Hybrid Algorithms for Distributed Constraint Satisfaction

David Alexander James Lee

A thesis submitted in partial fulfilment
of the requirements of
The Robert Gordon University
for the degree of Doctor of Philosophy

April 2010

Supervised by Dr. Ines Arana, Dr. Hatem Ahriz and Dr. Kit-Ying Hui

Abstract

A Distributed Constraint Satisfaction Problem (DisCSP) is a CSP which is divided into several inter-related complex local problems, each assigned to a different agent. Thus, each agent has knowledge of the variables and corresponding domains of its local problem together with the constraints relating its own variables (intra-agent constraints) and the constraints linking its local problem to other local problems (inter-agent constraints). DisCSPs have a variety of practical applications including, for example, meeting scheduling and sensor networks. Existing approaches to Distributed Constraint Satisfaction can be mainly classified into two families of algorithms: systematic search and local search. Systematic search algorithms are complete but may take exponential time. Local search algorithms often converge quicker to a solution for large problems but are incomplete. Problem solving could be improved through using hybrid algorithms combining the completeness of systematic search with the speed of local search.

This thesis explores hybrid (systematic + local search) algorithms which cooperate to solve DisCSPs. Three new hybrid approaches which combine both systematic and local search for Distributed Constraint Satisfaction are presented: (i) DisHyb; (ii) Multi-Hyb and; (iii) Multi-HDCS. These approaches use distributed local search to gather information about difficult variables and best values in the problem. Distributed systematic search is run with a variable and value ordering determined by the knowledge learnt through local search.

Two implementations of each of the three approaches are presented: (i) using penalties as the distributed local search strategy and; (ii) using breakout as the distributed local search strategy. The three approaches are evaluated on several problem classes. The empirical evaluation shows these distributed hybrid approaches to significantly outperform both systematic and local search DisCSP algorithms.

DisHyb, Multi-Hyb and Multi-HDCS are shown to substantially speed-up distributed problem solving with distributed systematic search taking less time to run by using the information learnt by distributed local search. As a consequence, larger problems can now be solved in a more practical timeframe.

Acknowledgments

I am extremely grateful to my supervisors Dr. Ines Arana, Dr. Hatem Ahriz and Dr. Kit-Ying Hui for the many insights, discussions and support offered during my PhD research. Through these discussions, I not only learned a lot but was able to formulate my sketchy ideas into the completed works presented in this thesis. I am also very grateful for the time and advice given by my examiners, Prof. Miguel-Angel Salido and Prof. Susan Craw.

I have throughly enjoyed my time at the School of Computing. I would particularly like to thank Colin, Susan, Iain and Tommy for always finding computers to run my experiments on. I would also like to thank Ann, Gosia, Kathy, Diane and Marie for all their administrative assistance.

The research facilities at the School have been excellent. I must thank all of my colleague in CTC for creating the right environment for productive research namely Amandine, Ben, Bayo, Guofu, Ibrahim, Jean-Claude, Leszek, Malcolm, Miki, Nana, Nuka, Olivier, Peng, Peter, Ratiba, Richard, Sandy, Stella, Stewart, Thierry, Ulises, Yanghui and Yunhyong.

My biggest thanks must go to my parents who have always encouraged me to pursue my dreams. Without their constant support and encouragement, I would never have been able to complete this thesis. I would also like to thanks all of my friends who have offered a kind word to keep me going when things were tough.

Declarations

I hereby confirm that this thesis is my own work. I have cited all other work in the bibliography.

Parts of this work have appeared in the following publications:

Chapter 6

David Lee, Ines Arana, Hatem Ahriz and Kit-Ying Hui, 2008. A Hybrid Approach to Distributed Constraint Satisfaction. In: Danail Dochev, Paolo Traverso and Marco Pistore, ed. *Artificial Intelligence: Methodology, Systems and Applications. 13th International Conference, AIMSA 2008 Varna, Bulgaria, September 4-6, 2008 Proceedings.* pages 375-379. 4th-6th September 2008. Varna, Bulgaria.

Chapter 7

David Lee, Ines Arana, Hatem Ahriz and Kit-Ying Hui, 2009. Multi-Hyb: A Hybrid Algorithm for Solving DisCSPs with Complex Local Problems. In: *Proceedings of 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2009)* pages 379-382. 15th-18th September 2009. Milan, Italy.

David Lee, Ines Arana, Hatem Ahriz and Kit-Ying Hui, 2009. A Hybrid Approach to Solving Coarse-grained DisCSPs. In: *Proceedings of the Eighth International Conference on Autonomous Agents and Multi Agent Systems (AAMAS 09)* pages 1235-1236. 10th-15th May 2009. Budapest, Hungary.

Contents

List of Figures

List of Tables

List of Abbreviations

ABT	Asynchronous Backtracking
AWCS	Asynchronous Weak Commitment Search
CPA	Current Partial Assignment
CSP, CSPs	Constraint Satisfaction Problem(s)
DBHyb	Weight-Based Distributed Hybrid Algorithm
DisBO	Distributed Breakout
DisBO-wd	Distributed Breakout with Weight Decay for Agents with Multiple Local Variables
DisBOBT	Distributed Breakout combined with Backtracking
DisBOBTWD	Distributed Breakout with Weight Decay combined with Backtracking
DisBOCBJ	Distributed Breakout combined with Conflict-Directed Backjumping
DisBOCBJWD	Distributed Breakout with Weight Decay combined with Conflict-Directed Backjumping
DisCSP, DisCSPs	Distributed Constraint Satisfaction Problem(s)
DisHyb	Distributed Knowledge-Based Hybrid Approach
DisPeL	Distributed Penalty Driven Search
DisPeL-1C	Distributed Penalty Driven Search imposing penalties after a single cycle of no improvements.
InterDisPeL	Distributed Penalty Driven Search with Multiple Local Variables for considering only inter-agent constraints with dynamic domains.
InterPODS	Distributed Systematic Search with Multiple Local Variables for considering only inter-agent constraints with dynamic domains.
Multi-ABT	Asynchronous Backtracking for Agents with Multiple Local Variables

Multi-AWCS	Asynchronous Weak Commitment Search for Agents with Multiple Local Variables
Multi-DisPeL	Distributed Penalty Driven Search with Multiple Local Variables
Multi-Hyb	Hybrid Framework for Agents with Multiple Local Variables
Multi-Hyb-DB	Weight-Based Hybrid Algorithm for Agents with Multiple Local Variables
Multi-Hyb-Pen	Penalty-Based Hybrid Algorithm for Agents with Multiple Local Variables
Multi-HDCS	Hybrid Distributed Concurrent Search Framework for Agents with Multiple Local Variables
Multi-HDCS-DB	Weight-Based Hybrid Distributed Concurrent Search Framework for Agents with Multiple Local Variables
Multi-HDCS-Pen	Penalty-Based Hybrid Distributed Concurrent Search Framework for Agents with Multiple Local Variables
NCCCs	Non-concurrent Constraint Checks
PenDHyb	Penalty-Based Distributed Hybrid Algorithm
SBT	Synchronous Backtracking
SEBJ	Synchronous Exhaustive Backjumping for Non-interchange Solutions to Complex Local Problems
SingleDB-wd	Distributed Breakout with Weight Decay
Stoch-DisPeL	Stochastic Distributed Penalty Driven Search
SynCBJ	Synchronous Conflict-Directed Backjumping
SynCBJ-CLP	Synchronous Conflict-Directed Backjumping for DisCSPs with Complex Local Problems

Chapter 1

Introduction

Constraint Satisfaction, an artificial intelligence technique, solves problems containing **variables**, a set of potential values for each of these **variables** (**domains**) and **constraints** restricting simultaneous value combinations between connected **variables**. The notion of **Constraint Satisfaction** permeates everyday living. For example, the clothes that you wear on a particular day are dependant on the clothes that you have in your wardrobe and the matching combinations of clothes (e.g. a tie cannot be worn with trainers). The variables would be the garments needed (e.g. trousers, shirt, socks), the domain would be the clothes available (e.g. suit trousers, t-shirt, tie, black socks, trainers) and the constraints would be valid or invalid combinations of garments (e.g suit trousers cannot be worn with trainers).

A computing agent is a process which is authorised to act on behalf of others. For example, an agent may be a stockbroker who is authorised to deal in a number of shares for a customer. **Constraint Satisfaction Problems** (CSPs) may often be distributed between several agents possibly to maintain privacy between participants (agents) in the problem or because of the cost of gathering all information centrally. As a result, no agent has enough information to solve the problem by itself. **Distributed Constraint Satisfaction** (DisCSPs) extends CSPs for distributed problems among several agents (for example, geographically dispersed computers via the Internet). Each agent in a **Distributed Constraint Satisfaction** problem represents a **constraint satisfaction problem** consisting of **variables**, **domains** and **constraints**. For example, a simple timetabling CSP may contain modules, lecturers and students (**variables**), times and rooms (**domain**) and restriction on lecturer and student availability (**constraints**) can be represented as a DisCSP when it involves multiple schools. Each school would be represented by an agent in DisCSPs. For example, a Business Computing course in the Computing school will contain modules run by the Computing school but also may contain modules run by the Business school. In this case, the Computing school would timetable its modules but would have to take into account the Business school module timetables to ensure that modules taken by students of the Business Computing course did not clash. The main challenges in **Distributed Constraint Satisfaction** are to utilise the information available to each agent efficiently in order to find solutions while incurring low computations and communication costs.

1.1 Research Objectives

Existing methods for solving CSPs can in general be classified as backtracking and local search algorithms. **Backtracking algorithms** take a systematic approach to search and consequently are guaranteed to find a solution if one exists, although they may take exponential time to do so. In addition, they are guaranteed to discover that a problem has no solution when a problem is unsolvable. **Local search algorithms** may converge quicker to a solution, but are not guaranteed to find a solution if one exists and cannot determine that a problem has no solution. Some authors have developed algorithms which combine these approaches into a **hybrid approach** to overcome the individual weaknesses of each approach. The vast majority of these approaches have been developed for centralised problems with very few approaches being developed for distributed settings.

In this study, we seek to investigate, propose and evaluate hybrid algorithms for Distributed Constraint Satisfaction. Our primary aim is to speed-up distributed problem solving through using local search as a learning tool which can be used to guide backtracking. In particular, we are interested in **naturally distributed problems** which consist of large complex local problems which are sparsely connected. Our research objectives are therefore as follows:

1. Investigate techniques for making local search complete.

2. Making systematic search faster through the use of local search information.

3. Take advantage of agent idle time in order to carry out additional computation and thereby minimise overall problem cost.

1.2 Key Contributions

This work contributes a number of new techniques for solving Distributed Constraint Satisfaction. Our primary contribution is a *knowledge-based hybrid framework for DisCSPs*. In this framework, distributed local search is used to gather information about difficult variables prior to or at the same time as distributed systematic search. Distributed systematic search can then use this information as a heuristic to potentially find a solution quicker. Specifically, we contribute three new hybrid approaches:

1. *DisHyb* is a fine-grained hybrid approach, suitable for DisCSPs with one variable per agent, running distributed local search to learn about the difficult variables in the problem and potentially the best values to assign to them. If local search is unable to find a solution, distributed systematic search is run which is guided by the knowledge learnt from local search.

2. *Multi-Hyb* is a hybrid approach for DisCSPs with complex local problems (several variables per agent). For each agent, *Multi-Hyb* runs a centralised systematic search to find all local appropriate solutions (partial solutions) for its complex local problem concurrently for each agent. Whilst this search is ongoing, a distributed local search attempts to combine these partial solutions for each agent into a global solution. In addition, distributed local search learns knowledge about difficult complex local problems and good value combinations. If distributed local search cannot find a global solution once all local solutions for each agent have been found, distributed systematic search is run. This systematic search uses the partial solutions generated

by centralised systematic searches and the knowledge learnt from distributed local search.

3. *Multi-HDCS* is a second hybrid approach for DisCSPs with complex local problems. *Multi-HDCS* uses centralised systematic search per agent to find all local appropriate solutions (partial solutions) for it's complex local problem (as *Multi-Hyb* does). However, Multi-HDCS runs a distributed local search and a distributed systematic search concurrently. These distributed searches run whilst the centralised systematic searches are finding solutions to their local problem. The distributed searches attempt to combine these partial solutions into a global solution. The distributed local search regularly synchronises information about difficult complex local problems and values to guide the distributed systematic search.

1.3 Scope of Study

This study principally focuses on Distributed Constraint Satisfaction where the objective is to find only the first solution which satisfies all constraints simultaneously. There may however be multiple solutions to a Distributed Constraint Satisfaction problem. The hybrid algorithms presented in this thesis could keep running to find more solutions. Our algorithms are also not specifically designed for Dynamic Distributed Constraint Satisfaction where the problem specification may change during the problem solving process. In the event that the problem specification changes, our algorithms must be re-run to find a solution to the updated problem specification.

1.4 Thesis outline

This thesis is presented as follows. Chapter 2 presents a formalisation for four particular problem types for Distributed Constraint Satisfaction which we will consider throughout this thesis. Chapter 3 presents a survey of the state-of-the-art algorithms for Constraint Satisfaction Problems in centralised environments. Chapter 4 extends this survey for Distributed Constraint Satisfaction. Chapter 5 presents *DisHyb*, our knowledge-based hybrid approach for single variable per agent algorithms. Chapter 6 presents *Multi-Hyb*, a two-phase hybrid approach for solving DisCSPs with complex local problems whilst chapter 7 presents a second approach entitled *Multi-HDCS*. In chapters 5, 6 and 7, extensive empirical evaluations are presented for each of our contributions. A thesis summary and interesting avenues of future work are proposed in Chapter 8. A glossary of terms is provided at the end of the thesis.

Chapter 2

Problem Formulisation

2.1 Introduction

In this chapter, we formally define **Distributed Constraint Satisfaction Problems** (DisCSPs). This definition is aided by a brief description of **Constraint Satisfaction Problems** (CSPs). In particular, we present four different types of DisCSPs: (i) **randomly generated**; (ii) **graph colouring**; (iii) **meeting scheduling** and; (iv) **sensor networks**. The reader is referred to chapter 3 for a description of search algorithms for CSPs and chapter 4 for a description of search algorithms for DisCSPs.

2.2 Distributed Constraint Satisfaction

A Constraint Satisfaction Problem (CSP) [21] is a tuple (V, D, C) where: $V = \{v_1, v_2, ..., v_N\}$ is a set of N variables in the problem, $D = \{Dom(v_1), Dom(v_2), ..., Dom(v_N)\}$ is a set of N domains - one domain per variable and $C = \{c_1, c_2, ..., c_P\}$ is a set of P constraints between variables in the problem. A Distributed Constraint Satisfaction problem (DisCSP)[91, 94, 97] is a tuple (A, V, D, C) where: $A = \{a_1, a_2, ..., a_M\}$ is a set of M agents, for each agent a_i, a set $V_i = \{v_{i1}, v_{i2}, ..., v_{in}\}$ of variables it represents such that $\forall i \neq j \; V_i \cap V_j = \emptyset; V = \bigcup V_i$ is the set of all variables in the DisCSP, $D = \{Dom(v_1), Dom(v_2), ..., Dom(v_N)\}$ is the set of N domains - one for each variable and $C = \{c_1, c_2, ..., c_P\}$ is a set of P constraints between variables. The set of constraints

(C) can be separated into two independent subsets: C_{intra} is the set of **intra-agent constraints** between variables belonging to the same agent whilst C_{inter} is the set of **inter-agent constraints** between variables belonging to different agents. In order to simplify the problem, a common assumption in the field is that each agent represents a single variable [97]. In this case, all constraints belong to the set of inter-agent constraints (C_{inter}). However, many DisCSPs would be more naturally expressed and formulated through having more than one variable per agent. Single variable per agent algorithms can be used for multiple variable per agent problems by either: (i) solving the local problems within each agent first and creating a **complex variable** for that agent which has the number of solutions to its local problem as its domain (compilation); (ii) making each variable in the local problem into a virtual agent [13]. In chapter 5, we will use this assumption of a single variable per agent. We will relax this assumption to deal with **DisCSPs with Complex Local Problems** where agents have more than one variables in chapters 6 and 7.

Yokoo et al. [97] also assumed that all constraints in the problem are binary (between two variables) and that each agent knows about all of the constraints involving its variable(s). We also make these assumptions. All CSPs involving non-binary constraints can be transformed into a CSP with only binary constraints [4] and whilst there is a substantial cost associated with this transformation [9], it is not normally counted by researchers. With relation to messages, we assume that agents can communicate with a particular agent if they know their address (i.e. share a constraint with one of its variables) and that messages between pairs of agents arrive in the order that they were sent in finite time [94].

2.3 Problem Areas

Four different types of DisCSPs are now described: **Randomly Generated Problems**, **Graph Colouring Problems**, **Meeting Scheduling Problems** and **Sensor Network Problems**. For each problem, we present two formulisations: (i) for a single variable per agent DisCSP; (ii) for a DisCSP with Complex Local Problems. The first formulisation will be used in chapter 5 whilst the other formulisation will be used in chapters 6 and 7.

2.3.1 Randomly Generated Problems

A randomly generated DisCSP is an example of a homogeneous unstructured problem. These problems have a number of variables with a fixed domain. Variables belonging to constraints are chosen at random. Specifically, we generated both solvable and unsolvable randomly generated DisCSPs using the Model-B method [66]. These problems had **one variable per agent** so all constraints are between variables belonging to different agents (inter-agent constraints). Specifically, a tuple $< n, d, p1, p2 >$ was used to generate where n is the number of variables, d is the domain size of all variables, $p1$ is the constraint density and $p2$ is the constraint tightness. The variables involved in constraints were chosen at random as was the restriction of certain value combinations of the variables involved in the constraint. We used binary constraints with the constraint density controlling how many constraints were generated and the constraint tightness determining the proportion of value combinations forbidden by each constraint. For example, a constraint density of 0.2 would generate 20% of the possible constraints in the problem (i.e. $(n*(n-1)/2)*0.2$ where n is the number of variables) and a constraint tightness of 0.4 would prevent 40% of the possible value combinations of variables involved in a constraint from satisfying the constraint. The Model-B method was modified to include preferential assignment of constraints to variables so that they resemble real-life problems [90] in a similar way as [8] for non-binary DisCSPs.

For **DisCSPs with Complex Local Problems**, we partitioned the variables to agents so that constraints involving variables would become either intra-agent or inter-agent constraints depending on whether both variables were within the same agent (making it an intra-agent constraint). We made this partition so that there was an imbalance between the number of constraints within an agent (intra-agent constraints) and those between agents (inter-agent constraints) such that the former had 70% to 90% of the total number of constraints to create naturally distributed problems.

2.3.2 Graph Colouring Problems

Graph colouring is a popular problem for DisCSPs to solve [8] since many problems can be transformed into a graph colouring problem. Specifically, we want to colour the nodes in a graph so that no two connected nodes share the same colour. We generated both solvable and unsolvable graph colouring problems using the method described in [29] for **one variable per agent**. Specifically, we have a tuple $< n, c, d >$ where n is the number of nodes in the graph, c is the number of colours available and d is the connectivity of the graph (determining the number of edges and therefore the number of constraints in the graph). For **DisCSPs with Complex Local Problems**, we generated problems using the partitioning method described in [46] ensuring that the generated graphs had a higher proportion of intra-agent to inter-agent constraints.

2.3.3 Meeting Scheduling Problems

This study also considers **structured problems** in the form of meeting scheduling problems. In meeting scheduling problems, a number of meetings must be scheduled involving a number of participants. Some of these meetings must be scheduled before others. Participants may belong to different departments and so they must have sufficient travelling time between meeting. We developed a generator based on Brito's meeting scheduling generator [11]. Each department (agent) holds a number of meetings (variables). A set of times make up the domain of each meeting. The attendee list for a meeting can contain employees within the department and employees outwith the department. Each department has at least one location where meetings can be held and employees from another department can attend meetings in that department provided they can arrive on time. A distance chart between locations is randomly generated so that the distance between two locations is assigned a value between 0 and the maximum possible distance indicating the travelling time required. There are three types of constraints: (i) **difference constraints** between all meetings held in the same department; (ii) **travelling time constraints** between inter-departmental meetings with one or more common participants (for example, if a participant had a meeting at 9am and the travelling time to the next meeting was

2 hours, the next meeting involving that participant could not take place until 12noon); (iii) **precedence constraints** between meetings.

