Jeavons. *"Expressiveness of Binary Constraints for the Frequency Assignment Problem"*, in Proceedings of the IEEE/ACM Workshop, Dial M for Mobility, 1997.

[DM89]R. Dechter and I. Meiri. *"Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems"*, in Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pages 290-296, 1989.

[DP89]R. Dechter and J. Pearl. *"Tree Clustering schemes for constraint-processing"*, Artificial Intelligence, Vol. 38(3), pages 353-366, April 1989.

[Fal94]Boi Faltings. *"Arc-Consistency for Continuous Variables"*, Artificial Intelligence, Vol. 65, pages 363-376, 1994.

[FCS97]Jeremy Frank, Peter Cheeseman, and John Stutz. *"When gravity fails: Local search topology"*, Journal of Artificial Intelligence Research, 7:249-281, 1997.

[FD95]D. Frost and R. Dechter. *"Look-ahead value ordering for constraint satisfaction problems"*, Proceedings of IJCAI, pages 572-577, 1995.

[FL96]H. Fargier and J. Lang. *"Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge"*, Proceedings of AAAI, pages 175-180, 1996.

[Fra96]Francesca Rossi. *"Existential Variables and Local Consistency in Finite Domain Constraint Problems"*, Principles and Practice of Constraint Programming, pages 382-396, 1996.

[Fre78]Eugene C. Freuder. *"Synthesizing constraint expressions"*, Communications of the ACM, Volume 21, Issue 11, November 1978.

[Fre95]E. C. Freuder. *"Systematic versus stochastic constraint satisfaction, Panel discussion"*, Proceedings of IJCAI, pages 2027-2032, 1995.

[FW89]Eugene C. Freuder and Richard J. Wallace. *"Partial Constraint Satisfaction"*, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89, Detroit, Michigan, USA, 1989.

[Gas74]J. Gaschnig. *"A Constraint Satisfaction Method for Inference Making"*, in Proceedings of the 12th Annual Allerton Conference on Circuit Systems Theory, pages 866-874, 1974.

[Gas78] J. Gaschnig. *"Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems"*, in Proceedings of the Canadian Artificial Intelligence Conference, pages 268-277, 1978.

[Gin93]M. L. Ginsberg. *"Dynamic Backtracking"*, Journal of Artificial Intelligence Research vol. 1, AI Access Foundation and Morgan Kaufmann, pages 25–46, 1993.

[GJC94]M. Gyssens, P. Jeavons, and D. Cohen. *"Decomposing constraint satisfaction problems using database techniques"*, Artificial Intelligence, 66:57–89, 1994.

[Glo89] F.Glover. *"Tabu Search – Part I"*, ORSA Journal on Computing, 1(3):190-206, 1989.

[GMC97]L. Getoor, G. Ottosson, M. Fromherz, and B. Carlson. *"Effective redundant constraints for online scheduling"*, in Proceedings of the Fourteenth National Conference on Artificial Intelligence, pages 302–307, 1997.

[GMPW96]I. Gent, E. MacIntyre, P. Prosser and T. Walsh. *"The constrainedness of search"*, in Proceedings of AAAI, 1996.

[Got00]G. Gottlob. *"A comparison of structural CSP decomposition methods"*, Artificial Intelligence, 124:243–282, 2000.

[GSW00]I. Gent, K. Stergiou and T. Walsh. *"Decomposable constraints"*, in New Trends in Constraints, Proceedings of ERCIM/Compulog-Net Workshop, Spring Verlag, 2000.

[GW93]I. Gent and T. Walsh. *"Towards an Understanding of Hill-climbing Procedures for SAT"*, in Proceedings of the Eleventh National Conference on Artificial Intelligence, pages 28-33, 1993.

[GW95]I. Gent, T. Walsh. *"Phase transitions from real computational problems"*, in Proceedings of 8th International Symposium on AI, pages 356-364, 1995.

[HDR78]R. Haralick, L. S. Dais and A. Rosenfeld. *"Reduction Operations for Constraint Satisfaction"*, Information Science 14:199-219, 1978.

[HDT92]P. van Hentenryck, Y. Deville and Choh-Man Teng. *"A generic arc-consistency algorithm and its specializations"*, Artificial Intelligence Vol. 57, pages 291-321, 1992.

[HE80]R. Haralick and G. Elliot. *"Increasing tree search efficiency for constraint satisfaction problem"*, Artificial Intelligence 14(3):263-313, 1980.

[HL88]C. Han and C. Lee. *"Comments on Mohr and Henderson's path consistency algorithm"*, Artificial Intelligence 36, pages 125-130, 1988.

[Hoo98]Holger H. Hoos. *"Stochastic Local Search – Methods, Models, Applications"*, PhD thesis, Darmstadt University of Technology, Germany, 1998.

[HS98]Holger H. Hoos and Thomas Stützle. *"Evaluating Las Vegas Algorithms -- Pitfalls and Remedies"*, In Proceedings of UAI-98, pages 238-245. Morgan Kaufmann Publishers, 1998.

[KB97]G. Kondrak and P. van Beek. *"A theoretical evaluation of selected backtracking algorithms"*, Artificial Intelligence, 89:365-387, 1997.

[Kum87]V. Kumar. *"Dept-First Search"*, in Encyclopedia of Artificial Intelligence, Vol2, pages 1004-1005, 1987.

[Jea95]P. Jeavons. *"Tractable constraints on ordered domains"*, Artificial Intelligence, 79:327–339, 1995.

[JC99]D. E. Joslin and D. P. Clements. *"Squeaky Wheel Optimization"*, Journal of Artificial Intelligence Research vol. 10, pages 353–373, 1999.

[JL02]N. Jussien and O. Lhomme. *"Local Search With Constraint Propagation and Conflict-Based Heuristics"*, Artificial Intelligence vol. 139, no. 1, pages 21–45, 2002.

[LD00]J. Larrosa and R. Dechter. *"On The Duel Representation of Non-Binary Semiring-Based CSPs"*, in workshop 1 (Soft Constraints) of the "Sixth International Conference on Principles and Practice of Constraint Programming" (CP2000), September, 2000.

[LLH95]J. Lee, H. Leung and H. Won. *"Extending GENET for non-binary constraint satisfaction problems"*, in 7th International Conference on Tools with Artificial Intelligence, pages 338-342, 1995.

[LMSV98]J. Larrosa, P. Meseguer, T. Schiex and G. Verfailli. *"Reversible DAC and Other Improvements for Solving Max-CSP"*, AAAI-98, pages 347-352, 1998.

[Kum92]Vipin Kumar. *"Algorithms for Constraint Satisfaction Problems: A Survey"*, the AI Magazine, pages 32-44, 1992.

[Mac77] A. K. Mackworth. *"Consistency in networks of relations"*, Artificial Intelligence, 8(1):99-118, 1977.

[Mc79]J. McGregor. *"Relational Consistency Algorithms and Their Applications in Finding Subgraph and Graph Isochronism"*, Information Science 19:229-250, 1979.

[MH86]R. Mohr and T.C. Henderson. *"Arc and Path consistency revisited"*, Artificial Intelligence 28:225—233, 1986.

[MM00]V. Manquinho and J. Marques-Silva. *"On solving boolean optimization with satisfiability-based algorithms"*, in Sixth International Symposium on Artificial Intelligence and Mathematics, January 2000.

[MPJL93] Steven Minton, Andy Philips, mark D. Johnston and Philip Laird. *"Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems"*, Journal of Artificial Intelligence Research, 1993.

[Nad88]B. Nadel. *"Tree search and arc consistency in constraint satisfaction algorithms"*, in Search in Artificial Intelligence, Springer-Verlag, pages 287-342, 1988.

[Nad90]B. Nadel. *"Some Applications of the Constraint Satisfaction Problem"*, Technical Report CSC-90-008, Computer Science Department, Wayne State University, 1990.