When generating **DisCSPs with Complex Local Problems**, the ratio of intra-agent to inter-agent constraints varied between 70:30 and 90:10.

2.3.4 Sensor Network Problems

Sensor networks is an example of a pratical application of multi-agent technology [100]. We used the Grid-based SensorDCSP generator described in [100]. Specifically, we wish to assign 3 sensors to track each target. There is a limited pool of sensors which can view particular targets (defined as **visibility**) and only some of these sensors can be positioned to ensure a triangle is formed around the target (defined as **compatibility**). For example, there are two targets t_1 and t_2 with six sensors $s_1, s_2, s_3, s_4, s_5, s_6$ in a sensor problem. The visibility may be that s_1, s_4, s_5, s_6 are capable of tracking t_1 whilst sensors s_2, s_3, s_4, s_5 are capable of tracking t_2. The compatibility would then refer to the positioning of these sensors in relation to the targets. Therefore, each agent (representing a target) has 3 variables (representing the sensors that are tracking the specific target). The variable value is the sensor that is selected to perform the task of tracking that target in the particular position of the triangle (for example, s_4 may be chosen to be the 2nd sensor tracking t_1 so that variable 2 of agent 1 would have a value of 4). These problems were specifically chosen because they have a high ratio of inter-agent to intra-agent constraints in contrast with the problems previously described in this chapter. Indeed the ratio of inter-agent to intra-agent constraints is 85:15.

2.4 Summary

In this chapter, four particular types of problems which can be represented as Distributed Constraint Satisfaction problems have been presented: randomly generated DisCSPs, graph colouring, meeting scheduling and sensor networks. With the exception of sensor networks, we have shown that the problems can be represented with a single variable per agent or with multiple variables per agent. In the next chapter, we must first of all consider algorithms for solving Constraint Satisfaction Problems before we can consider algorithms for solving Distributed Constraint Satisfaction Problems in chapter 4.

Chapter 3

Constraint Satisfaction

3.1 Introduction

There are many problems in everyday life which involve constraints. For example, the clothes we choose to wear each day are determined by the clothes that we have in our wardrobe. Depending on the amount of available options (in this case, the size of the wardrobe), it may not be easy to find a solution as to which clothes to wear assuming that we want to match clothes so that we do not wear conflicting colours. This particular issue of finding a solution to a problem has been a major focus of research in the Artificial Intelligence community. Consequently, an area of research has emerged into **Constraint Satisfaction Problems** (CSPs). Dechter [21] defines a CSP as a triple (X, D, C) where $X = \{x_1, ..., x_n\}$ is a set of **variables**, $D = \{D_1, ..., D_n\}$ is a set of **domains**, one per variable, and C is a set of **constraints** which restrict the values that variables can take simultaneously. A formal definition for CSPs was given in section 2.2. A **solution** to a CSP is defined as an assignment of a value from its domain to each variable so that all constraints are satisfied [7]. A value can be assigned to each variable so that each variable's constraints are satisfied by attempting different combinations of values for variables through a **search algorithm**. In this chapter, we introduce methods for the resolution of CSPs, namely the **constraint propagation** technique and two classes of search algorithms: (i) **systematic search algorithms**; (ii) **local search algorithms**.

3.2 Definitions

Prior to introducing the algorithms for solving CSPs, a number of formal concepts must be introduced.

A variable x_i is said to be a **neighbour** of a variable x_j if variable x_i shares a constraint with x_j.

Variable x_i's **neighbourhood** is the set of all variables $N = \{n_1, ..., n_m\}$ which share a constraint with x_i.

An **improvement** to a variable is the assigning of another value to that variable which lowers the number of constraint violations it is involved in.

A neighbourhood is said to be in **local optima** if there are no improvements which can be made to any of the variables in the neighbourhood.

Two variables are **connected** if they share a constraint.

3.3 Constraint Propagation

Constraint Propagation is a technique which removes values from each variable's domain that cannot satisfy the variable's constraints. Constraint propagation increases in complexity from node consistency (removes values based on constraints involving only that variable), arc consistency (removes values based on constraints involving pairs of connected variables) to path consistency (ensuring values satisfy all binary constraints in a path between two variables). The k-consistency of a CSP is defined as a CSP where each (k-1) tuple can be extended to a k compatible tuple which satisfies constraints. Consequently, node consistency corresponds to 1-consistency with arc consistency to 2-consistency. However, constraint propagation may not remove all inconsistent values [7] so the potential search space may remain large [8]. Frequently, propagation is used as a pre-processing technique for a search algorithm (see below).

3.4 Systematic Search Algorithms

This family of algorithms takes a systematic approach to looking for solutions in the entire search space. For example, consider the simple problem illustrated in figure 3.1. Assume that you would like to timetable 3 modules in your department. The modules (WEB, OOP, $DATABASES$) have to be assigned a time so that no two modules are timetabled at the same time. In addition, there must be a two hour break between the WEB and $DATABASES$ modules so that the absolute difference between WEB and $DATABASES$ is greater than or equal to 2 (i.e. $|WEB - DATABASES| >= 2$). The WEB module has four times available (11am, 1pm, 2pm, 3pm) whilst the other modules have two possible times but the times are different for each module (10am and 12pm for OOP and 10am and 11am for $DATABASES$ respectively). Formally, this problem can be modelled as a CSP as $X = \{WEB, OOP, DATABASES\}, D_{WEB} = \{11am, 1pm, 2pm, 3pm\}, D_{OOP} = \{10am, 12pm\}, D_{DATABASES} = \{10am, 11am\}, C_1 = [WEB \neq OOP], C_2 = [DATABASES \neq OOP], C_3 = [|WEB - DATABASES| >= 2]$.

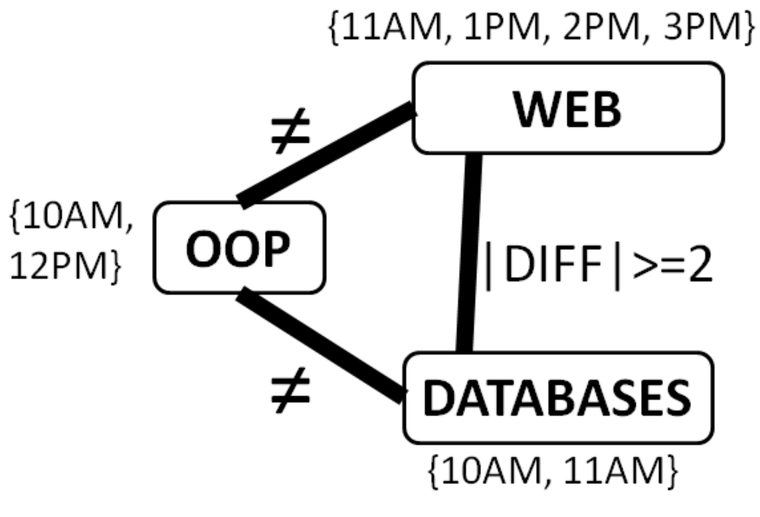

Figure 3.1: A simple Constraint Satisfaction Problem

The simplest algorithm in this backtracking family is the **Naive Backtracking** algorithm [87].

Initially, all variables are unassigned. For our sample problem, it is assumed that the variables are ordered e.g. WEB, OOP and $DATABASES$ and that values are chosen in

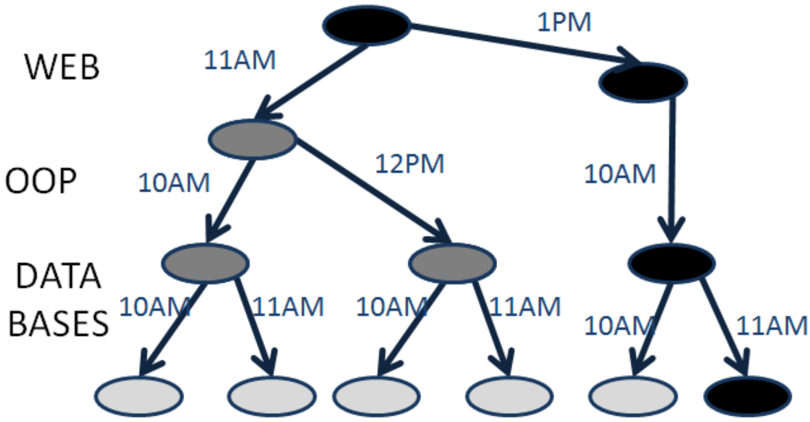

Figure 3.2: The Naive Backtracking search tree for our simple CSP

the order that they appear in the domain. The search tree produced by Naive Backtracking when solving this problem is shown in figure 3.2 where black circles indicate the solution and grey circles indicate value combinations that do not lead to a solution. The algorithm then selects a variable (e.g. the WEB variable) and assigns a value to that variable which satisfies all constraints. Then the next variable (e.g. the OOP variable) is selected and assigned a value consistent with the constraints, and with the value already assigned to previous variables (e.g. the WEB variable). This process is repeated until all variables are assigned, i.e. a solution to the problem is found. However, if a variable cannot be assigned a value without violating constraints, the algorithm backtracks to the previous variable, choosing another value for this variable which does not violate constraints. For example, in the sample problem, the $DATABASES$ variable has no consistent value whilst the WEB variable has value 11am. This will result in the algorithm initially backtracking to the OOP variable and exhausting all possible values for that variable before backtracking to the WEB variable and changing its value.

The Naive Backtracking algorithm attempts all variable value combinations in the worst case [21]. Consequently, a number of improvements to this algorithm have been proposed. **Backjumping** [33] maintains a backjumping variable which is a neighbour (i.e. shares a constraint with) of the current processing variable and is the lowest variable higher in the search ordering than the current processing variable. When the current processing variable has no consistent values, the algorithm tries to find a new value for this backjumping variable rather than the previous variable in the search ordering.

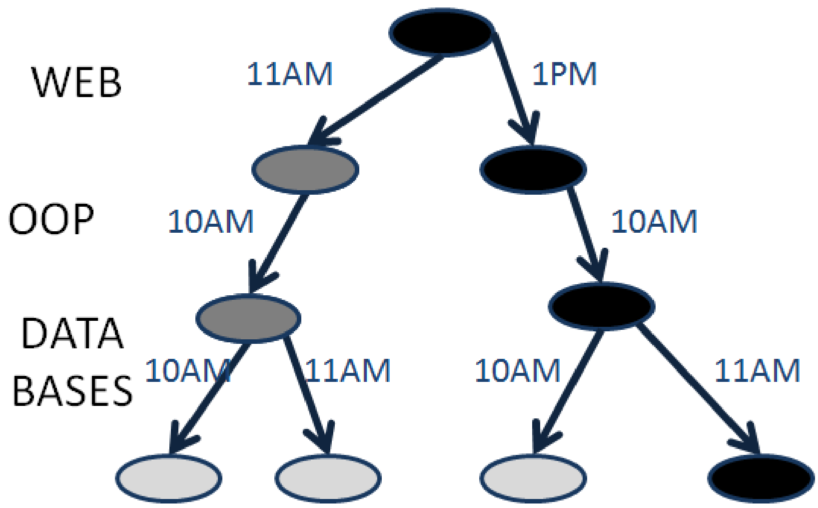

Figure 3.3: The Backjumping search tree for our simple CSP

For example, the sample problem (in figure 3.1) would produce the search tree shown in figure 3.3 where black circles indicate the solution and grey circles indicate value combinations that do not lead to a solution. Specifically, the search would start with WEB being assigned 11am as its first domain value. This is consistent with all constraints since there are no constraints to check. OOP would be assigned 10am since this satisfies the constraint of $OOP \neq WEB$. We now instantiate the $DATABASES$ variable but there are no consistent values since 10am and 11am violate the constraint between $DATABASES$ and WEB [1]. In Naive Backtracking, we would backtrack to the OOP variable as it is the previous variable in the search. In Backjumping, we would backjump to the WEB variable as this is the variable which is causing $DATABASES$ to be unable to choose a value i.e. changing the value of OOP cannot improve the situation. If the backjump variable also has no consistent values, then naive backtracking is used to backtrack to the previous variable.

Conflict-Directed Backjumping [74] is an improvement over backjumping in which a conflict set of possible backjumping variables is maintained. This enables repeated backjumping with the deepest variable in the conflict set being chosen when a backjump

[1] Incidentally, 10am violates the constraint between OOP and $DATABASES$ whilst 11am satisfies this constraint.

is required and that variable then being removed from the conflict set. This conflict set takes the form of nogoods where variable assignments which are found not to lead towards a solution are recorded.

Dynamic Backtracking [34] is a further improvement to Conflict-Directed Backjumping which allows variable re-ordering and maintains the search information after backjumping which has not been invalidated by the backjump. Other improvements to backtracking include backmarking [33] and forward checking [3].

Forward checking, which can be combined with any systematic search algorithm, checks if unassigned neighbours of the current processing variable have any consistent values with the proposed assignment of the current processing variable. If not, the proposed assignment for the current processing variable is abandoned and another assignment is chosen. For example, in the sample problem, forward checking would discover when assigning 11am to WEB that there is no consistent value for $DATABASES$ when WEB takes 11am as its value and so would cause WEB to choose 1pm before moving to assign OOP as the next variable in the search. This prevents the exploration of parts of the search tree that are doomed to failure because of a particular variable value combination (as WEB being set to 11am was causing the problem).

Systematic search algorithms are **complete** in that they are guaranteed to find a solution or determine that the problem is unsolvable. However, they may take exponential time to do this.

3.5 Local Search Algorithms

A second family of search algorithms are called local search algorithms. These algorithms begin with an initial value chosen (possibly at random) for every variable. Dechter [21] presents the **simplest stochastic local search algorithm**, generating a new random initialisation for all variables for a maximum number of tries. Most local search algorithms uses an iterative repair technique on the variables whose values lead to violated constraints. This heuristic-based repair technique guides the search towards a possible solution. For example in the simple problem in figure 3.1 above, the first complete assignment to be generated may be $SOL = \{WEB = 11am, OOP = 10am, DATABASES = 10am\}$ which produces two constraint violations (i.e. $WEB \neq OOP$ and $|WEB - DATABASES| >= 2$). The local search may then discover that by changing the value of OOP to 12pm (i.e. $SOL = \{WEB = 11am, OOP = 12pm, DATABASES = 10am\}$) only one constraint is now violated ($|WEB - DATABASES| >= 2$).

Stochastic local search algorithms may get stuck on a local optima. Consequently, the algorithm always finds a particular complete assignment more desirable than trying any other combination of values even if this complete assignment is not a real solution (i.e. some constraints remain violated) [21]. Since most local search algorithms do not allow multiple variables which are neighbours to change value at the same time, the algorithm will not see any possible changes that reduce the number of constraint violations. For example, the local search algorithm may not be able to find any improvements by changing a single value in our example above to reduce the constraint violations to 0 (i.e. satisfy the constraint $|WEB - DATABASES| >= 2$). However, it may be that by moving to another complete assignment with the same number of constraint violations or temporarily increasing the constraint violations, will ultimately lead to a solution. Consequently, local search approaches have been improved through a number of new techniques.

Hill-climbing [59] uses heuristics to choose which variables are allowed to change their values and thereby "climb the hill" to the problem solution. The search is restarted with different values if the search gets stuck in a local optimum.

Tabu search [36] is an alternative approach where certain combinations of values are temporarily banned forcing the algorithm to search a different area of the problem search space.

Simulated annealing [51] allows moves which increase the number of constraint violations for variables with a higher initial probability that descends over time.

Guided local search [89] uses penalties on values for variables which violate constraints to guide the search to more promising areas of the search space.

Variable neighbourhood search [42] focuses the search on a particular search space area for each run.

In the **breakout algorithm** [62], a **weight** of 1 is assigned to each constraint. This weight is increased if the constraint is violated. The summation of these weights is added to the constraint violations when evaluating how 'bad' a solution is. If a neighbourhood is in local optima, the weights are increased so that the neighbourhood has a higher weight than others and so can escape from local optima. This increase in weights is called **'breakout'**. Whilst recent work on satisfiability problems (a subset of constraint satisfaction problems) has shown that local search may determine why a problem is insolvable [38], local search remains **incomplete** [2].

3.6 Variable and Value Ordering Heuristics

Choosing the next variable or value in backtracking search [5] or the neighbourhood to explore for some local search approaches [43] is crucial to obtain good performance. Variable ordering heuristics include **maximum degree** (most heavily constrained variables preferred) and **maximum cardinality** (first variable randomly selected then choose the variable connected to most assigned variables) [58]. Search rearrangement [58] (often referred to as **min-domain**) chooses to assign the variable which has the fewest number of remaining consistent values with already instantiated variables. Value ordering heuristics control the order in which the values are chosen after variable ordering. **Min-conflicts** [59] is a value ordering heuristic choosing the value which minimises violations with neighbouring variables.

3.7 Problem Decomposition

There have been a number of approaches which attempt to decompose the problem into easier subproblems which can be joined together to form a solution to the original problem. Anand et al. [1] use a lazy evaluation algorithm to divide the CSP into subproblems since they argue it is very difficult to cleanly split a CSP. Earlier, Freuder and Hubbe [31] attempted to extract unsolvable subproblems to therefore prove global unsolvability.

[2] The algorithms may not find a solution to a solvable problem and cannot determine an unsolvable problem.

These approaches usually make use of either backtracking or local search algorithms to solve the subproblems.

3.8 Hybrid Algorithms

Recently, research has focused on the combination of systematic and local search algorithms in order to overcome the perceived individual weakness of each [48]. Table 3.1 summarises the properties of backtracking algorithms (see section 3.4) and local search algorithms (see section 3.5).

	Backtracking Algorithms	**Local Search Algorithms**
Theoretical Completeness	Complete	Incomplete
Search Strategy	Systematic sequential assignment	Iterative repair of initial assignment
Scalability	Small problems	Larger problems
Convergence	Very Slow	Quicker (for larger problems)

Table 3.1: Contrasting the properties of backtracking algorithms with local search algorithms

As illustrated in Table 3.1, backtracking and local search algorithms have complementary advantages. Whilst theoretically complete, backtracking algorithms often converge so slowly that they cannot practically be used with large problems. Local search algorithms replace backtracking's sequential assignment with iterative repair of values resulting in incompleteness. However, local search often converges quicker to solutions for large problems.

Many authors have presented **hybrid approaches** combining backtracking and local search. Jussien [48] classified these into three categories: (i) **local search then systematic search**, or vice versa; (ii) **systematic search using local search** at some point; (iii) **local search using systematic search** for neighbour selection or search pruning. We describe these categories below, making minor modifications to these categories so all hybrid approaches are classified.

3.8.1 Local Search Before/After Systematic Search

This category contains algorithms either running the two search approaches consecutively or running similar amounts of backtracking and local search. Table 3.2 briefly describes each of these approaches.

Reference	Description
Caseau and Laburthe [14]	Interleaving constructive (backtracking) and local search produces optimised solutions over running constructive then local search.
Eisenberg [22]	Presents the **BOBT algorithm** which runs local search algorithm for a number of breakout steps. When this number is exceeded, systematic search is run using the weights from the breakout (local search) algorithm [62].
Schaerf [80]	Produces a framework for combining backtrack and local search using backtracking until no consistent value and then local search to determine the best values to resolve situation.
Zhang and Zhang [99]	A specified number of variables is initialised in the partial assignment. A higher number indicates more local search whereas a small number indicates more backtracking.

Table 3.2: Hybrid algorithms running similar amounts of backtracking and local search or running one after the other.

Caseau and Laburthe argue that interleaving should be done through backtracking and then local search rather than vice versa whilst Zhang and Zhang provide an approach that enables the amount of local search and backtracking to be controlled for different problem types. Eisenberg's approach is similar to Zhang and Zhang except Eisenberg uses the breakout algorithm [62] (see section 3.5 for details of the breakout algorithm). The duration of the breakout algorithm is determined by the number of times the weights are increased (i.e. the number of breakout steps performed). In Zhang and Zhang, the number of variables initialised determines the boundary of local search. Schaerf's approach of backtracking then local search is contrasted with Zhang and Zhang who run local search on the initial partial assignment extending it through backtracking [78]. Jussien's Decision-Repair algorithm [48] generalises Schaerf's approach [73, 71] which Jussien classifies in the "Local Search with Systematic Search during Search" category. Consequently, we could categorise Schaerf's approach in this third category of Jussien's classification, but as the approach runs local search when backtracking reaches a dead-end, the amount of local search run will depend on how many dead-ends backtracking occurs. As such, this approach could be categorised in either this first category or the third category. Since the amount of local search and backtracking depends on the problem, we choose to leave it in this category. Whilst Schaerf's and Zhang and Zhang's approaches are well cited in literature, the exclusive use of propositional satisfiability (SAT) problems by Zhang and Zhang with boolean values, disputes the overall applicability of their approach.

3.8.2 Systematic Search using Local Search

Jussien's second category contains backtracking (systematic search) algorithms incorporating local search to converge quicker to a solution whilst often losing completeness. We extend this category to all hybrid approaches having clear systematic origins. Table 3.3 describes these approaches.

Whilst Prestwich integrates multiple local search properties to backtracking sacrificing completeness, Ginsberg and McAllester, Gomes et al., Richards and Richards and Yokoo are all able to implement local search techniques whilst maintaining completeness. The key to maintaining completeness in these approaches is keeping a systematic record of the values considered. Other approaches such as Kamarainen and Sakkout, Mazure et al., Nareyek et al., Sakkout and Wallace and Yoshikawa et al. use local search as the decision-maker to choose the next variable or value. This needs to be used with caution as if it is used too frequently, the costs of running local search outweighs the benefits. Hogg and Williams' approach of cooperative search is particularly interesting given the increased processing power which is now available that permits search algorithms to run in parallel.

3.8.3 Local Search with Systematic Search during search

Jussien's final category includes local search algorithms using systematic search properties for search space pruning or choosing the candidate neighbour (the variable that is allowed to change its value). Table 3.4 summarises these approaches.

Cotta's work is based on constraint optimisation problems where an objective function

Reference	Description
Caseau et al. [15]	Explores parameters for hybrid variants of systematic search algorithms for vehicle routing problems.
Ginsberg and McAllester [35]	Presents **Partial-order dynamic backtracking algorithm** offering more flexibility than dynamic backtracking in choosing search path but maintaining completeness.
Gomes et al. [37]	Presents a general framework for including randomisation in complete search to speed-up run time.
Hogg and Williams [47]	Introduces notion of hints between algorithms to form a cooperative search technique.
Kamarainen and Sakkout [49]	Describes **local probing algorithm** using local search to solve easy constraints initially and then systematic search for all remaining constraints. Local search is essentially used to instantiate variables belonging to easy constraints.
Mazure et al. [55]	Presents **DP+TSAT** algorithm for SAT problems. During a backtracking search, guided local search determines the next variable to instantiate.
Nareyek et al. [63]	Uses systematic search (termed refinement search) with local search acting as a guiding heuristic for each search decision. As a consequence of running local search so frequently, the technique is outperformed by simpler heuristics.
Prestwich [71, 73, 72]	Introduces an **Incomplete Dynamic Backtracking algorithm** using local search, forward checking and dynamic variable ordering. Also **Constrained Local Search algorithm** determining how far to backtrack heuristically.
Richards and Richards [75]	**Learn-SAT algorithm** maintaining completeness using systematic restarts rather than backtracking.
Sakkout and Wallace [79]	Presents the **Unimodular probing algorithm** where good values are selected to guide the backtracking search and minimise search effort (reminiscent of choosing neighbouring values to change in local search).
Yokoo [92]	Presents the **weak-commitment search algorithm** where tentative values in partial assignment revised using min-conflicts heuristic until dead-end then restart with new assignment. Nogoods storing old partial assignments maintain completeness.
Yoshikawa et al. [98]	Presents **SchoolMagic** combining arc consistency (systematic constraint propagation) with minimum conflicts heuristic for a high school scheduling problem.

Table 3.3: Hybrid algorithms with systematic origins using local search.

Reference	Description
Barnier and Brisset [6]	Combines power of genetic algorithms (local search) with systematic CSP techniques using a hybrid parameter similar to Zhang and Zhang [99].
Cotta et al. [16]	Presents **HEAGRASP** algorithm using local search to generate solutions for an optimisation problem then a systematic search technique to combine these solutions into an optimal solution.
Crawford [17, 18]	GSAT local search algorithm calculates weights for ordering of systematic search algorithm for SAT problems. **ISAMP algorithm** restarts when discovering a contradiction with a random variable.
David [19]	Uses arc consistency and iterative improvement step to produce sub-optimal solutions where time available did not allow for a full exhaustive search.
De Backer et al. [20]	Framework for local search containing some backtracking properties for vehicle routing problems.
Fang and Ruml [27]	Presents **complete local search (CLS)** algorithm framework for making local search complete for SAT problems using constraint learning and an objective function. Exponential space complexity.
Jussien and Lhomme [48]	**Decision-repair abstract algorithm** with local search over a partial assignment of variables. Instance of Schaerf's [80] framework.
Lever [52]	Presents **Full LS/CP Hybrid** where local search is used to determine a good bound for a branch-and-bound systematic search.
Mitra and rae Kim [60]	Presents **MC-FC algorithm** uses min-conflicts local search to solve the first part of the problem and forward checking to solve the remaining part.
Nowicki and Smutnicki [65]	Exchange of assignments extended to complete solution using constructive search techniques for job shop scheduling problems.
Pesant and Gendreau [67]	Combines local search and backtracking techniques for optimisation problems (finding the best solution rather than any solution).
Shaw [82]	Presents **Large Neighbourhood Search** which applies local search to tree-based systematic search for vehicle routing problems only.
Verfaillie and Schiex [88]	Presents **local changes algorithm** extending consistent partial assignment through local changes for dynamic CSPs (CSPs which have constraints added or retracted during problem resolution).

Table 3.4: Hybrid algorithms running overall local search with some systematic search properties.

determines criteria that forms the best (**optimal**) solution out of many solutions to the problem. Cotta's work indicates that local search comes close to the optimal solution but requires a systematic search to reach the optimal state. This applies to satisfaction problems when systematic search finds the solution based on local search's best values. Crawford's use of local search weights for backtracking is similar to Eisenberg's **BOBT** [22] but here it is only used as a tie-breaker for ordering whereas it is the primary method of ordering in BOBT. Crawford and Baker [18] conducted an experimental evaluation for SAT problems of local search (**GSAT**), backtracking (**TABLEAU**) and hybrid (**ISAMP**). ISAMP outperformed GSAT and TABLEAU, but was helped by a large number of not uniformally distributed solutions. Jussien and Lhomme's work offers completeness, but remains an abstract framework with little experimental evaluation. Fang and Ruml's work on **Complete Local Search** offers the first completeness guarantee for local search, with experimental evaluation, but only on propositional satisfiability (SAT) problems with boolean values as the domain. Additionally, their nogood store has exponential space complexity.

3.9 Limitations of Study

This study does not consider constraint optimization problems or dynamic constraint satisfaction. Unsolvable problems are only considered within the context of detecting that the problem is unsolvable.

3.10 Summary

In this chapter, we have introduced Constraint Satisfaction Problems (CSPs). We have shown that there are two main families of algorithms to solve CSPs: backtracking and local search algorithms. Many authors have combined elements of backtracking and local search algorithms into hybrid algorithms. Jussien classified these approaches into three categories which we have adapted to include all approaches: (i) local search before/after systematic search; (ii) systematic search using local search during search; (iii) local search with systematic search during search. In addition, we have explained other techniques which can be combined with these algorithms such as constraint propagation and problem decomposition. In the next chapter, we consider algorithms for solving Distributed Constraint Satisfaction Problems.

Chapter 4

Distributed Constraint Satisfaction

4.1 Introduction

Many problems such as scheduling and resource allocation problems are difficult to model using the Constraint Satisfaction techniques defined in previous chapter since they assume the whole problem is available to a single solver. Constraint Satisfaction Problems are now referred to as **centralised Constraint Satisfaction Problems**. For example, Faltings [25] highlights that arranging a two person meeting may impact on the participants' other meetings which would have to be included in the centralised CSP. A similar scenario for industry occurs between collaborating companies in different stages of the supply chain [25].

Yokoo et al. [91, 94, 97] proposed solving these multi-agent problems through a new approach called **Distributed Constraint Satisfaction problems** (DisCSPs). A formal defintion of DisCSPs was presented in chapter 2. Briefly, a DisCSP is a tuple (A, V, D, C) where: $A = \{a_1, a_2, ..., a_M\}$ is a set of M agents, for each agent a_i, a set $V_i = \{v_{i1}, v_{i2}, ..., v_{in}\}$ of variables it represents such that $\forall i \neq j V_i \cap V_j = \emptyset; V = \bigcup V_i$ is the set of all variables in the DisCSP, $D = \{Dom(v_1), Dom(v_2), ..., Dom(v_N)\}$ is the set of D domains - one for each variable and $C = \{c_1, c_2, ..., c_P\}$ is a set of P constraints between variables. The set of constraints C is split into two subsets - C_{inter} contains the **inter-agent constraints** between variables belonging to different agents and C_{intra} contains the **intra-agent constraints** between variables belonging to the same agent. Each agent controls only the variables which it represents and knows only the domains of those variables and the constraints they are involved in. Agents communicate through sending messages. These messages contain the proposed values for the sender's variables and are only sent to agents who have variables that share constraints with some of the sender's variables [1]. We discuss algorithms for solving DisCSPs which can be split into two categories. Some algorithms assume that each agent represents a single variable and these are discussed in section 4.2. Other algorithms for **DisCSPs with complex local problems** where agents represent several variables have been developed. These algorithms are described in section 4.3.

4.2 Distributed Constraint Satisfaction with One Variable per Agent

A large part of the research into DisCSPs has concentrated on the assumption that each agent represents a single variable. This section discusses these approaches.

Agents could send their information to a central agent [97] and then that agent uses centralised problem techiques (see chapter 3) to solve the problem and returns the solution to each agent. Mailler and Lesser [54] solve the hardest problem parts using centralised techniques whilst using distributed techniques for the remaining agents. However, information may be stored in different formats or privacy concerns exist which prevent these approaches [97]. Consequently, only distributed algorithms are now considered. These are evaluated in relation to two common metrics: (i) the **number of messages** passed between agents indicative of communication costs; (ii) the **number of non-concurrent constraint checks** performed [56] indicative of time taken. A good distributed algorithm will minimise these metrics.

[1] Yokoo et al. [94] assume finite message delay with messages between two agents arriving in sent order. We also make this assumption.

4.2.1 Distributed Constraint Propagation

We may attempt to solve a DisCSP using constraint propagation through Hamadi's DisAC-9 [39] or Ringwelski's DDAC4 [76] arc consistency algorithms. These approaches have the same drawbacks as those explained in section 3.3.

4.2.2 Distributed Backtracking

Distributed Backtracking algorithms can be divided into **synchronous algorithms** and **asynchronous algorithms**. In synchronous algorithms, agents perform actions in a pre-defined order whilst agents act concurrently with other agents in asynchronous algorithms. Whilst one may assume that the concurrent behaviour of asynchronous algorithms means that asynchronous algorithms will always be beneficial over synchronous algorithms, Brito and Meseguer [12] have shown this not always to be the case. Many backtracking algorithms use **nogoods** which record values for variables which have been attempted and do not lead a solution.

Synchronous Backtracking (SBT) [94] is the simplest distributed algorithm and is based on the Naive Backtracking algorithm [87]. Ordered agents sequentially instantiate their variable and send a message with a **consistent partial assignment** (CPA) containing all values assigned so far to the next agent. Agents send a backtracking message to the previous agent if no consistent value exists. A solution is found when every agent has a consistent instantiated variable.

Synchronous Backtracking has been improved by Zivan and Meisels [102] to **SynCBJ** which is a distributed version of conflict-directed backjumping [74] combined with dynamic backtracking [34]. SynCBJ [102] is a synchronous systematic search algorithm where each agent keeps track of the reasons why values have been eliminated from their variable's domain. When a backtrack step is required, the agent is able to determine the variable responsible for the conflict and backjumps to the agent holding that variable. This increases the performance of the algorithm very substantially when compared to SBT [102] whilst it has also been shown to outperform asynchronous algorithms on some problems [12]. We have implemented a distributed version of backjumping (see Appendix A.4) and found that SynCBJ significantly outperforms this backjumping algorithm. A revised version of SynCBJ entitled **Multi-CBJ** [81] speeds up the search by running multiple SynCBJ search processes with different orderings on different processors.

Yokoo et al. developed **Asynchronous Backtracking** (ABT) [94, 97] to allow agents to search in parallel for a solution. Each agent has a priority (or place) in the ordering. This is often lexicographic by agent id, but other heuristics such as maximum degree or minimum domain can be used. Statically ordered agents (later approaches use dynamic ordering) send values to lower priority agents which evaluate shared constraints. Asynchronously, each agent assigns a consistent value to its variable, sending OK messages to agents sharing constraints (connected agents) to determine if violated constraints exist. Each agent's view of the problem is updated with the new value of that agent and constraints are checked. If constraints are violated, and the agent cannot change to a consistent value, the agent sends a nogood message to the lowest priority agent with a higher priority than itself. This agent checks the nogood for validity (message delay and parallel execution may affect this) prior to changing its value. If an agent receives a nogood with an unconnected agent, a link is added between the two agents. This process is repeated until all agents have consistent assignments or all values of the highest priority agent lead to failure.

A simple Distributed Constraint Satisfaction problem is shown in figure 4.1 with three agents WEB, OOP and $DATABASES$ each representing a single variable with the same name as its agent. We assume for the sample execution of ABT that agent WEB has the highest priority, followed by OOP with $DATABASES$ having the lowest priority. WEB and OOP send OK messages to $DATABASES$ with their chosen values (e.g. 11am and 10am respectively). At this point, $DATABASES$ knows that the value of WEB is 11am and OOP is 10am. These values are stored in the agent view of $DATABASES$. $DATABASES$ cannot assign a value consistent with this agent view and so sends a nogood {(WEB,11am)(OOP,10am)} to the lowest priority agent higher than itself (i.e. OOP). Since this nogood, contains a variable (WEB) which OOP has no constraints with, a link is added between WEB and OOP. The value of WEB as

11am is now stored in the OOP agent's view. OOP would now send an OK message to $DATABASES$ with its other value of 12noon but $DATABASES$ generates an additional nogood {(WEB,11am)(OOP,12noon)} to OOP. OOP now has no consistent values and so sends a nogood to WEB {(WEB,11am)}. WEB now changes its value to 1pm and sends an OK message to $DATABASES$ and OOP. $DATABASES$ can now assign a consistent value to its variable (10am) and a solution is found to the problem {(WEB,1pm)(OOP,12noon)($DATABASES$,10am)}.

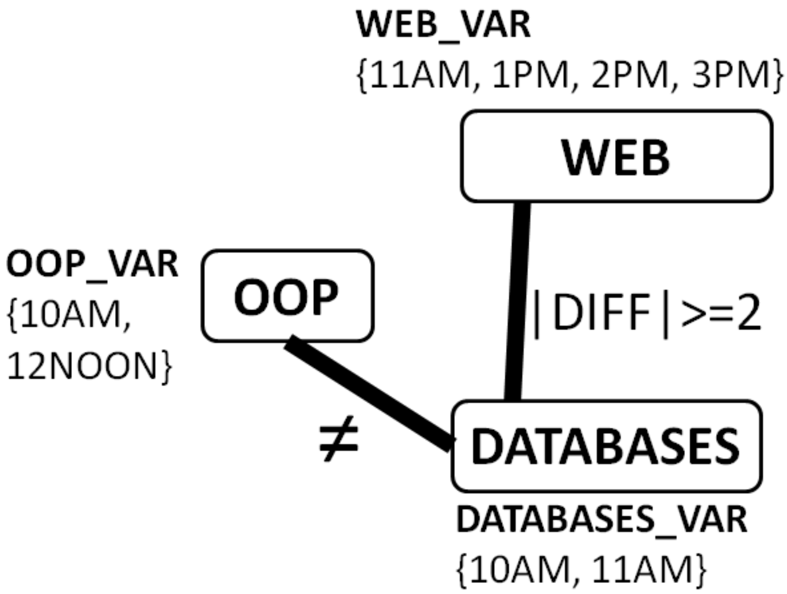

Figure 4.1: A simple Distributed Constraint Satisfaction Problem

ABT has been extended by various authors. Yokoo proposed **Asynchronous Weak-Commitment Search** (AWCS) [93, 97], a new algorithm with dynamic ordering where backtracking agents become the highest priority agent within their neighbourhood. For example in the simple problem, $DATABASES$ would be promoted to the highest priority agent when it composes a nogood to backtrack to OOP. This algorithm requires exponential space complexity to store all nogoods generated for completeness [2]. Some authors classify AWCS as a local search algorithm.

Bessière et al. [10] removed the need for new links between unconnected agents in

[2]This algorithm is a descendant of Weak-Commitment Search (see section 3.8), but is no longer a hybrid approach in this distributed version.

ABT. Fernandez et al. [28] proposed to negate the message delay effect through a random restart procedure.

Meisels and Zivan explored integrating forward-checking constraint propagation with sequential variable assignment [57], parallel exploration through the **ConcDB** algorithm [103] and message delay effects on ABT and AWCS [101].

Silaghi's extensions include aggregations where constraints are sent between agents rather than variable values [83], combining ABT and AWCS to reduce AWCS's space complexity [85] and asynchronous consistency [84].

Brito and Meseguer [12] create **ABT-Hyb**, introducing synchronised steps for part of ABT to reduce the number of messages between agents.

Nguyen et al. [64] developed **Dynamic Distributed Backjumping** with synchronous forward assignment of variables phase and asynchronous backjumping phase when assignments fail.

Sycara et al. developed a heuristic-based search for scheduling problems which is also called **Asynchronous Backtracking** search [86].

Concurrently with Yokoo's ABT and AWCS, Hamadi has developed **IDIBT/CBJ-DkC** [40] using parallel exploration of search trees, constraint propagation and conflict-directed backjumping.

Harvey et al. [44] developed **Support-Based Distributed Search** using argumentation techniques and message ordering rather than variable ordering. The approach is complete through nogood construction.

Distributed Backtracking with Sessions [61] is a recent approach which aims to minimise message processing time as opposed to the number of messages actually sent. The algorithm uses the notion of sessions to only process those messages which contain the same session number. Sessions are closed whenever an agent is able to assign a value to its variable. Backtracks are only processed if the session number between the agents is identical. This reduces processing effort of obsolete messages.

The distributed backtracking approaches have identical disadvantages as centralised backtracking techniques (see section 3.4). Specifically, they incur a high number of non-concurrent constraint checks. In addition, for distributed problems, they incur a high cost of sending messages.

4.2.3 Distributed Local Search

A number of local search approaches exist for distributed problems. In local search, an ordering is defined for all agents. The first agent processes all relevant messages for this agent and sends appropriate messages to neighbours before passing processing to the next agent in the ordering. The next agent then processes all relevant messages for this agent and sends appropriate messages to neighbours and passes on processing to the next agent in the ordering. This continues until all agents have had the opportunity to process. Once all agents have had the opportunity to process, processing returns to the first agent. All agents having the opportunity to process constitutes a **cycle**. Since local search algorithms are incomplete, they may execute indefinitely. Consequently, if an algorithm has not found a solution in a bounded number of cycles, the algorithm is terminated with no solution found (although a solution may exist).

Hirayama and Yokoo [45] developed the **Distributed Breakout Algorithm** (DBA) which is based on the breakout algorithm [62]. This algorithm sets an initial random assignment for variables with a weight of one for each constraint. In each cycle, agents then calculate value proposals to lower the number of constraint violations for their own variables. Where conflicts (i.e. neighbouring agents) occur, the agent with the maximum improvement value gets to change its value. If there is no conflict, then the agents may change their value to obtain the calculated improvement. If local optima occurs i.e. there are no improvements in one cycle, the weight of the constraints which remain violated is increased - a 'breakout' is performed. These weights are added to the number of constraint violations when calculating improvements. Consequently, after a 'breakout' values which do not violate heavily weighted constraints will appear more promising and the algorithm may be able to choose these as improvements and escape the local optimum. The problem is solved when all constraints are satisfied.

The **Distributed Stochastic Algorithm** (DSA) originates from Fabiunke's work [24] suggesting a probability of a variable keeping its existing value or taking a new value which minimises constraint violations. DSA [100] is probability based where initial values are improved to reduce constraint violations according to probabilities in each cycle. Several versions exist, but the most common, DSA-B, implements all improvements. In addition, DSA-B implements those changes which do not increase the number of violations according to a specified probability. These changes that do not increase the violations, but also do not decrease them, allow variables to move to other values that may in combination with other variables' values, reduce the number of constraint violations and escape local optima. Arshad and Silaghi [2] note that DSA has no method of escaping from local minima, but outperforms DBA on scan scheduling problems [100]. Arshad and Silaghi have extended DSA with a controlled descent approach to the probability of choosing changes which do not increase violations entitled **Distributed Simulated Annealing**.