[Nag01]Sivakumar Nagarajan. *"On Dual Encoding for Constraint Satisfaction"*, PhD thesis, University of Regina, 2001.

[Par97]Andrew J. Parkes. *"Clustering at the phase transition"*, in Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), pages 340-345, 1997.

[PG94] Armitage P and Berry G. *"Statistical Methods in Medical Research"*, 3rd edition, Oxford: Blackwell Scientific Publications, 1994.

[PG97]Wanlin Pang and Scott D. Goodwin. *"Constraint-Directed Backtracking"*, Australian Joint Conference on Artificial Intelligence, 1997.

[PG97a]W. Pang and S. D. Goodwin. *"A revised sufficient condition for backtrack-free search"*, in Proceedings of 10th Florida AI Research Symposium, pages 52--56, Daytona Beach, FL, May 1997.

[PG00]W. Pang and S. D. Goodwin. *"Consistency in general CSPs"*, in The 6th Pacific Rim International Conference on AI, pages 469–479, 2000.

[PG01]W. Pang and S. D. Goodwin. *"Binary representation for general CSPs"*, In Proceedings of 14th Florida AI Research Symposium (FLAIRS-2001), 2001.

[PR95] D. G. Pothos and E. B. Richards. *"An Empirical Study of Min-Conflict Hill Climbing and Weak Commitment Search"*, Proceedings of CP-95 Workshop: Studying and Solving Really Hard Problems, pages 140—146, 1995.

[Pre00]S. D. Prestwich. *"Stochastic Local Search in Constrained Spaces"*, Proceedings of Practical Applications of Constraint Technology and Logic Programming, pages 27-39, Practical Applications Company, 2000.

[Pre01]S. D. Prestwich. *"Local Search and Backtracking vs Non-Systematic Backtracking"*, AAAI 2001 Fall Symposium on Using Uncertainty within Computation, Cape Cod, MA, November 2001.

[Pre02]Steve Prestwich. *"Maintaining Arc-Consistency in Stochastic Local Search"*, in Workshop on Techniques for Implementing Constraint Programming Systems, 2002.

[Pre02a]S. D. Prestwich. *"Coloration Neighbourhood Search With Forward Checking"*, Annals of Mathematics and Artificial Intelligence vol. 34 no. 4, pages 327–340, 2002.

[RPD90]F. Rossi, C. Petrie and V. Dhar. *"On the equivalence of constraint satisfaction problems"*, in Proceedings of the 9th ECAI-90, pages 550-556, 1990.

[Rut94]Zs. Ruttkay. *"Fuzzy constraint satisfaction"*, Proceedings of 3rd Int. Conf. on Fuzzy Systems, pages 1263-1268, 1994.

[San94]M. Sannella. *"The SkyBlue Constraint Solver and its Applications"*, MIT Press, 1994.

[San99]T. Sandholm. *"An algorithm for optimal winner determination in combinatorial auctions"*, in Proceedings of IJCAI-99, 1999.

[SBHW96]B. Smith, S. C. Brailsford, P. M. Hubbard, and H. P. Williams. *"The progressive party problem: Integer linear programming and constraint programming compared"*, Constraints, 1:119–138, 1996.

[SC89] George W. Snedecor and William G. Cochran. *"Statistical Methods, Eighth Edition"*, Iowa State University Press, 1989.

[SF94]D. Sabin and E. C. Freuder. *"Contradicting conventional wisdom in constraint satisfaction"*, in Proceedings of the 11th European Conference on Artificial Intelligence, pages 125–129, Amsterdam, 1994.

[SF96]M. Sqalli and E. C. Freuder. *"Inference-based constraint satisfaction supports explanation"*, in Proceedings of AAAI, pages318-325, 1996.

[SF97]Daniel Sabin and Eugene C. Freuder. *"Understanding and Improving the MAC Algorithm"*, in Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP97), Austria, 1997.

[SG91]R. Sosic and J. Gu. *"3,000,000 queens in less than one minute"*, SIGART Bulletin, Vol. 2, pages 22-24, 1991.