The **Distributed Penalty Driven Search Algorithm** (DisPeL) [8] is a deterministic penalty-driven local search algorithm. DisPeL is an iterative improvement algorithm where agents take turns to improve a random initialisation in a fixed order. DisPeL uses two penalty mechanisms to force the search away from local optima, proving more effective and more informed than DSA and DBA. During each cycle, agents choose a value according to penalties imposed on variable values and the number of constraint violations, with values having a small penalty and a small number of constraint violations preferred. In order to resolve deadlocks (quasi-local-optima where an agent's view remains unchanged for 2 iterations), DisPeL applies penalties to variable values which are used in a 2-phased strategy as follows: (i) First the current values at quasi-local-optima are penalised with a **temporary penalty** of 3 in order to encourage agents to assign other values with the temporary penalty discarded in the next cycle and; (ii) If the temporary penalties fail to resolve a deadlock **incremental penalties** of 1 per penalty are imposed on the culprit values. Penalties therefore indicate values that, though looking promising, fail to lead to a solution. The higher the penalties accumulated by a value, the less desirable it becomes. The incremental penalties are discarded when the penalised agents become consistent or the search space becomes distorted (penalties have too big effect on problem). DisPeL is a deterministic algorithm relying on a good random initialisation to find a solution quickly.

Stoch-DisPeL [8], is a stochastic variation of DisPeL where agents decide randomly to either impose a temporary penalty (with probability p) or to increase the incremental penalty (with probability 1-p). Larger penalties may now be built-up where a repeated incremental penalty is topped up with a temporary penalty. Stoch-DisPeL is non-deterministic and outperforms DisPeL on bad initialisations and performs comparatively on good initialisations.

All of these local search algorithms are generally incomplete, although DBA is complete for cyclic graph problems [100].

4.2.4 Distributed Variable and Value Ordering

Hamadi et al. [41] proposes the **Distributed Agent Ordering** framework for static variable ordering with the max degree heuristic. Brito and Meseguer [12] have investigated ordering heuristics for synchronous and asynchronous search whilst Zivan and Meisels [104] have devised the **ABT_DO** reordering algorithm for ABT, recently updated to support nogood and minimum domain ordering [105]. Petcu and Faltings [68] describe a value ordering heuristic version for DBA.

4.2.5 Distributed Hybrid Algorithms

While there are many hybrid approaches for centralised CSPs, there are very few for distributed CSPs.

DisBOBT [22] runs DBA [95] as its main problem-solver and, if within a number of breakout steps it fails to solve the problem, DBA's weight information is used to order the agents for Yokoo's Synchronous Backtracking search [97]. The algorithm's purpose was to identify unsolvable problems and is not a fully optimised hybrid algorithm.

LSDPOP [70] is inspired by the centralised hybrid approach in [52] and runs the systematic algorithm DPOP [69], until the maximum inference limit is exceeded when local search guided by DPOP is run. LSDPOP is an optimisation algorithm and, therefore, focuses on finding the best solution to a problem with many solutions as opposed to finding any solution to the problem.

Ringwelski and Hamadi [77] provide a framework for combining multiple distributed algorithms based on Gomes et al's framework for centralised algorithms [37], although no applicability for combining backtracking and local search algorithms is given.

4.3 Distributed Constraint Satisfaction with Complex Local Problems

In section 4.2, algorithms are presented for the resolution of **fine-grained DisCSPs** i.e. DisCSPs where each agent is responsible for only one variable. In this section, we consider algorithms for the resolution of DisCSPs with complex local problems i.e. DisCSPs where each agent is responsible for more than one variable. Algorithms for the resolution of fine-grained DisCSPs can be used to solve DisCSPs with complex local problems through creating a virtual agent for each variable in a complex local problem. Thus, an agent is only responsible for one variable, instead of for a set of variables and, therefore, cannot make full use of all the knowledge contained in the complex local problem. This can result in both additional constraint checks and increased message costs. Alternatively, these approaches could have, for each complex local problem, a **complex variable** whose variable is the aggregation of all solutions to the local problems. Recently, there has been research into a compilation formulation for existing distributed algorithms primarily for distributed optimization [13]. This compilation formulation focuses on generating the set of intra-agent solutions first and then solving the global inter-agent problem for DisCSPs with complex local problems. **ABT-cf** [23] is a revised approach based on ABT which generates solutions to the local problem (intra-agent constraints) and then attempts to find solutions to the global problem (inter-agent constraints).

In this section, algorithms particularly designed for DisCSPs with complex local problems are considered.

4.3.1 Distributed Backtracking for Complex Local Problems

The **Asynchronous Weak-Commitment Search Algorithm for Complex Local Problems** (Multi-AWCS) [96] uses a local AWCS solver to ensure the satisfaction of intra-agent constraints, whilst a global AWCS solver ensures the satisfaction of inter-agent constraints. The global solver checks whether an assignment that the agent found for its own local variables is compatible with other agent's assignments.

The **Asynchronous Backtracking for Complex Local Problems** (Multi-ABT) [46] is a comparable extension for ABT. This also runs a local ABT solver and a separate global ABT solver. Maestre and Bessiere [53] produced an alternative version of this algorithm with nogood learning and value selection techniques.

Whilst all distributed backtracking algorithms for DisCSPs with Complex Local Problems are complete, they may take exponential time and, for Multi-AWCS, may require an exponential number of nogoods to be stored.

4.3.2 Distributed Local Search for Complex Local Problems

The Distributed Breakout Algorithm has been extended for Complex Local Problems through the **Multi-DB** algorithm [45]. This algorithm was initially extended by Eisenberg [22] (**DisBO**) whilst Basharu added a weight decay mechanism to the weights attached to constraints in **DisBO-wd** [8]. In DisBO-wd, each constraint is assigned an initial weight of 1. At the end of each iteration, the weight is updated so that it is increased if the constraint is violated and it is decayed if the constraint is satisfied. DisBO-wd has been shown to improve DisBO [22] and our experiments have confirmed this result.

Multi-DisPeL [8] extends the DisPeL framework for DisCSPs with Complex Local Problems through a modified steepest descent local search within the agent's local problem and the Stoch-DisPeL framework on the inter-agent constraints. Multi-DisPeL was found to often outperform Multi-AWCS [96] and DisBO-wd [8].

4.3.3 Distributed Hybrid Algorithms for Complex Local Problems

Whilst DisBOBT [22] is described with agents having more than one variable, only one variable per agent is processed at a time and consequently the algorithm is not specifically designed to handle the extra information available within complex local problems. **DCDCOP** [50] is a very recent approach to solving distributed constraint optimization problems. This algorithm computes local optimal solutions through branch and bound whilst determining the global optimal solution through the combined optimality of the agents including their inter-agent constraint. It is shown to outperform ADOPT, a leading distributed constraint optimisation algorithm [3].

4.4 Comparing Distributed Backtracking and Distributed Local Search

We evaluated one of the best synchronous distributed systematic search algorithms (SynCBJ) against one of the best synchronous distributed local search algorithms (Stoch-DisPeL). We were interested to see where their particular strengths and weaknesses lie within three problem classes: randomly generated problems, graph colouring problems and meeting scheduling problems. We also wanted to consider whether a combination of systematic or local search may be more beneficial when both algorithms do not perform well. In [102] SynCBJ was shown to outperform synchronous backtracking and asynchronous search on particular types of problems [12]. Stoch-DisPeL has been shown to outperform other distributed local search algorithms in [8]. Our implementations were verified for SynCBJ with the distributed randomly generated problems described in [102] ($n = 10$, $d = 10$, $p1 = 0.7$ and $p2 \in \{0.1, ..., 0.9\}$) and for Stoch-DisPeL with the distributed randomly generated problems in [8]. The results were at least as good as those reported by the authors.

[3]A personal communication with this author has shown this version of ADOPT to be the original version of ADOPT and not the current leading variant of ADOPT. As a result, some of the conclusions of the paper may differ on the current leading variant of ADOPT.

Randomly Generated Problems

We compared the performance of systematic search (SynCBJ [102]) against local search (Stoch-DisPeL [8]) on **solvable random DisCSPs** with 30 to 60 variables (n), 30 to 60 agents, 10 variable values (d), 0.15 constraint density ($p1$) and constraint tightness ($p2$) between 0.1 and 0.9 in steps of 0.1. The median number of messages over 100 runs for 50 variables are shown in figure 4.2 and for 60 variables in figure 4.3. For constraint checks, results for 50 variables are shown in figure 4.4 and for 60 variables in figure 4.5. Note that at a constraint tightness of 0.4, Stoch-DisPeL was only able to solve 97% of problems and the effort wasted for the 3% of problems that were not solved is not included in the results.

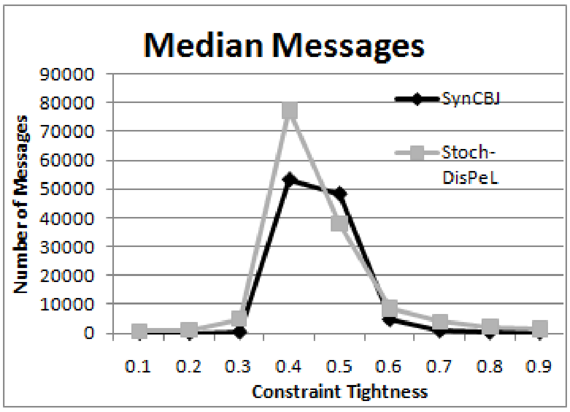

Figure 4.2: Messages for $< n = 50, d = 10, p1 = 0.15, p2 \in 0.1, 0.2, ..., 0.9 >$

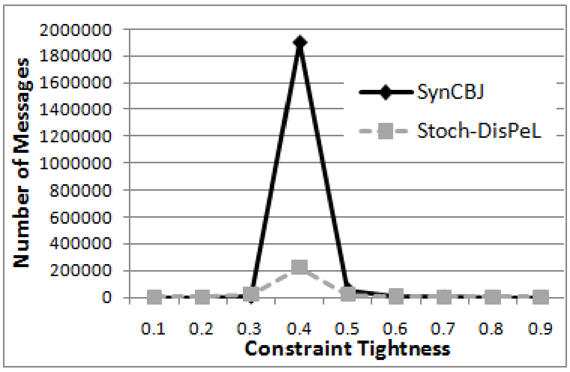

Figure 4.3: Messages for $< n = 60, d = 10, p1 = 0.15, p2 \in 0.1, 0.2, ..., 0.9 >$

The **phase transition point** (marking the boundary where very difficult problems exist) occurs close to a constraint tightness of 0.4. For 30 and 40 variables (not shown

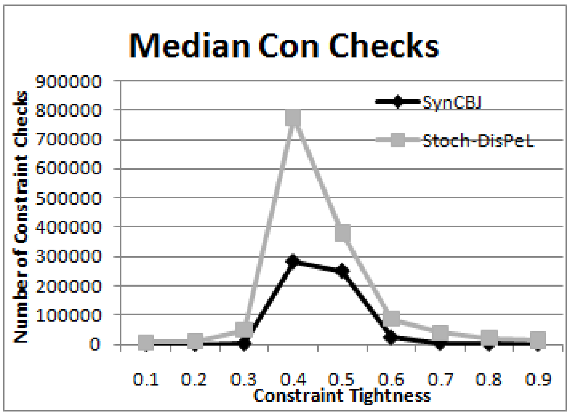

Figure 4.4: Constraint checks for $< n = 50, d = 10, p1 = 0.15, p2 \in 0.1, 0.2, ..., 0.9 >$

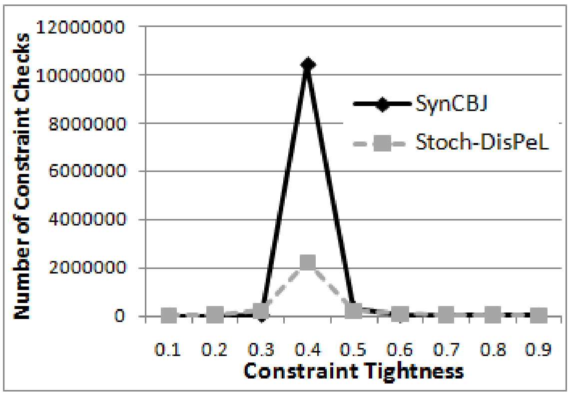

Figure 4.5: Constraint checks for $< n = 60, d = 10, p1 = 0.15, p2 \in 0.1, 0.2, ..., 0.9 >$

here), SynCBJ was substantially better. For 50 variables, SynCBJ is almost always better for both messages and constraint checks (with the exception of messages at the phase transition point) than Stoch-DisPeL because these are easier problems. The difference is a minimum of 650 messages and 5000 constraint checks but often higher. For larger problems with 60 variables, at the phase transition point, SynCBJ uses a very large number of messages and constraint checks. Whilst Stoch-DisPeL uses fewer messages and constraint checks, it is not able to solve all problems since it is an incomplete algorithm. For all other constraint tightness values, both algorithms perform similarly although SynCBJ is always able to solve problems with fewer messages and constraint checks than Stoch-DisPeL.

We also conducted experiments with SynCBJ and Stoch-DisPeL for 70 or more variables. For these problems, SynCBJ was unable to solve all problems within a practical time limit of 24 hours.

Consequently, we conjecture that for random DisCSPs at the phase transition point and particularly for problems with larger number of variables, hybrid algorithms combining local search and backtracking could be beneficial in terms of obtaining completeness with fewer messages and constraint checks than systematic search. Outside of the phase transition point, SynCBJ would appear to perform well.

Graph Colouring Problems

We conducted an experiment to compare the performance of systematic search (SynCBJ) and local search (Stoch-DisPeL) on **solvable graph colouring DisCSPs** with 125 to 200 nodes (n), 125 to 200 agents, 3 colours (c) and degree (d) between 4.3 and 5.6. For 125 and 150 nodes, SynCBJ was the better performing algorithm. The median results over 100 runs are shown for messages with 175 variables in figure 4.6, for messages with 200 variables in figure 4.7, for constraint checks with 175 variables in figure 4.8 and for constraint checks with 200 variables in figure 4.9.

With graph colouring DisCSPs, the phase transition point is less defined than for randomly generated DisCSPs. The peak point occurs at a degree of 4.9 but it is only slightly higher than degrees of 4.8 and 5.0. For graph colouring DisCSPs, Stoch-DisPeL is

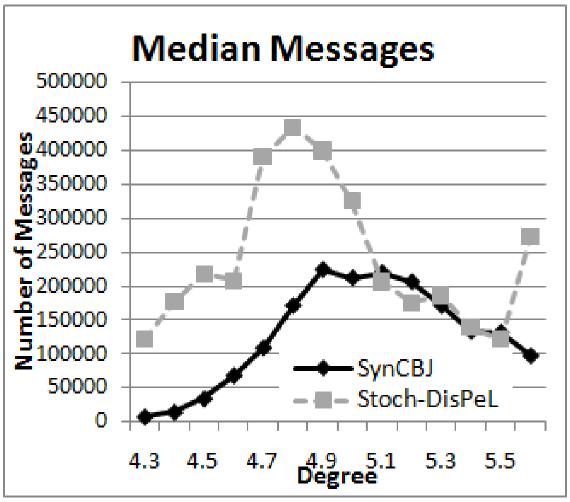

Figure 4.6: Messages for $< n = 175, c = 3, d \in 4.3, 4.4, ..., 5.6 >$

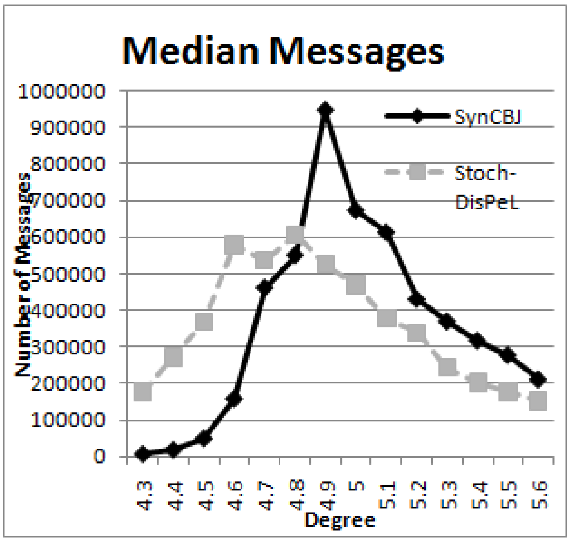

Figure 4.7: Messages for $< n = 200, c = 3, d \in 4.3, 4.4, ..., 5.6 >$

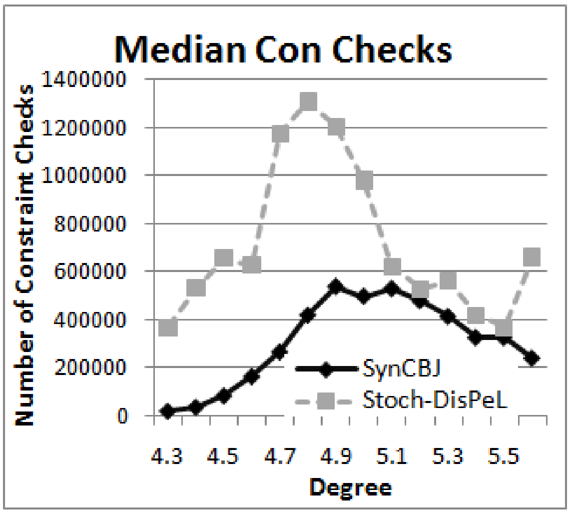

Figure 4.8: Constraint Checks for $< n = 175, c = 3, d \in 4.3, 4.4, ..., 5.6 >$

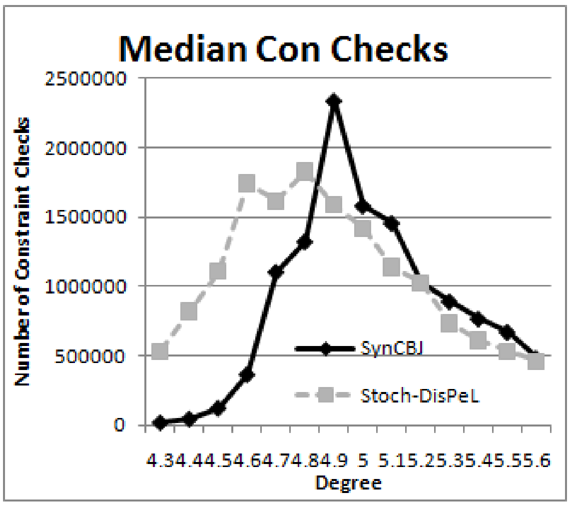

Figure 4.9: Constraint Checks for $< n = 200, c = 3, d \in 4.3, 4.4, ..., 5.6 >$

competitive for degrees between 5.1 and 5.5. However, the cost of solving the problem is high for both algorithms particularly from a degree of 4.7 onwards. It would appear that for graph colouring problems, local search cannot easily offset systematic search when the later performs badly. Consequently, we included graph colouring problems in our evaluation to determine if hybrid algorithms combining systematic search and local search can improve performance.

Meeting Scheduling Problems

We conducted an experiment comparing systematic search and local search on **solvable meeting scheduling problems** with 30 to 60 meetings (m), 30 to 60 agents, maximum possible distance (md) of 3 and constraint density (d) between 0.1 and 0.25. For 30 and 40 variables, SynCBJ was the better performing algorithm. The median results over 100 runs for messages are shown in figure 4.10 for 50 variables and figure 4.11 for 60 variables and for constraint checks for 50 variables in figure 4.12 and for 60 variables in figure 4.13.

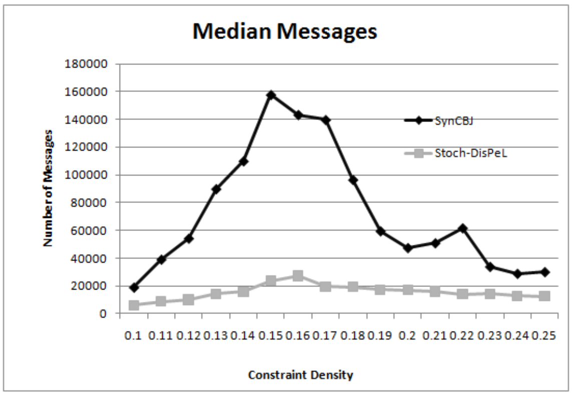

Figure 4.10: Messages for $< m = 50, md = 3, d \in 0.1, 0.11, ..., 0.25 >$

Stoch-DisPeL outperforms SynCBJ for meeting scheduling DisCSPs for both messages and constraint checks. However, with the exception of a density of 0.18, Stoch-DisPeL did not solve all problems with a density between 0.1 and 0.19. Therefore, whilst Stoch-DisPeL can improve performance, the algorithm is not suitable when a solution is required and not just an approximation. Consequently, there is a need for hybrid algorithms to guarantee completeness for meeting scheduling DisCSPs whilst also improving performance over systematic search.

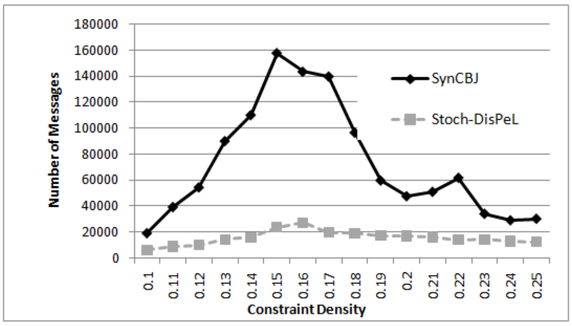

Figure 4.11: Messages for $< m = 60, md = 3, d \in 0.1, 0.11, ..., 0.25 >$

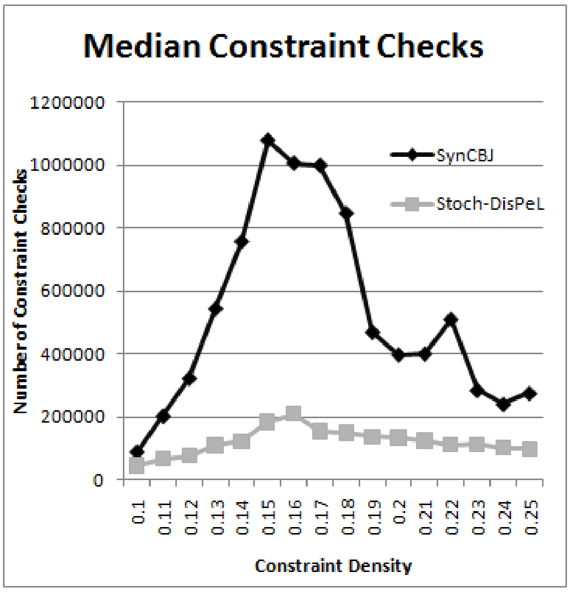

Figure 4.12: Constraint Checks for $< m = 50, md = 3, d \in 0.1, 0.11, ..., 0.25 >$

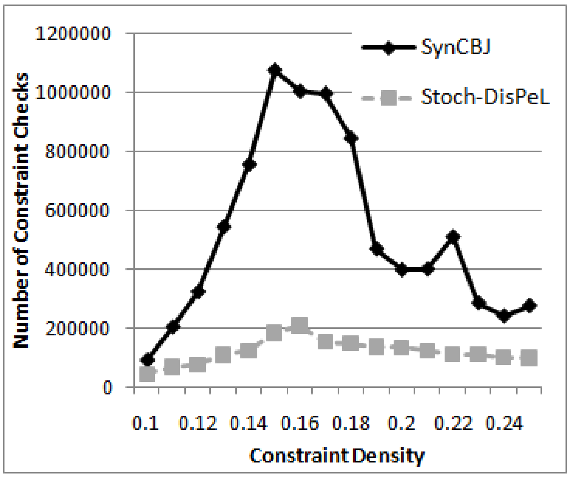

Figure 4.13: Constraint Checks for $< m = 60, md = 3, d \in 0.1, 0.11, ..., 0.25 >$

This short experimental study did not consider unsolvable problems. For these problems, hybrid algorithms would be an important contribution since systematic search algorithms still offer poor performance at the phase transition point and local search algorithms cannot detect unsolvability.

4.5 Summary

In this chapter, we have introduced Distributed Constraint Satisfaction problems (DisCSPs). We have shown two formalisations of Distributed Constraint Satisfaction: (i) DisCSPs with only one variable per agent; (ii) DisCSPs with complex local problems (multiple variables per agent). We have described backtracking and local search algorithms for solving both formalisations and discussed the lack of hybrid approaches particularly for DisCSPs with complex local problems. An experimental study has shown that systematic search has weaknesses around the phase transition points for these problems and particularly for larger numbers of variables and that local search does not offer completeness for all problems in these cases. Therefore, in the following chapters, we present hybrid algorithms combining systematic and local search to guarantee completeness in a practical timeframe for these problems.

Chapter 5

Using Knowledge from Local Search to guide Systematic Search

5.1 Introduction

In the previous chapter, we saw that there are problems associated with backtracking and local search algorithms. In this chapter, a new approach to Distributed Constraint Satisfaction entitled *DisHyb* is proposed which combines local search with systematic search in order to speed-up the latter. In DisBOBT (see section 4.2.5), it was shown that local search could improve the ordering of backtracking search for those problems which local search could not solve itself. In the *DisHyb* approach, we seek to collect information during a frequently short execution of local search. This information is used as a heuristic to guide systematic search.

A diagram of the approach is shown in figure 5.1. The approach has two phases. In the first phase, a distributed local search algorithm is run. This algorithm gathers knowledge about difficult variables and values. In the second phase of the algorithm, a distributed systematic search algorithm is run with the agents reordered. The agents are reordered according to the knowledge about difficult variables and values which distributed local search gathered. The distributed local search in phase 1 can be seen as a pre-processing ordering heuristic for the distributed systematic search. However, distributed local search remains potentially capable of solving the problem in the phase 1 and so phase 2 may not be required.

The novel aspect of this approach is the use of information from distributed local search to distributed systematic search. Consequently, whilst existing distributed local and systematic search algorithms are reused, a novel interface is used to connect them.

In this approach, a trade-off exists between the cost of local search and the benefit of the knowledge gained. In order to gain optimal performance, distributed local search must be run for a long enough period to learn very good knowledge, but not too long in order to incur too high costs. An analysis of this trade-off is presented later in this chapter.

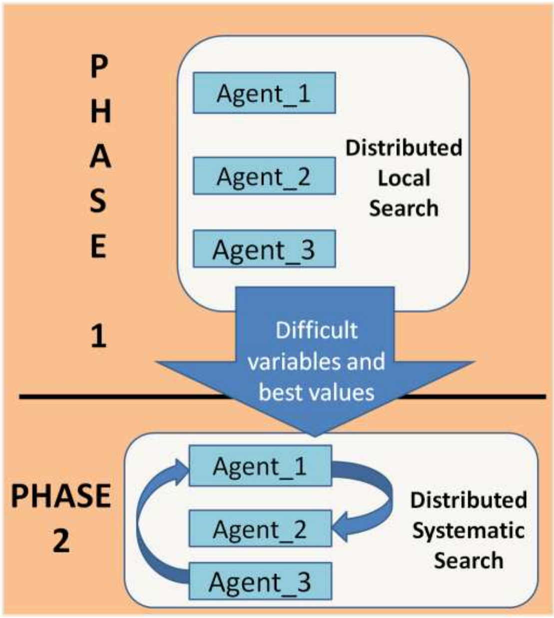

Figure 5.1: The DisHyb approach.

We present two implementations of our approach, *PenDHyb* and *DBHyb* which differ in the local search strategy used. An overview of the approach and algorithms presented in this chapter is shown in table 5.1.

Approach	Algorithm	Local Search Strategy
DisHyb	PenDHyb	Penalties on values
DisHyb	DBHyb	Constraint weights ('Breakout')

Table 5.1: Chapter Overview.

We present empirical evaluation of our algorithms on randomly generated, graph colouring and meeting scheduling DisCSPs.

5.2 DisHyb: Distributed Knowledge-Based Hybrid Approach

Local search approaches rely heavily on strategies for escaping local optima, e.g. **weights on constraints** [45] or **penalties on values** [8]. Despite the effectiveness and efficiency of some of these approaches, local search is generally incomplete. However, local search approaches are used because, for large problems, they can be faster than the systematic approaches [30].

In local search, when a local optimum is reached, algorithms tend to modify the search landscape to encourage exploration of other areas of the search space. This usually is designed to affect the values taken by variables that participate in constraints which are currently violated. Rather than just using this information to escape the local optima, it is also possible to **discover knowledge** from this local search operation. It is possible to discover the following knowledge from local search:

- **Difficult variables:** Those variables which are frequently found to be involved in local optima can be seen as more difficult to assign than variables which are rarely involved in local optima. Consequently, variables could be ordered according to their difficulty for the systematic search algorithm.

- **Best variable values:** the best solution found (i.e. the value for each variable that contributed to the lowest total constraint violations for all agents) in the local search algorithm, can be the first value considered for variables in a systematic search.

We present the synchronous *DisHyb* framework in Algorithm 1 using the knowledge described above from local search to drive systematic search. The flow of execution in *DisHyb* is shown in figure 5.2.

Firstly, local search is executed. The local search runs as normal but *DisHyb* notes each time that a variable is involved in local optima (as detected by the local search algorithm). In addition, the final agent in the search checks if a new best solution (i.e. minimising total constraint violations for all agents) has been found. If a new best solution exists, a

Algorithm 1 DisHyb

```
initialise agents with variables
for each variable v_i do
   set difficulty d_i to 0
   set its best value bv_i to its initial instantiation
end for
repeat
   for each agent a_i responsible for variable v_i do
      if message received stating current value participates in best solution found then
         set bv_i to current value
      end if
      local_search_agent_main_loop(termination_cond)
      if v_i_in_local_optima then
         increase d_i.
      end if
   end for
until termination_cond
if solution found by local search then
   return solution
else
   ao ← list of agents sorted by max-degree and their variables' difficulty (d_i).
   for each variable v_i do
      Prioritise best values (bv_i)
   end for
   systematic_search(ao)
   if solution found by systematic search then
      return solution
   else
      return "unsolvable problem"
   end if
end if
```

message is sent to the first agent to save its value as the best value so far. This message is cascaded down to all agents during the next processing cycle. This minimises the addition of new messages. If the local search does not find a solution within a bounded number of cycles (the termination condition), then a systematic search is run with the agents ordered according to the difficulty of the variables represented by that agent. If this systematic search is complete, then the *DisHyb* approach is complete.

In Jussien's classification of hybrid algorithms [48] (see chapter 3), our knowledge-based hybrid approach would be classified in the performing local search before/after systematic search category.

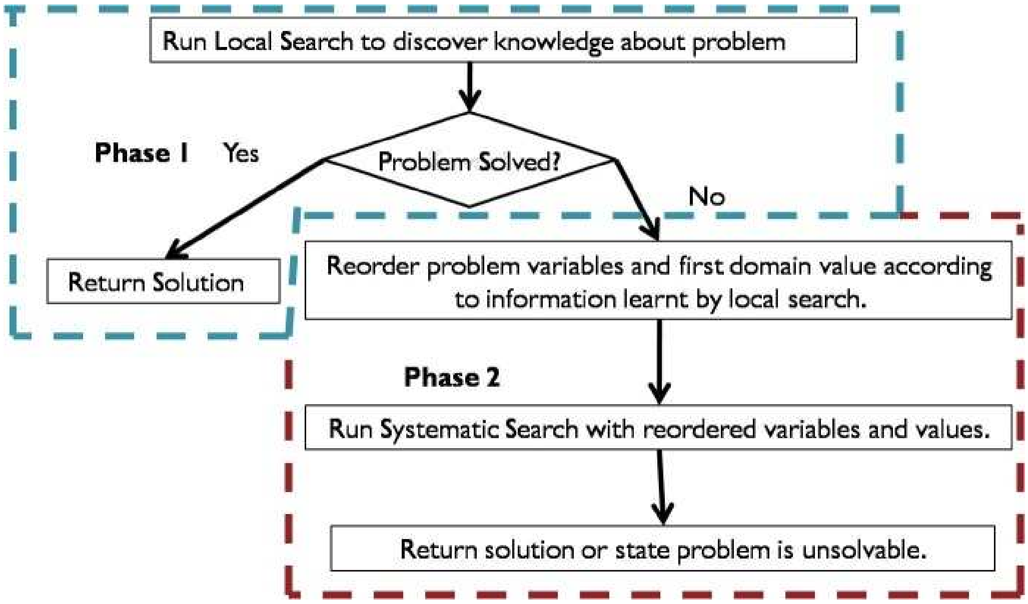

Figure 5.2: The flow of execution in the DisHyb approach.

5.3 DisHyb Implementations

We present two instances based on our approach which differ in the strategy used by the local search algorithm. The Penalty-Based Distributed Hybrid Algorithm (*PenDHyb*) uses local search with penalties on values [8] (a short execution of Stoch-DisPeL [8]) with a systematic search algorithm (SynCBJ [102]). The Weight-Based Distributed Hybrid Algorithm (*DBHyb*) uses local search with breakout [45] (a short execution of SingleDB-wd) with a systematic search algorithm (SynCBJ [102]). An alternative implementation, DisPBJ, combining local search with penalties on values [8] (a short execution of Stoch-DisPeL [8]) with distributed backjumping is discussed in Appendix A.

5.3.1 Penalty-based Distributed Hybrid algorithm (PenDHyb)

In this section, we present the Penalty-based Distributed Hybrid algorithm (*PenDHyb*), an instance of *DisHyb* which uses **penalties on values** as the local search strategy. *PenDHyb* combines Stoch-DisPeL as the local search algorithm and SynCBJ as the systematic search algorithm.

The information we learn from Stoch-DisPeL can be summarised as follows:

- **Difficult variables:** Penalties on values are used to learn which variables are difficult to assign during problem solving. A variable which has many heavily penalised values is seen as more troublesome than a variable whose values have few or no penalties. Variables are ordered in decreasing number of penalties and this order is used to drive SynCBJ.

- **Best variable values:** the best solution found (the one with the least constraint violations) in DisPeL, is used for selecting the first value for each variable in SynCBJ.

Algorithm Description

The reader is referred to section 4.2.3 for a description of the Stoch-DisPeL algorithm. In the remainder of this chapter, Stoch-DisPeL is referred to as DisPeL. In order to learn penalty information for use in SynCBJ, DisPeL was modified by adding:

- A *penalty counter* pc_i for each variable v_i. pc_i is incremented by the value of the penalty whenever a penalty is imposed on any of v_i's values. Unlike penalties on values, penalty counters are never reset and, therefore, highlight repeated penalisation of variables, i.e. *troublesome* variables.

- A *best value* store bv_i for each variable v_i which keeps the value participating in the best solution found by DisPeL so far. In order to determine whether the current value is the best so far, each agent adds its current score (*constraint violations* + *penalties*) to the score passed to it by its predecessor and sends this score to the next agent. The last agent determines if the current solution is the best so far and if so it informs the first agent who will inform others in the next cycle. During this cycle, the message that gives control to the next agent of processing also includes a parameter to save current value as best value so far before choosing a new value. Therefore, the best solution found can be determined without incurring any additional messages.

The *PenDHyb* algorithm is shown in Algorithm 2. After agent initialisation, DisPeL runs (as described for Stoch-DisPeL in [8]) but only for a very small number of cycles (see section 5.3.1). If the problem is solved, the solution is returned. Otherwise, variables are

Algorithm 2 PenDHyb

```
initialise
for each variable v_i do
    set its penalty count pc_i to 0
    set its best value bv_i to its initial instantiation
end for
repeat
    for each agent a_i responsible for variable v_i do
        if message received stating current value participates in best solution found then
            set bv_i to current value
        end if
        DisPeL_agent_main_loop(termination_cond)
        if penalty_imposed then
            increment pc_i.
        end if
    end for
until termination_cond
if solution found by DisPeL then
    return solution
else
    ao ← list of agents sorted by max-degree and their variables' penalty count (pc_i).
    for variable v_i do
        Prioritise best values (bv_i)
    end for
    SynCBJ(ao)
    if solution found by SynCBJ then
        return solution
    else
        return "unsolvable problem"
    end if
end if
```

arranged, in descending order, according to their maximum degree (number of constraints) with ties broken by penalty count before SynCBJ is run. In addition to the variable ordering information, SynCBJ makes use of value ordering information as follows: for each variable v_i, the first value to be tried is the best value bv_i found by DisPeL, i.e. the one participating in the best instantiation found.

Properties

PenDHyb is complete since either DisPeL reports a solution within the small number of cycles (typically DisPeL solves 5% of problems) or SynCBJ runs. Since SynCBJ is complete, completeness of *PenDHyb* is guaranteed. *PenDHyb* is sound since both DisPeL and SynCBJ are sound [8, 102].

Variable and Value Ordering in PenDHyb

A number of methods for exploiting the knowledge gained from running DisPeL were evaluated in order to provide variable and value ordering for SynCBJ. Solvable random binary distributed constraint satisfaction problems were used in the experiments with $n = 50$, $d = 10$, $p1 = 0.15$ and $p2 = 0.4$. These problems correspond to the phase transition region where, the number of messages and non-concurrent constraint checks incurred by systematic search become much higher and consequently local search becomes more desirable (see section 4.4). Therefore, hybrid algorithms are likely to be important on these problems in guaranteeing completeness where systematic search is not as effective. Combinations of the following options were investigated:

- **Variable ordering:**
  - **Penalties:** this heuristic orders variables according to max-degree with ties broken by penalty counts. Seeks to measure whether penalties are a positive addition to the basic max-degree heuristic.
  - **Last penalties:** in the original DisPeL search, incremental penalties are discarded periodically (reset) when the search space becomes distorted (see section 4.2.3). This heuristic uses the penalty counts between the last reset and the end of DisPeL execution as opposed to the cumulative penalty counts proposed above. It seeks to measure whether it is beneficial to allow time for the penalties to stabilise before learning from them.
- **Value ordering:**
  - **Sticking values:** the first variable value to be tried by SynCBJ is DisPeL's best value for that variable, i.e. the one participating in the best solution found. All other values are considered in their original order. This is inspired by [32] who used the last value assigned to a variable. The idea is that DisPeL often gets close to a solution and therefore the work already done by DisPeL can be reused in SynCBJ. It must be noted that SynCBJ does ultimately consider all values and so use of this heuristic does not impact on completeness.

– **Selected sticking values:** uses the sticking value only if that value led to no constraint violations in DisPeL. Otherwise, it uses the first value in the domain. The idea is that this will maximise the benefit of sticking values by giving SynCBJ the correct values whilst not forcing it to use incorrect values (i.e. those with constraint violations).

Table 5.2 presents the median results for messages and constraint checks over 100 runs.

	Number of Messages	**Number of Constraint Checks**
SynCBJ	262,178	1,344,941
Penalties	111,638	559,004
Sticking values	75,694	345,654
Penalties + selected sticking	77,482	332,050
Last penalties	131,602	624,803
PenDHyb	**50,929**	**321,237**

Table 5.2: Comparison of variable and value ordering heuristics ($n = 50$, $d = 10$, $p1 = 0.15$, $p2 = 0.4$).

The best performing method, is the one used in *PenDHyb*, where variables are sorted using max-degree and penalties (penalties heuristic) and values are prioritised using sticking values. Ordering variables according to penalties which are periodically reset produces the worst results since the algorithm cannot effectively determine the variable difficulty. It is worth noting that sticking values (selectively or not) leads to increased performance. This may be because DisPeL provides values which are normally part of a solution.

Determining Optimal Number of Cycles for DisPeL

Randomly Generated Problems						*Graph Colouring Problems (degree = 5)*					
Problem spec.			**Optimal values**			**Problem spec.**			**Optimal values**		
num vars	**dom size**	**num con**	**msgs**	**ccs**	**both**	**num nodes**	**num cols**	**num con**	**msgs**	**ccs**	**both**
30	10	90	3	3	3	125	3	313	11	4	4
40	10	120	12	4	14	150	3	375	9	5	9
50	10	150	65	45	65	175	3	438	53	5	53
60	10	180	99	66	96	200	3	500	108	108	108

Table 5.3: Sample of data used to determine optimal cycle cutoffs.