[SG95]B. Smith and S. Grant. *"Sparse constraint graphs and exceptionally hard problems"*, in Proceedings of IJCAI, pages 646-651, 1995.

[SGS00]Josh Singer, Ian P. Gent and Alan Smaill. *"Backbone fragility and the local search cost peak"*, Journal of Artificial Intelligence Research, 12:235-270, 2000.

[SKC94]B.Selman, Henry A. Kautz and Bram Cohen. *"Noise Strategies for Improving Local Search"*, In AAAI'94, pages 337-343, 1994.

[SLM92]B. Selman, H. Levesque and D. Mitchell. *"A new method for solving hard satisfiability problems"*, in Proceedings of AAAI, pages 440-446, 1992.

[Smi94]Barbara M. Smith. *"The Phase Transition and the Mushy Region in Constraint Satisfaction Problems"*, European Conference on Artificial Intelligence (ECAI-94), pages 100-104, 1994.

[SRGV96]T. Schiex and J. C. Regin and C. Gaspin, G. Verfaille. *"Lazy arc consistency"*, in Proceedings of 13th National Conference on Artificial Intelligence (AAAI-96), pages 216-221, 1996.

[SSS97]Olaf Steinmann, Antje Strohmaier and Thomas Stützle. *"Tabu Search vs. Random Walk"*, KI-97 Advances in Artificial Intelligence, Springer Verlag, LNCS, Vol. 1303, 1997.

[SV98]Eddie Schwalb and L. Vila. *"Temporal Constraints: A Survey"*, Constraints, 3(2-3), pages 129-149, 1998.

[SW99]K. Stergiou and T. Walsh. *"Encodings of non-binary constraint satisfaction problems"*, in Proceedings of the National Conference on Artificial Intelligence, AAAI-99, pages 163-168, 1999.

[TBK95]E. Tsang, J. Borrett and A. Kwan. *"An attempt to map the performance of a range of algorithm and heuristic combinations"*, in Proceedings of AI and Simulated Behaviour, pages 203-215, 1995.

[Tsa93]E.P.K.Tsang. *"Foundations of Constraint Satisfaction"*, Academic Press, 1993.

[TW92]E. Tsang and C. Wang. *"A generic neural network approach for constraint satisfaction problems"*, Neural Network Applications, Springer-Verlag, 1992.

[Ull76]J. R. Ullman. *"An Algorithm for Subgraph Isomorphism"*, Journal of the ACM 23:31-42, 1976.

[Wal75]D.L. Waltz. *"Understanding line drawings of scenes with shadows"*, in The Psychology of Computer Vision, McGraw-Hill, New York, 1975.

[Wal96]R. Wallace. *"Practical applications of constraint programming"*, Constraints, Vol. 1, pages 139-168, 1996.

[WBHW01]J. P. Watson, J.C. Beck, A.E. Howe, and L.D. Whitley. *"Toward an Understanding of Local Search Cost in Job-shop Scheduling"*, in Proceedings of the Sixth European Conference on Planning (ECP-2001), 2001.

[WBF98]R. Weigel, C. Bliek, and B. Faltings. *"On reformulation of constraint satisfaction problems"*, in Proceedings of the 13th European Conference on Artificial Intelligence, pages 254–258, Brighton, United Kingdom, 1998.

[Yok97]Makoto Yokoo. *"Why adding more constraints make easier For Hill-climbing"*, Principles and Practice of Constraint Programming, 1997.

[YY97]C. Yang and Ming-Hsuan Yang. *"Constraint networks: A survey"*, in Proceedings of the IEEE volume 2, pages 1930-1935, 1997

Vita Auctoris

Mingyan Huang was born in 1974 in Shanghai, China. She graduated from Hua Gong Fu Zhong High School in 1992. From there she went to Wuhan University where she obtained a B.Eng. in Computer Software in 1996. From 1996 to 2001 she had been working in Shanghai Mobile Communication Corporations. She is currently a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in June 2004.