The only parameter which needed to be determined in *PenDHyb* was the number of cycles DisPeL should run for optimal performance. Its value was obtained as follows: (i) Experiments were run on three problem classes - solvable random binary DisCSPs, solvable graph colouring problems and solvable meeting scheduling problems - with varying characteristics (see below for details) and cutoff points for randomly generated problems and meeting scheduling problems between $0.1n$ and $2n$ cycles in increments of $0.1n$ or between $0.03n$ and $0.60n$ in increments of $0.03n$ for graph colouring problems (where n is the number of variables in the problem [1]). The cutoff recommended was always substantially lower than $2n$ for randomly generated problems and meeting scheduling problems and $0.60n$ for graph colouring problems and so $2n$ and $0.60n$ were chosen as upper limits; (ii) For each type of problem, the three most optimal cycle cutoffs were selected, i.e. the ones where the number of messages, the number of constraint checks or a combination of both is minimal (see Table 5.3 for sample data). For the combination of messages and constraint checks, the data was normalised and the optimal normalised value was chosen to remove scale bias; (iii) Multiple linear regression was used for predicting the linear relationship between the problem features (e.g. number of variables, domain size, number of constraints) and the optimal cycle cutoff obtaining the following:

$$cutoff = \alpha + (\beta * domainSize) + (\gamma * constraints) \tag{5.1}$$

Values for α, β and γ are given in Table 5.4 where the column heading indicates which metric (messages, non-concurrent constraint checks or both) is to be minimised. Note that we did not consider unsolvable problems when determining the optimal number of cycles since DisPeL will never be able to detect unsolvability. Consequently, the optimal number of cycles for an unsolvable problem is very low since we need to move quickly to the backtracking phase in order to detect unsolvability.

	Randomly Generated Problems			*Graph Colouring Problems*		
optimise	**messages**	**constr. checks**	**both**	**messages**	**constr. checks**	**both**
α	-134.810	-77.948	-146.334	-27.684	-31.305	-22.880
β	5.580	0.000	6.560	0.000	0.000	0.000
γ	0.888	0.798	0.872	0.113	0.127	0.101
	Meeting Scheduling Problems					
optimise	**messages**	**constr. checks**	**both**			
α	-10.615	-10.229	-10.718			
β	2.391	2.297	2.406			
γ	0.026	0.025	0.026			

Table 5.4: Parameter values for α, β and γ in Equation (5.1).

[1] Since the number of variables affects the size of the problem, it is intuitive to increase the number of cycles permitted as the problem size increases. Consequently, the number of cycles was always proportional to the number of variables.

The experiments for tuning the number of cycles allocated to DisPeL, detailed above were run on the following problems: (i) Randomly generated problems with number of variables $n \in \{30, 40, 50, 60, 70\}$, domain size $d \in \{8, 9, 10, 11, 12\}$, constraint density $p1 \in \{0.1, 0.15, 0.2\}$ and constraint tightness $p2 \in \{0.6(30), 0.5(40), 0.45(50), 0.4(60)\}$; (ii) Graph colouring problems with number of nodes $n \in \{100, 125, 150, 175, 200\}$, $d = 3$ and *degree* $\in \{4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3\}$; (iii) Meeting scheduling problems with number of variables $n \in \{30, 40, 50, 60\}$, domain size $d \in \{5, 6\}$, constraint density $cd \in \{0.1, 0.12, 0.14, 0.16, 0.18, 0.2, 0.22, 0.24\}$ and maximum distance $d \in \{2, 3\}$. These are problems in the difficult regions where SynCBJ is particularly expensive to run.

5.3.2 Weight-Based Distributed Hybrid Algorithm (DBHyb)

In this section, we present an alternative instance of *DisHyb*, the Distributed Breakout Hybrid algorithm (*DBHyb*), which combines DisBO-wd as the local search algorithm with SynCBJ as the systematic search algorithm.

We refer to DisBO-wd [8] as SingleDB-wd in the remainder of this chapter since each DisBO-wd search will only have a single variable per agent.

In SingleDB-wd, when a **local optima** is reached, weights on violated constraints are increased. Variables involved in heavily weighted constraints can be seen as more difficult to assign. In summary, we can obtain the following information from SingleDB-wd:

- **Difficult variables:** the weights on constraints can be used to learn which variables are difficult to assign. A variable involved in constraints which are heavily weighted is seen as more difficult to assign than a variable involved in few or none heavily weighted constraints. Variables are ordered in descending order of difficulty and this order is used to drive SynCBJ.

- **Best variable values:** the best solution found (the one with the least constraint violations) in SingleDB-wd, is used for value ordering in SynCBJ.

Algorithm Description

In order to learn weight information SingleDB-wd was modified by adding the *best value* bv_i store to retain the value participating in the best overall solution found by SingleDB-wd so far. This is determined by each agent adding its current constraint violations to the number of violations passed to it by its predecessor and sends this number to the next agent. The last agent determines if the current solution is the best found so far and so informs agents in the next cycle to retain their current value as the best value before searching for a new value. Consequently, no additional messages are incurred.

DBHyb is shown in Algorithm 3. Specifically, it differs from *PenDHyb*, in that constraint weights are used instead of penalties. A standard SingleDB-wd search runs only for a very small number of cycles (see section 5.3.2). If a solution is found, this is returned. Otherwise, each variable determines its highest constraint weight out of the constraints involving that variable. Variables are then arranged, in descending order, according to their degree and constraint weight before SynCBJ is run. In addition to this variable ordering, value ordering through the best value is performed in the same way as *PenDHyb*.

Properties

DBHyb is complete since either SingleDB-wd reports a solution within the small number of cycles (typically SingleDB-wd solves 3% of problems) or SynCBJ runs. Since SynCBJ is complete, completeness of *DBHyb* is guaranteed. *DBHyb* is sound since both SingleDB-wd and SynCBJ are sound [8, 102].

Variable and Value Ordering in DBHyb

A number of methods for exploiting the knowledge gained from running SingleDB-wd were evaluated in order to provide variable and value ordering for SynCBJ. Solvable random binary distributed constraint satisfaction problems were used in the experiments with $n = 50$, $d = 10$, $p1 = 0.15$ and $p2 = 0.4$. These problems are at the boundary where local search performs better than systematic search. Combinations of the following options were used:

Algorithm 3 DBHyb

```
initialise
for each constraint $c_i$ do
    set its constraint weight $cw_i$ to 1
end for
for each variable $v_i$ do
    set its best value $bv_i$ to its initial instantiation
end for
repeat
    for each agent $a_i$ responsible for variable $v_i$ do
        if message received stating current value participates in best solution so far then
            set $bv_i$ to current value
        end if
        SingleDBwd_agent_main_loop(termination_cond)
        for each constraint $c_i$ do
            if constraint $c_i$ is violated then
                increase $cw_i$
            else
                decay $cw_i$
            end if
        end for
    end for
until termination_cond
if solution found by SingleDBwd then
    return solution
else
    for each agent $a_i$ responsible for variable $v_i$ do
        $cwv_i \leftarrow$ highest weight of a constraint belonging to $v_i$.
    end for
    $ao \leftarrow$ list of agents sorted by max-degree and their variables' weight ($cwv_i$).
    for variable $v_i$ do do
        Prioritise best values ($bv_i$)
    end for
    SynCBJ(ao)
    if solution found by SynCBJ then
        return solution
    else
        return "unsolvable problem"
    end if
end if
```

- **Variable ordering:**
  - **Weights:** uses a combination of max-degree and constraint weights.
  - **Last weights:** as above but it uses the values of SingleDB-wd's constraint weights at the last cycle.
- **Value ordering:**

– **Sticking values:** the first variable value to be tried by SynCBJ is SingleDB-wd's best value for that variable, i.e. the one participating in the best solution found. This is inspired by [32] who used the last value assigned to a variable.

– **Selected sticking values:** uses the sticking value for that variable (i.e. the best value assigned by SingleDB-wd for that variable) only if that value led to no constraint violations in SingleDB-wd. Otherwise, it uses the first value in the domain as the sticking value.

Table 5.5 presents the median results over 100 runs.

	Number of Messages	**Number of Constraint Checks**
SynCBJ	262,178	1,344,941
Weights	41,725	289,065
Sticking values	35,199	239,827
Weights + selected sticking	28,907	212,382
Last weights	47,713	298,876
DBHyb	**22,259**	**173,825**

Table 5.5: Comparison of variable and value ordering heuristics ($n = 50$, $d = 10$, $p1 = 0.15$, $p2 = 0.4$).

All variants of *DBHyb* outperformed the base line SynCBJ. Last weights is the least efficient approach. Best weights improves on this since it gives SynCBJ more information about difficult variables since these will be the variables which have a high constraint weight even with a low number of overall constraint violations. As with *PenDHyb*, sticking values proves to be an effective addition. Whilst selected sticking performs quite well in this case, the version of *DBHyb* outlined above (i.e. best weights + sticking values for all variables) is the best performing version and the one used throughout the rest of this chapter.

Determining Optimal Number of Cycles for SingleDB-wd

The only parameter which needed to be determined in *DBHyb* was the number of cycles SingleDB-wd should run for optimal performance. This was determined in the same way as for *PenDHyb* (see section 5.3.1). The values for α, β and γ in the cutoff formula are given in Table 5.6 where the column heading indicates which metric is to be minimised.

	Randomly Generated Problems			*Graph Colouring Problems*		
optimise	**messages**	**constr. checks**	**both**	**messages**	**constr. checks**	**both**
α	-62.129	-67.388	-62.295	-50.288	-31.410	-38.943
β	3.260	1.660	2.920	0.000	0.000	0.000
γ	0.429	0.494	0.421	0.216	0.136	0.166
	Meeting Scheduling Problems					
optimise	**messages**	**constr. checks**	**both**			
α	-116.461	-117.758	-98.875			
β	19.781	21.281	17.438			
γ	0.254	0.131	0.182			

Table 5.6: Parameter values for α, β and γ in Equation (5.1).

5.4 Experimental Evaluation

We evaluated *PenDHyb* and *DBHyb* on **distributed randomly generated problems**, **distributed graph colouring problems** and **distributed meeting scheduling problems** against SynCBJ [102] and DisBOBT [22]. Our SynCBJ implementation was verified with the distributed randomly generated problems described in [102] ($n = 10$, $d = 10$, $p1 = 0.7$ and $p2 \in \{0.1, ..., 0.9\}$) whilst our DisBOBT implementation was verified with the implementation described in [22]. For both algorithms, the results obtained were at least as good as those reported by their authors.

We made modifications to SynCBJ and DisBOBT to ensure fair comparisons with our hyrbid approaches.

SynCBJ was modified to use max-degree variable ordering instead of lexicographic ordering obtaining substantially better results (see Table 5.7). Consequently, SynCBJ with max-degree variable ordering was used to compare its performance to PenDHyb's.

	Number of Messages									*Number of Constraint Checks*								
n	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
L	**10**	11	31	290	1611	1138	241	146	86	51	91	294	2,113	11,702	7,881	1,498	888	590
M	**10**	**10**	**15**	**57**	**358**	**183**	**103**	**46**	**30**	**46**	**74**	**149**	**520**	**2,952**	**1,484**	**783**	**381**	**281**

Table 5.7: Comparison of SynCBJ with lexicographic (L) and max-degree (M) variable orderings ($n = 10$, $d = 10$, $p1 = 0.7$, $p2 = 0.1...0.9$).

DisBOBT [22] was originally presented as a hybrid algorithm combining DisBO [22] and SBT [96]. However since its development, DisBO-wd has been shown to outperform DisBO [8] for DisCSPs with Complex Local Problems and SynCBJ to outperform SBT [102] for DisCSPs with one variable per agent. Consequently, we conducted an experiment with four variants of DisBOBT: (i) DisBO and SBT (DisBOBT); (ii) SingleDB-wd and SBT (*DisBOBTWD*); (iii) DisBO and SynCBJ (*DisBOCBJ*); (iv) SingleDB-wd and SynCBJ (*DisBOCBJWD*). We conducted experiments on: (i) randomly generated problems with 40 variables, 8 domain values, constraint density of 0.2 and constraint tightness of 0.6; (ii) graph colouring problems with 50 nodes and a degree of 2.4; (iii) meeting scheduling problems with 5 timeslots, 20 meetings, constraint density of 0.19 and max distance of 2. We considered a range of breakout steps (where local search is stopped and systematic search begins) between 100 and 1000 in steps of 100 for randomly generated problems and between 5 and 50 in steps of 5 for graph colouring problems and meeting scheduling problems. DisBO and SingleDB-wd are able to solve almost all problems if given a number of breakout steps greater than 50 and consequently we used a lower breakout steps threshold to measure the impact of the ordering schema on systematic search. The median results for 100 problems for each problem type are shown in table 5.8.

For randomly generated problems, *DisBOCBJWD* (i.e. SingleDB-wd and SynCBJ) substantially outperforms the other variants on both messages and constraint checks particularly as the number of breakout steps increases and SingleDB-wd is given more time to perfect its ordering scheme for systematic search. Results for graph colouring problems were similar where *DisBOCBJWD* was once again the optimal algorithm. With a very small amount of breakout steps, it was able to easily guide systematic search to a good ordering with very few messages and constraint checks. For meeting scheduling problems, similar trends were present where *DisBOCBJWD* outperformed the other DisBOBT variants. It should be noted here that SingleDB-wd imposes breakout steps more quickly than DisBO. Consequently, the DisBOBT variants using SingleDB-wd have their optimum performance at higher number of breakout steps than DisBO. Since *DisBOCBJWD* outperformed the other variants on the three problem types tested, we will evaluate *DisHyb* against *DisBOCBJWD*.

We evaluated *PenDHyb*, *DBHyb*, SynCBJ and *DisBOCBJWD* measuring: (i) the number of messages sent; (ii) the number of non-concurrent constraint checks performed. Note that the number of messages required for termination detection is not counted for any of the algorithms as reported by other researchers [96]. Although CPU time is not an es-

	Randomly Generated Problems									
n	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
	Number of Messages									
DisBOBT	3,658	3,720	3,626	4,020	3,865	4,088	4,006	4,047	3,556	3,698
DisBOBTWD	49,330	22,913	31,652	27,244	12,313	7,330	6,903	4,227	2,864	3,303
DisBOCBJ	**3,642**	3,823	**3,305**	3,788	3,402	4,086	4,408	4,563	4,052	3,695
DisBOCBJWD	4,581	**3,576**	4,556	**3,422**	**2,422**	**2,229**	**1,946**	**2,393**	**1,556**	**1,801**
	Number of Constraint Checks									
DisBOBT	46,446	49,725	45,257	52,054	51,679	50,464	53,516	51,316	46,690	50,359
DisBOBTWD	331,071	145,305	200,307	173,113	86,776	61,381	59,641	44,017	39,717	44,477
DisBOCBJ	47,208	47,283	41,600	48,635	42,544	51,902	53,438	60,191	52,627	50,601
DisBOCBJWD	**21,917**	**20,606**	**28,267**	**25,609**	**23,169**	**24,955**	**24,902**	**30,014**	**29,892**	**33449**
	Graph Colouring Problems									
n	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
	Number of Messages									
DisBOBT	287,423	157,282	80,316	77,167	30,684	9,793	9,266	7,779	9,714	8,762
DisBOBTWD	23,788	43,281	24,760	19,542	35,060	52,829	34,695	70,600	19,744	24,826
DisBOCBJ	1,529	2,341	2,910	3,893	4,032	5,197	5,987	6,256	6,798	7,882
DisBOCBJWD	**328**	**377**	**334**	**372**	**368**	**392**	**365**	**406**	**387**	**364**
	Number of Constraint Checks									
DisBOBT	864,762	434,988	223,887	219,381	85,383	18,614	18,926	14,155	17,514	16,913
DisBOBTWD	72,225	125,019	78,153	56,495	100,142	149,603	95,667	185,177	63,714	72,514
DisBOCBJ	3,742	5,243	6,144	8,102	8,048	9,878	11,586	12,010	12,540	14,923
DisBOCBJWD	**1,394**	**1,467**	**1,373**	**1,462**	**1,472**	**1,541**	**1,434**	**1,617**	**1,533**	**1,422**
	Meeting Scheduling Problems									
n	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
	Number of Messages									
DisBOBT	1,021	1,003	1,083	1,157	936	1,053	1,139	1,145	950	1,112
DisBOBTWD	1,586	414	994	921	1,531	1,138	778	595	445	246
DisBOCBJ	644	796	946	1,188	1,097	1,095	958	772	968	1,124
DisBOCBJWD	**120**	**96**	**112**	**111**	**112**	**138**	**140**	**154**	**173**	**190**
	Number of Constraint Checks									
DisBOBT	4,005	3,276	3,135	3,436	2,793	3,200	3,452	3,242	2,546	3,401
DisBOBTWD	7,114	1,850	5,073	4,820	8,412	4,972	4,118	2,842	2,570	1,729
DisBOCBJ	2,261	2,669	2,899	3,496	3,170	3,043	2,871	2,285	2,805	3,066
DisBOCBJWD	**796**	**704**	**786**	**740**	**795**	**928**	**1,133**	**1,181**	**1,341**	**1,459**

Table 5.8: Comparison of DisBOBT variants on randomly generated problems, graph colouring problems and meeting scheduling problems.

tablished measure for DisCSPs [56], we also measured it and the results obtained were consistent with the other measures used.

In both experiments, the cutoff number of cycles for minimizing both the number of messages and the number of constraints (see Equation (5.1) and Table 5.4 for *PenDHyb* and Table 5.6 for *DBHyb*) was used. For *DisBOCBJWD*, 900 breakout steps were permitted before termination for randomly generated problems and 50 for graph colouring problems and meeting scheduling problems. These parameters meant that SingleDB-wd ran for a similar length of time as the local search within our hybrid framework and was able to solve a similar amount of problems.

For small problems ($n < 30$ for random and meeting scheduling problems and $n < 100$ for graph colouring problems), SynCBJ easily solves them and, the use of *PenDHyb* or *DBHyb* leads to decreased performance for these problems. This is unsurprising given that systematic search is generally faster than local search for small problems [30].

Randomly Generated Problems

PenDHyb and *DBHyb* were evaluated against SynCBJ and *DisBOCBJWD* on a wide variety of randomly generated problems ($n \in \{30, 40, 50, 60\}$, $d \in \{8, 9, 10, 11, 12\}$, $p1 \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$ and $p2 \in \{0.1, 0.15, ..., 0.95\}$). The results presented here are at the phase transition point which represents hard problems for SynCBJ (see section 4.4). The results, shown in Table 5.9 for problems with ($n \in \{30, 40, 50, 60\}$, $d = 10$, $p1 = 0.15$ and $p2 = 0.6(30), 0.5(40), 0.45(50), 0.4(60)$), are median values over 100 problems.

N. messages	**solvable problems**				**unsolvable problems**			
n	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**
SynCBJ	2,301	22,590	262,178	1,897,645	**5,154**	58,395	557,360	3,069,301
DisBOCBJWD	6,264	37,882	2,451,565	*	13,244	166,239	3,915,508	*
PenDHyb	2,471	**17,439**	50,929	**277,437**	5,507	55,311	**451,507**	2,705,595
DBHyb	**2,271**	21,020	**22,259**	981,882	5,564	**51,018**	464,754	**2,537,364**
N. cnstr. checks	**solvable problems**				**unsolvable problems**			
n	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**
SynCBJ	**11,489**	119,209	1,344,941	10,421,510	**25,468**	294,393	2,924,331	17,153,384
DisBOCBJWD	63,146	222,689	12,490,214	*	94,108	868,154	20,434,334	*
PenDHyb	13,365	**108,311**	321,237	**1,877,084**	27,162	309,176	**2,357,739**	14,605,275
DBHyb	12,816	115,277	**173,825**	5,490,257	28,352	**252,638**	2,329,112	**14,017,517**

Table 5.9: Performance of SynCBJ, DisBOCBJWD, PenDHyb and DBHyb on randomly generated problems.

For solvable problems, *PenDHyb* is significantly more efficient than SynCBJ with performance difference increasing with the number of variables. *DBHyb* outperforms SynCBJ for both messages and constraint checks but is outperformed by *PenDHyb* for 40 and 60 variables. In one case (50 variables), *DBHyb* outperforms *PenDHyb* for both number of messages and constraint checks. *DisBOCBJWD* is uncompetitive and we were unable to record a result for 60 variables as the algorithm took over 24hrs without solving any problems [2]. For unsolvable problems, SynCBJ is marginally better on problems with 30 variables but *PenDHyb* is substantially better on problems with 40 or more variables. *DBHyb* outperforms SynCBJ and also outperforms *PenDHyb* for medium-sized to larger problems with the exception of problems with 50 variables where *PenDHyb* is more efficient. *DisBOCBJWD* remains uncompetitive as with solvable problems.

Graph Colouring Problems

The performance of *PenDHyb* and *DBHyb* was also evaluated against SynCBJ and *DisBOCBJWD* on distributed graph colouring problems ($nodes \in \{125, 150, 175, 200\}$, $d = 3$ and degree $k \in \{4.6, ..., 5.3\}$). These problems are of similar size to the ones used for the experiments on randomly generated problems above. Median values over 100 solvable problems and 100 unsolvable problems are shown in Table 5.10 for problems with a degree of 5.

N. messages	**solvable problems**				**unsolvable problems**			
n	**125**	**150**	**175**	**200**	**125**	**150**	**175**	**200**
SynCBJ	18,781	75,778	191,988	722,256	127,054	660,334	1,957,622	6,793,331
DisBOCBJWD	326,626	2,131,103	*	*	5,014,184	*	*	*
PenDHyb	**18,577**	60,005	161,213	463,601	**113,590**	**557,434**	1,849,564	5,357,801
DBHyb	19,040	**53,866**	**130,512**	**396,185**	116,731	641,682	**1,733,777**	**5,218,837**
N. cnstr. checks	**solvable problems**				**unsolvable problems**			
SynCBJ	**46,234**	178,942	477,713	1,750,199	309,383	1,587,410	4,518,670	15,694,031
DisBOCBJWD	787,372	4,946,810	*	*	11,587,907	*	*	*
PenDHyb	52,534	162,748	416,520	463,601	**281,142**	**1,327,274**	4,498,886	12,527,968
DBHyb	56,064	**150,880**	**348,521**	**967,257**	294,890	1,545,884	**4,041,727**	**12,047,485**

Table 5.10: SynCBJ, DisBOCBJWD, PenDHyb and DBHyb on graph colouring problems for degree = 5.

The results show that both *PenDHyb* and *DBHyb* are significantly more efficient for both solvable and unsolvable problems compared with SynCBJ and *DisBOCBJWD*. *DB-*

[2] All algorithms except *DisBOCBJWD* were able to solve 100 problems within 24hrs whilst *DisBOCBJWD* was not able to solve any.

Hyb is the best performing hybrid approach for solvable problems with 150 nodes or greater and unsolvable problems with 175 nodes or greater. Experiments for other degrees gave similar results, i.e. *PenDHyb* and *DBHyb* performed better, especially for graphs with a large number of nodes. Once again, *DisBOCBJWD* had such long execution times that we were unable to record results for all but the smallest problems. Those problems which took too long to solve are indicated by an asterix. Consequently, we propose that *DBHyb* is optimal for graph colouring problems. It would appear that SingleDB-wd is able to provide a better ordering than DisPeL for graph colouring problems which enables the systematic search to find a solution to the problem quicker.

Meeting Scheduling Problems

PenDHyb and *DBHyb* were also evaluated against SynCBJ and *DisBOCBJWD* on meeting scheduling problems. The results presented here are at the phase transition point which represents hard problems for SynCBJ (see section 4.4). The results, shown in Table 5.11 for problems with (number of meetings $\in \{30, 40, 50, 60\}$, timeslots $= 6$, max distance between locations $= 3$ and constraint density $= 0.24(30), 0.22(40), 0.16(50), 0.14(60)$), are median values over 100 problems.

N. messages	**solvable problems**				**unsolvable problems**			
n	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**
SynCBJ	2,281	16,982	33,250	119,988	8,744	12,729	42,843	81,263
DisBOCBJWD	8,063	156,863	578,836	569,110	23,728	143,295	*	*
PenDHyb	1,577	4,904	11,507	19,366	7,336	13,669	32,259	57,745
DBHyb	**1,061**	**2,631**	**5,025**	**7,622**	**6,708**	**9,792**	**246,98**	**38,141**
N. cnstr. checks	**solvable problems**				**unsolvable problems**			
n	**30**	**40**	**50**	**60**	**30**	**40**	**50**	**60**
SynCBJ	12,109	123,058	270,843	928,067	57,567	**84,394**	255,964	427,270
DisBOCBJWD	54,216	1,275,090	4,298,185	3,887,218	165,567	932,049	*	*
PenDHyb	**12,048**	**39,404**	**84,400**	136,789	**46,956**	95,558	198,880	305,291
DBHyb	18,513	52,188	116,462	**102,652**	51,272	96,839	**169,796**	**267,276**

Table 5.11: Performance of SynCBJ, DisBOCBJWD, PenDHyb and DBHyb on meeting scheduling problems.

For solvable scheduling problems, *DBHyb* is the optimal algorithm in terms of numbers of messages and also for large problems (60 variables) for constraint checks. *PenDHyb* is the optimal algorithm in terms of constraint checks for problems with 30 to 50 variables. SynCBJ and *DisBOCBJWD* become increasingly uncompetitive as the number of variables increases. For unsolvable scheduling problems, *DBHyb* remains the optimal algorithm in terms of number of messages (for 30 to 60 variables) and number of non-concurrent constraint checks (for 50 variables or greater). It would appear that *DBHyb* and *PenDHyb* are able to learn the challenging areas of the problem effectively and therefore direct the systematic search algorithm to a search faster than systematic search alone is able to do. There is only one case where neither *DBHyb* nor *PenDHyb* is the optimal algorithm i.e. 40 variables for unsolvable problems. However, in this case, *DBHyb* is the optimal algorithm for number of messages and offers a substantial performance improvement on messages over SynCBJ, which was the optimal algorithm for number of constraint checks. In general, it would appear that a weight-based hybrid approach (*DBHyb*) is optimal for meeting scheduling problems. Again, it would appear that SingleDB-wd is able to provide a better ordering than DisPeL for the systematic search.

5.5 Discussion

5.5.1 Analysing the Effectiveness of Using Information Learnt from Local Search in Systematic Search

An experiment was conducted on randomly generated problems to explore why our approach was significantly better than *DisBOCBJWD* and SynCBJ. The experiment included problems with 30, 40, 50 and 60 variables, both solvable and unsolvable randomly generated problems and measured the number of backjumps performed during search and, for *PenDHyb* and *DBHyb*, the percentage of cases where max-degree was insufficient to sort agents (i.e. tie-breaking using penalty counts was used) and the number of agents which changed position in the ordering (i.e. they are in a different position in the ordering from a max-degree ordering). The problem characteristics were the same as in the experiments conducted in section 5.4 i.e. ($n \in \{30, 40, 50, 60\}$, $d = 10$, $p1 = 0.15$ and $p2 = 0.6(30), 0.5(40), 0.45(50), 0.4(60)$). Those problems which *DisBOCBJWD* took too long to solve are indicated by an asterix. The median results over 100 runs are shown in Table 5.12.

The results show that, for all problems with 40 or more variables, *PenDHyb* backjumps

		Solvable Problems			Unsolvable Problems		
	n. vars	**n. backjumps**	**% ties**	**n. changes**	**n. backjumps**	**% ties**	**n. changes**
SynCBJ	30	812	-	-	1,896	-	-
DisBOCBJWD	**30**	**572**	-	-	3,510	-	-
PenDHyb	**30**	765	30.21	29	**1,703**	**30.67**	**29**
DBHyb	30	655	27.75	29	1,932	27.66	29
SynCBJ	40	7,636	-	-	19,108	-	-
DisBOCBJWD	40	12,006	-	-	59,486	-	-
PenDHyb	**40**	**4,134**	**30.93**	**39**	17,963	29.68	39
DBHyb	**40**	6,930	30.84	39	**16,521**	**29.97**	**39**
SynCBJ	50	85,061	-	-	183,963	-	-
DisBOCBJWD	50	925,424	-	-	1,491,140	-	-
PenDHyb	**50**	**38,011**	**28.45**	**49**	**138,817**	**28.48**	**49**
DBHyb	50	56,267	29.21	49	141,731	29.54	49
SynCBJ	60	625,158	-	-	930,641	-	-
DisBOCBJWD	60	*	-	-	*	-	-
PenDHyb	**60**	**61,856**	**26.99**	**59**	847,950	27.45	59
DBHyb	**60**	313,166	28.6	59	**847,811**	**28.91**	**59**

Table 5.12: Backjumping properties of SynCBJ, DisBOCBJWD, PenDHyb and DBHyb.

on significantly fewer occasions than all other algorithms for solvable problems. *DBHyb* backjumps on significantly fewer occasions than all other algorithms for unsolvable problems for 40 and 60 variables whilst *PenDHyb* backjumps less for 30 and 50 variables. Both *PenDHyb* and *DBHyb* perform less backjumps than *DisBOCBJWD* and SynCBJ for all problems with 40 or more variables. It would appear that the penalties on values strategy is more efficient for solvable problems whilst the breakout strategy is more efficient for unsolvable problems. *DisBOCBJWD* is only optimal for the number of backjumps for solvable problems with 30 variables. However, so much effort is required to determine an ordering which minimises backjumps that it actually performs the worst out of all algorithms when measured by number of messages and non-concurrent constraint checks. It is interesting to note that for both *PenDHyb* and *DBHyb* the difference with SynCBJ is profound for unsolvable problems given that DisPeL and SingleDB-wd themselves cannot detect unsolvability. However, it appears that DisPeL and SingleDB-wd can learn useful information thereby allowing systematic search to determine quicker that the problem has no solution. In *PenDHyb* and *DBHyb*, penalties or constraint weights respectively are used to break max-degree ties in 25-30% of variable selections which allows the ordering to be altered substantially. Indeed, almost all agents change position.

5.5.2 Longer Executions of Local Search

We also evaluated longer executions of DisPeL and SingleDB-wd as part of the *PenDHyb* and *DBHyb* algorithms respectively so that local search solved the vast majority of problems with systematic search only solving the very few problems which local search could not solve. These longer executions are not suitable for unsolvable problems since local search algorithms cannot detect unsolvability. We determined the optimal cutoff point for longer executions by running a series of experiments on both distributed randomly-generated problems, graph colouring problems and meeting scheduling problems.

Randomly Generated Problems: We conducted experiments to determine the optimal cutoff point for solvable randomly-generated problems for 30 to 60 variables in steps of 10, 10 domain values, number of constraints equal to 3 times the number of variables and constraint tightness of 0.5. We used these parameters since local search had performed well on these parameters [8].

We ran experiments with cutoff points between $2n$ and $30n$ in steps of $2n$ where n is the number of variables. Previously, we had tuned the formula of *PenDHyb* and *DBHyb* on cutoff points less than $2n$ (since the cutoff recommended was always substantially less than $2n$) and therefore this seemed an appropriate boundary between local search as a heuristic and local search as more of a problem solver. At the other end of the scale, we chose $30n$ since the performance of increased local search had tailed by this point and therefore additional cycles did not appear to offer substantial benefits.

For each cutoff point, we ran 50 runs on 100 solvable problems. We measured the number of messages which our hybrid approach used at each cutoff point (for example, 43,200 messages at $2n$ cycles for 60 variables) and determined the number of problems solved using that number of messages. This gave a measurement which indicates the **effectiveness of continuing to run the local search** phase over the **effectiveness of reordering the variables and starting backjumping search**. Further points were measured for number of messages by adding the number of messages used at $2n$ to the current amount (i.e. for $2n$ cycles and 60 variables the sequence would be $\{43,200, 86,400, 129,600, ...\}$ until all potential cutoffs (between $2n$ and $30n$ cycles) had solved 100% of problems. We took the median of these 50 runs and then took a ranking of each of the cycle cutoffs at each measuring point. We repeated the analysis for number of constraint checks to ensure that our conclusions were robust. For example, if only two problems were solved with 10,000 messages and they used 5,600 and 3,200 messages respectively, the cumulative message cost would be 8,800 messages and the percentage of problems solved would be 2%. The cumulative total message cost measures the total cumulative cost which increases between different cutoff brackets. If we next measured the number of messages at 20,000 and another problem was solved with 14,000 messages then the cumulative for 20,000 messages would be 14,000 messages but the cumulative total would be 22,800 messages.

Consider the sample data for 30 variables in table 5.13 which compares by number of messages for the *DBHyb* algorithm. Specifically, we measure in the first column, the maximum number of messages which SingleDB-wd would use if it ran for the full bounded number of cycles i.e. if SingleDB-wd ran for $2n$ cycles, it would send 10,800 messages and if SingleDB-wd ran for $4n$ cycles, it would send 21,600 messages. At each of these cutoff points (i.e. 10,800 and 21,600 and so on), we measured the percentage of problems solved by SingleDB-wd. In the case of 10,800 messages and $2n$ as the bounded number of cycles, this would be the total percentage of problems solved by SingleDB-wd (64%) as all other problems would then be solved by the systematic search guided by the information learnt from SingleDB-wd. In the case of 21,600 messages and $4n$ as the bounded number of cycles, this would measure the percentage of problems solved by SingleDB-wd with under 10,800 messages (58%). Note that where the number of messages is less than or equal to the number of messages used at a bounded cycle (e.g. 10,800 for $2n$ and 10,800 and 21,600 for $4n$), we are comparing different executions of SingleDB-wd across the columns of the table and therefore the percentage of problems solved varies according to the random instantiation. Each percentage is a median of the percentage of problems solved during 50 runs. The cumulative message column (e.g. 691,200 for $2n$ for 10,800 messages) measures the cumulative number of messages that have been used to solve the problems within that cutoff range (i.e. between 0 and 10,800 messages) whilst the cumulative total message column gives a running total of the number of messages used over all cutoff brackets. We can therefore say that the $2n$ version solved 64% of problems in the SingleDB-wd phase and that this increases to 84% once systematic search has had a suitable run (i.e. it has 10,800 messages on its own + the 10,800 messages which SingleDB-wd used). In this particular example, we can say that $4n$ incurs less messages initially for those problems that it solves but that $2n$ solves the most problems overall and therefore is the better approach.

Messages	**2n**	**2n Cum**	**2n Cum T**	**4n**	**4n Cum**	**4n Cum T**
10,800	64%	691,200	691,200	58%	626,400	626,400
21,600	84%	432,000	1,123,200	83%	540,000	1,166,400
32,400	90%	194,400	1,317,600	90%	226,800	1,393,200
43,200	93%	129,600	1,447,200	93%	129,600	1,522,800
54,000	95%	108,000	1,555,200	95.5%	135,000	1,657,800
64,800	97%	129,600	1,684,800	97%	97,200	1,755,000
75,600	97.5%	37,800	1,722,600	98%	75,600	1830600
86,400	98%	43,200	1,765,800	98%	0	1,830,600
97,200	98%	0	1,765,800	99%	97,200	1,927,800
108,000	99%	108,000	1,873,800	99%	0	1,927,800
118,800	99%	0	1,873,800	99%	0	1,927,800
129,600	99%	0	1,873,800	99%	0	1,927,800
140,400	99%	0	1,873,800	99%	0	1,927,800
151,200	99%	0	1,873,800	99%	0	1,927,800
162,000	99%	0	1,873,800	99.5%	81,000	2,008,800
172,800	99%	0	1,873,800	99.5%	0	2,008,800
183,600	99%	0	1,873,800	100%	91,800	2,100,600
194,400	100%	194,400	2,068,200	100%	0	2,100,600

Table 5.13: Sample data for longer executions of local search for randomly generated problems with 30 variables and DBHyb.

We summarise our findings for randomly generated problems for both *PenDHyb* and *DBHyb* in table 5.14 [3]. A cutoff of $< 2n$ indicates that the formula described in section 5.3.1 for *PenDHyb* and section 5.3.2 for *DBHyb* should be used.

	PenDHyb		**DBHyb**	
Num Variables	**Messages**	**Constraint Checks**	**Messages**	**Constraint Checks**
30	$< 2n$	$< 2n$	2n	$< 2n$
40	2n	$< 2n$	4n	2n
50	22n	4n	24n	6n
60	22n	12n	18n	16n

Table 5.14: The optimal cutoff for particular number of variables for PenDHyb and DBHyb on randomly generated problems.

For randomly generated problems, we found that higher cutoffs were beneficial when the number of variables was larger. This is consistent with our findings that systematic search substantially increases as the number of variables increases and so longer runs of

[3] Full data for these experiments can be found at http://www.comp.rgu.ac.uk/staff/dl/cutoffs.html

local search will become more desirable.

Graph Colouring Problems: We also conducted experiments to determine an optimal cutoff for solvable graph colouring problems with 125 to 200 variables in steps of 25 and a degree between 4.3 and 5.3 in steps of 0.1. These parameters were chosen since local search has performed well on those parameters solving most but not all problems [8].

We ran experiments with cutoffs of $0.5n$, $0.75n$ and $1n$. Previously, *PenDHyb* and *DBHyb* had been tested on cutoffs lower than $0.6n$ and since the cutoff recommended was never higher than $0.5n$ - $0.5n$ indicates where the cutoff formula should be used. There were no cases where $1n$ was optimal and so this was the highest cutoff used. We performed identical experiments for each cycle cutoff point as per those described for randomly generated problems. We summarise our findings for graph colouring problems for both *PenDHyb* and *DBHyb* in table 5.15 [4] . A cutoff of $< 0.5n$ indicates that the formula described in section 5.3.1 for *PenDHyb* and section 5.3.2 for *DBHyb* should be used.

For graph colouring problems, we found that higher cutoffs were required for problems which had a large number of variables and a high degree. We found this to be consistent with our findings for graph colouring problems in that these were where SynCBJ often did badly and so longer runs of local search are preferable.

Meeting Scheduling Problems: We also conducted experiments to determine an optimal cutoff for solvable meeting scheduling problems with 30 to 60 variables in steps of 10, constraint density between 0.1 and 0.24 in steps of 0.02, number of timeslots between 5 and 6 and maximum distance between 2 and 3. These parameters were identical to those used for determining the optimal cutoff formula values in section 5.3.1 for *PenDHyb* and section 5.3.2 for *DBHyb*.

We ran experiments with cutoffs between $2n$ and $10n$ in steps of $2n$. Previously, *PenD-Hyb* and *DBHyb* had been tested on cutoffs lower than $2n$ (since the cutoff recommended was always substantially lower than $2n$) therefore $2n$ indicates where the cutoff formula should be used. There were no cases where $10n$ was optimal and so this was the highest

[4] Full data for these experiments can be found at http://www.comp.rgu.ac.uk/staff/dl/cutoffs.html

		PenDHyb		**DBHyb**	
Num Nodes	**Degree**	**Messages**	**Constraint Checks**	**Messages**	**Constraint Checks**
125	4.3	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
125	4.4	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
125	4.5	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
125	4.6	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
125	4.7	$< 0.5n$	$< 0.5n$	0.5n	$< 0.5n$
125	4.8	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
125	4.9	$< 0.5n$	$< 0.5n$	0.5n	$< 0.5n$
125	5.0	$< 0.5n$	$< 0.5n$	0.5n	$< 0.5n$
125	5.1	$< 0.5n$	$< 0.5n$	0.5n	$< 0.5n$
125	5.2	$< 0.5n$	$< 0.5n$	0.5n	$< 0.5n$
125	5.3	$< 0.5n$	$< 0.5n$	0.5n	$< 0.5n$
150	4.3	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
150	4.4	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
150	4.5	$< 0.5n$	$< 0.5n$	0.5n	$< 0.5n$
150	4.6	0.5n	$< 0.5n$	0.5n	0.5n
150	4.7	0.5n	0.5n	0.5n	0.5n
150	4.8	0.5n	0.5n	0.5n	0.5n
150	4.9	0.5n	0.5n	0.5n	0.5n
150	5.0	0.5n	0.5n	0.5n	0.5n
150	5.1	0.5n	0.5n	0.5n	0.5n
150	5.2	0.5n	$< 0.5n$	0.5n	$< 0.5n$
150	5.3	0.5n	$< 0.5n$	0.5n	$< 0.5n$
175	4.3	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
175	4.4	$< 0.5n$	$< 0.5n$	0.5n	0.5n
175	4.5	0.5n	$< 0.5n$	0.5n	0.5n
175	4.6	0.5n	0.5n	0.5n	0.5n
175	4.7	0.5n	0.5n	0.5n	0.5n
175	4.8	0.5n	0.5n	0.75n	0.5n
175	4.9	0.75n	0.75n	0.75n	0.5n
175	5.0	0.75n	0.5n	0.75n	0.5n
175	5.1	0.75n	0.75n	0.75n	0.5n
175	5.2	0.5n	0.5n	0.75n	0.5n
175	5.3	0.75n	0.5n	0.75n	0.5n
200	4.3	$< 0.5n$	$< 0.5n$	$< 0.5n$	$< 0.5n$
200	4.4	$< 0.5n$	$< 0.5n$	0.5n	$< 0.5n$
200	4.5	0.5n	$< 0.5n$	0.5n	0.5n
200	4.6	0.5n	0.5n	0.5n	0.5n
200	4.7	0.5n	0.5n	0.75n	0.5n
200	4.8	0.75n	0.75n	0.75n	0.75n
200	4.9	0.75n	0.75n	0.75n	0.75n
200	5.0	0.75n	0.75n	0.75n	0.75n
200	5.1	0.75n	0.75n	0.75n	0.75n
200	5.2	0.75n	0.75n	0.75n	0.75n
200	5.3	0.75n	0.75n	0.75n	0.75n

Table 5.15: The optimal cutoff for particular number of nodes for PenDHyb and DBHyb on graph colouring problems.

cutoff used. We performed identical experiments for each cycle cutoff point as per those described for randomly generated problems and graph colouring problems. We summarise our findings for meeting scheduling problems for both *PenDHyb* and *DBHyb* in table 5.16 where the number of timeslots was 6 and the maximum distance was 3 [5]. A cutoff of $< 2n$ indicates that the formula described in section 5.3.1 for *PenDHyb* and section 5.3.2 for *DBHyb* should be used.

		PenDHyb		**DBHyb**	
Num Vars	**Con Density**	**Messages**	**Constraint Checks**	**Messages**	**Constraint Checks**
30	0.1	2n	2n	2n	2n
30	0.12	2n	2n	2n	2n
30	0.14	8n	2n	2n	2n
30	0.16	2n	$< 2n$	2n	2n
30	0.18	2n	2n	2n	2n
30	0.2	2n	2n	2n	2n
30	0.22	2n	2n	2n	2n
30	0.24	2n	2n	2n	2n
40	0.1	2n	2n	2n	2n
40	0.12	2n	2n	2n	2n
40	0.14	2n	2n	2n	2n
40	0.16	2n	2n	2n	2n
40	0.18	4n	4n	2n	2n
40	0.2	4n	2n	2n	2n
40	0.22	4n	2n	4n	2n
40	0.24	4n	2n	6n	4n
50	0.1	6n	2n	2n	2n
50	0.12	4n	2n	4n	4n
50	0.14	6n	4n	4n	2n
50	0.16	6n	6n	6n	6n
50	0.18	6n	6n	6n	6n
50	0.2	6n	6n	4n	4n
50	0.22	2n	2n	8n	6n
50	0.24	6n	6n	4n	2n
60	0.1	2n	2n	8n	4n
60	0.12	6n	6n	8n	8n
60	0.14	$< 2n$	$< 2n$	$< 2n$	$< 2n$
60	0.16	$< 2n$	$< 2n$	$< 2n$	$< 2n$
60	0.18	4n	$< 2n$	8n	$< 2n$
60	0.2	2n	2n	4n	4n
60	0.22	2n	2n	2n	$< 2n$
60	0.24	4n	2n	2n	2n

Table 5.16: The optimal cutoff for particular number of meetings for PenDHyb and DBHyb on meeting scheduling problems.

For meeting scheduling problems, we found that the cutoff varied according to the number of variables and constraint density. A high cutoff was found to be beneficial for those problems with larger number of variables and lower constraint densities.

[5]Full data for these experiments can be found at http://www.comp.rgu.ac.uk/staff/dl/cutoffs.html

5.6 Contributions

The following contributions have been made:

1. A knowledge-based hybrid approach to Distributed Constraint Satisfaction (*DisHyb*) for fine-grained DisCSPs which uses information learned from the local search phase to guide the systematic search phase.

2. Two implementations of the *DisHyb* approach: *PenDHyb* using the penalty-on-values local search strategy and *DBHyb* using the breakout local search strategy.

3. An alternative hybrid algorithm entitled *DisPBJ* which also uses penalties on values and combines a distributed local search (DisPeL) with a distributed systematic search (Distributed Backjumping). A description of DisPBJ is in Appendix A.

4. A formula has been derived which determines the optimal cutoff for three problem areas (randomly generated, graph colouring and meeting scheduling DisCSPs). The same methodology can be employed for other problem areas. Consideration has also been given for longer runs of local search for larger solvable problems.

5. New variants of the DisBOBT algorithm [22] including *DisBOCBJWD* which combines SingleDB-wd and SynCBJ. We have shown that this algorithm outperforms the original DisBOBT algorithm in three problem classes.

5.7 Summary

In this chapter, a hybrid approach to Distributed Constraint Satisfaction (*DisHyb*) has been presented. This approach uses knowledge from a local search algorithm to learn about the problem and utilises this knowledge to guide a systematic search algorithm. We have presented two instances of our approach which differ in the strategy used by the local search algorithm: (i) *PenDHyb* - uses penalties on values as the local search strategy (DisPeL) with a systematic search algorithm (SynCBJ); (ii) *DBHyb* - uses breakout as the local search strategy (SingleDB-wd) with a systematic search algorithm (SynCBJ). In all cases, the performance of our hybrid approach was significantly better than that of the systematic search algorithm on its own for large, difficult problems. *DBHyb* was the best algorithm, in general, for graph colouring problems and meeting scheduling problems whilst both algorithms performed well on randomly generated problems. In addition, we have shown that *DisBOCBJWD* is an improved hybrid approach over DisBOBT and that our hybrid approach outperforms both algorithms. In the next chapter, we extend our *DisHyb* approach for DisCSPs with complex local problems where each agent has more than one variable per agent.

Chapter 6

Multi-Hyb - Hybrid Framework for Solving DisCSPs with Complex Local Problems

6.1 Background and Motivation

In the previous chapter, we introduced a hybrid approach for DisCSPs with one variable per agent. However, some DisCSPs are **coarse-grained** consisting of a set of inter-related sub-problems (complex local problems). In this chapter, a new hybrid approach is presented which learns solutions to complex local problems at the same time as learning difficult variable and best value information for the global problem. The information learnt is used to provide potential solution values and variable ordering information for a distributed systematic search. New algorithms to determine solutions to complex local problems and to solve the global problem.

An overview of the approach is shown in figure 6.1.

The Multi-Hyb approach is an adaption of the DisHyb approach (presented in figure 5.1) for DisCSPs with complex local problems. As with DisHyb, Multi-Hyb has two phases. In the first phase, centralised systematic searches are added to the distributed local search from DisHyb to find all solutions to complex local problems of external relevance.

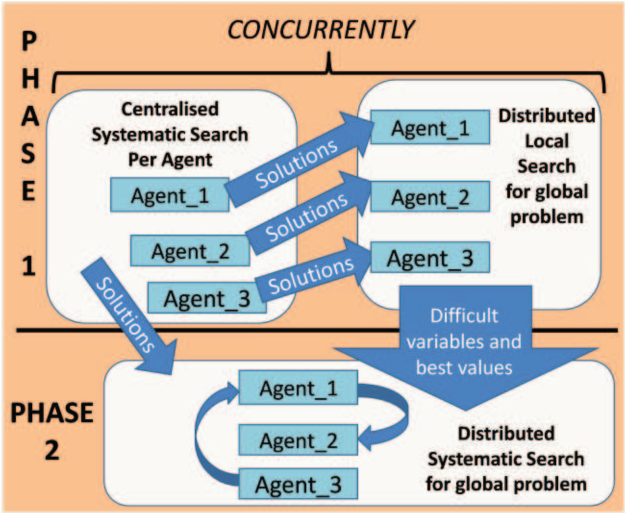

Figure 6.1: The Multi-Hyb approach.

The distributed local search now gathers knowledge about difficult variables and values in the global problem. In the second phase (as with DisHyb), distributed systematic search is run to solve the global problem using agents which have been reordered according to the difficult variables and values learnt by distributed local search. Therefore, the main differences between DisHyb and Multi-Hyb are: (i) the addition of centralised systematic searches to cope with the complex local problems; (ii) distributed local search and distributed systematic search now concentrate on the global problem. Multi-Hyb attempts to reduce the overall communication costs between agents. This is done at the potential cost of having to store a large number of solutions for complex local problems.

We present two implementations of our approach: *Multi-Hyb-Pen* uses penalties on values as the local search strategy and *Multi-Hyb-DB* uses constraint weights as the local search strategy.

An overview of the approach and algorithms presented in this chapter is shown in table 6.1.

Approach	**Algorithm**	**Local Search Strategy**
Multi-Hyb	Multi-Hyb-Pen	Penalties on values
Multi-Hyb	Multi-Hyb-DB	Constraint weights (‘Breakout’)

Table 6.1: Chapter Overview.

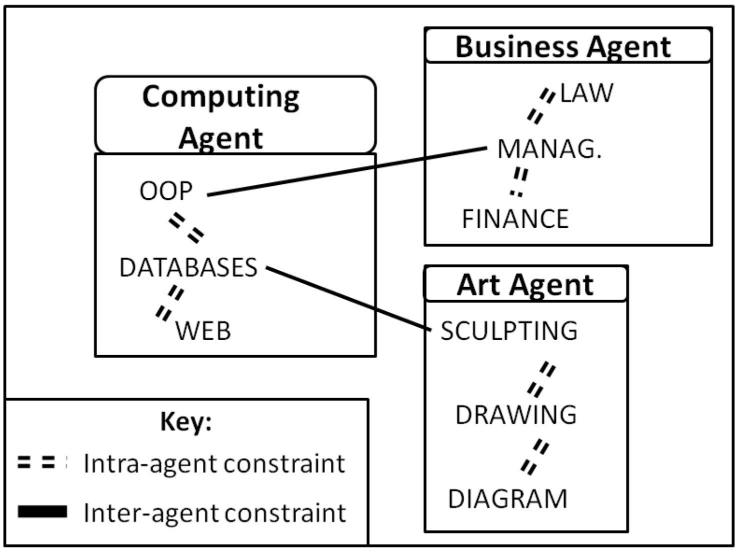

Figure 6.2: A scheduling DisCSP with complex local problems.

6.2 Description of approach

Complex local problems are linked together by a set of constraints which relate variables in two or more local problems. In section 4.3, two sets of constraints in a DisCSP with complex local problems were introduced: C_{intra} containing the **intra-agent constraints** and C_{inter} containing the **inter-agent constraints**. Specifically, we are interested in **naturally distributed subproblems** i.e. those for which an imbalance exists between inter-agent and intra-agent constraints, with a higher number of the latter. For example, a University department will be responsible for scheduling most of its classes (its complex local problem) but will have to negotiate with other departments for classes where teaching is shared between departments (inter-agent constraints between complex local problems). It will also need to find classrooms for the classes, which may also be wanted at the same time by other departments.

Multi-Hyb (see Algorithm 4) is a novel two-phase complete distributed hybrid approach for solving DisCSPs with complex local problems which are naturally distributed, i.e. with a high intra-agent to inter-agent constraint ratio. In order to explain each phase and the interaction between the two phases, a very simple university timetabling DisCSP with complex local problems is used (see Figure 6.2).

A University has three schools: Computing, Business and Art. Each school has a number of courses. Schools teach a number of modules. Courses in a school can consist of modules taught by that school or modules taught by other schools (external modules). For example, the Computing school has two courses: C1 and C2. C1 takes the modules Databases, OOP and Management (from the Business school). C2 takes the modules Databases and Web. The Business school has one course (B1) which takes the modules Law, Management and Finance. The Art school has three courses: A1, A2 and A3. A1 takes the modules Databases (from the Computing School) and Sculpting. A2 takes the modules Sculpting and Drawing. A3 takes the modules Drawing and Diagram. Two modules which share a common course cannot be timetabled at the same time. For simplicity, it is assumed that classes can only be scheduled at 9am, 10am, 11am and 12noon on Mondays. Room, resources or lecturer availability are not considered for this simple example. When preparing their individual timetable, schools must consider their internal modules (represented by their intra-agent constraints) as well as ensuring that their times do not clash with external modules (represented by inter-agent constraints).

Algorithm 4 Multi-Hyb

Initialise all agents with their subproblem and let phase1 $\leftarrow$ true
while (phase1) CONCURRENTLY **do**
    for each agent a_i **do**
        Run centralised systematic search and dynamically pass local problem solutions to distributed local search
        if an agent's centralised systematic search fails to find a solution **then**
            return "Problem is unsolvable."
        end if
    end for
    Run distributed local search on **inter-agent constraints**.
    if local search finds solution S **then**
        Return S.
    end if
    if all centralised systematic searches have found all solutions of **external relevance then**
        phase1 $\leftarrow$ false
    end if
end while
Run distributed systematic search algorithm using solutions found by centralised systematic searches and knowledge learnt by distributed local search.

In Phase 1, each agent finds all 'relevant' (**non-interchangeable**) solutions to its complex local problem using a centralised systematic search. The notion of **interchangeable**

local assignments was introduced by Burke [13]. For example, in the simple scheduling problem described (see figure 6.2), the variable Web is not involved in any inter-agent constraints. Consequently, only the variables OOP and $Databases$ have **external relevance** i.e. inter-agent constraints. Consequently, the solutions ($OOP = 9AM$, $Databases = 10AM$, $Web = 9AM$) and ($OOP = 9AM$, $Databases = 10AM$, $Web = 11AM$) are identical from an external perspective (agents Business and Art) when solving inter-agent constraints i.e. they are interchangeable and, therefore, only one of them is required. While agents are concurrently searching for solutions to their complex local problem and after each agent has found at least one solution, a distributed local search attempts to find a solution to the global problem. Phase 1 finishes when: (i) an agent determines that there is no solution to its complex local problem and, therefore, the overall problem is unsolvable; (ii) all agents have found all 'relevant' local solutions or; (iii) local search finds a solution to the global problem. If a solution is not found, and no unsolvability has been detected by a centralised systematic search, *Multi-Hyb* starts Phase 2. Phase 2 uses the knowledge learnt from Phase 1 to drive a distributed systematic search algorithm.

An overview of the properties of the different components of *Multi-Hyb* is given in table 6.2.

Phase	**Component**	**Variables**	**Domains**	**Constraints Considered**	**Knowledge Exchanged**
1	Centralised Systematic Searches	All variables in an agent	Static	Intra-agent constraints	Solutions to complex local problems passed to distributed local search and distributed systematic search.
1	Distributed Local Search	One complex variable per agent	Dynamic	Inter-agent constraints	Solutions to complex local problems from centralised systematic searches. Knowledge about difficult variables and best values passed to distributed systematic search.
2	Distributed Systematic Search	One complex variable per agent	Static	Inter-agent constraints	Solutions to complex local problems from centralised systematic searches. Knowledge about difficult variables and best values from distributed local search.

Table 6.2: Overview of Multi-Hyb components.

6.2.1 Completeness and Termination

In phase 1, any centralised systematic search algorithm can be used to find all non-interchangeable solutions. This type of algorithm is complete with respect to external variables. If the distributed local search is unable to find a solution, a distributed systematic search algorithm will be run in phase 2. This algorithm is complete so it will either return a solution or state that the problem is unsolvable. Therefore, the *Multi-Hyb* framework is complete.

Each instance of the centralised systematic search terminates when either: (i) it has found all non-interchangeable solutions to its local problems; (ii) it finds that it has no solution to its local problem and has informed all other agents; (iii) one of the agents sends a message stating that the problem is unsolvable; (iv) distributed local search sends a message stating that it has found a solution.

Distributed local search stops when either: (i) it has found a solution; (ii) all instances of centralised systematic searches have terminated.

Distributed systematic search terminates when either: (i) it has found a solution; (ii) detected that the problem is unsolvable.

Since centralised systematic search, distributed local search and distributed systematic search terminate, *Multi-Hyb* also terminates.

6.3 Implementations

We present two implementations of our *Multi-Hyb* approach which differ in the strategy used by the distributed local search. *Multi-Hyb-Pen* uses penalties on values whilst *Multi-Hyb-DB* uses the breakout strategy.

6.3.1 Multi-Hyb-Pen

Multi-Hyb-Pen uses *SEBJ* (see below) as the centralised systematic solver, *DisPeL-1C* (see below) as the distributed local search algorithm and *SynCBJ-CLP* (see below) as the distributed systematic search algorithm.

SEBJ

We developed *SEBJ* (see Algorithm 5) which is a systematic search algorithm that finds all **non-interchangeable solutions** to a complex local problem, i.e. the set of all solutions which differ on at least one value for an external variable (a variable linked by a constraint to another complex local problem). The algorithm uses SynCBJ [102] for finding the first solution and SBT [96] after the first solution has been found until a backtrack to the first agent is reached. At this point, the algorithm switches again to SynCBJ. This switching to backtracking is required since SynCBJ may miss solutions. Particularly, SynCBJ will generate a nogood after the last value is attempted for the last externally relevant variable. If this backjump is generated when a solution was found with one of the values of the externally relevant variable, then the backjump will only be decided according to values which caused constraint violations. Consequently, a backjump may be recommended to a point earlier in the search than guarantees all solutions to be found and so solutions are missed. The use of backtracking avoids this problem.

For example, *SEBJ* running on the Computing agent (see figure 6.2) would only consider solutions where the values of variables OOP and $Databases$ differ since they are the only variables of **external relevance** for that agent. Thus solutions ($OOP = 9AM$, $Databases = 10AM$, $Web = 9AM$) and ($OOP = 9AM$, $Databases = 10AM$, $Web = 11AM$) are identical when solving **inter-agent constraints** i.e. they are **interchangeable** and, therefore, only one of them is required.

SEBJ orders variables of **external relevance** before other variables. For example, in the Computing agent, a possible ordering would be OOP, $Databases$ and Web since both OOP and $Databases$ are externally relevant. When *SEBJ* finds the first solution to the problem using backjumping, it records the solution and then tries to find other solutions by restarting the search from the next value in the last externally relevant variable's domain. For example, if the first solution was ($OOP = 9AM$, $Databases = 10AM$, $Web = 9AM$) for the Computing agent, the algorithm would restart from the partial solution ($OOP = 9AM$, $Databases = 11AM$) in the search for a new solution. It would not consider ($OOP = 9AM$, $Databases = 10AM$, $Web = 11AM$) since the change (i.e. $Web = 11AM$) is not of **external relevance**, so solution ($OOP = 9AM$, $Databases = 10AM$, $Web = 9AM$) can be used. The generation of only **non-interchangeable solutions** to complex local problems can significantly reduce computational cost. Although the idea of

Algorithm 5 *SEBJ*

```
initialise - order external variables before internal variables
Set solutionFound ← false and solutionCount ← 0
for each variable v_i do
    for each value d_i in variable v_i's domain do
        if all higher priority constraints are satisfied then
            if solutionFound = true OR solutionFound = false AND all higher priority nogoods
            are not consistent with all variable values then
                assign value d_i to variable v_i
                return to variable for loop.
            end if
        else if solutionFound = false then
            for each higher priority constraint which is violated do
                Add the variable/value pair to a nogood for value d_i to variable v_i
            end for
        end if
    end for
    if variable v_i has no assigned value then
        if first variable is v_i and solutionCount = 0 then
            return "unsolvable problem"'
        else if first variable is v_i then
            return "all solutions found"
        else if solutionFound = false then
            Create variable v_i's conflict set with all variables involved in nogoods for values of
            variable v_i
            Backjump to lowest priority variable in the conflict set.
        else if solutionFound = true then
            Backtrack to previous variable in for loop.
            if lowest priority variable is first variable then
                set solutionFound ← false.
            end if
        end if
    end if
end for
Add solution to solution store.
Set solutionFound ← true and increment solutionCount
Restart for loop with variable v_i as last variable which has external links with value d_i as the
next value in its domain.
```

interchangeability has been proposed before using a branch and bound algorithm which rejects interchangeable solutions [13], *SEBJ* is different in that it does not even consider the exploration of the search space where there may only be interchangeable solutions.

SEBJ is sound and complete with regard to identifying all solutions of **external relevance**. Since the underlying algorithm is SynCBJ which is complete, the first solution is always guaranteed to be found. After this first solution is found, all values for externally relevant variables are considered through backtracking until backtracking reaches the first agent again. This use of backtracking guarantees that solutions are not missed because of nogoods imposed by SynCBJ. Once we reach the first agent, we are running a SynCBJ on a different part of the tree and are therefore guaranteed completeness on that part of the tree. Therefore, all variable value combinations belonging to externally relevant variables will be discovered.

In further experiments (see Appendix B), the use of forward checking (FC) for external variables in *SEBJ* was analysed. Forward checking has often proved to be beneficial in reducing NCCCs [3] and therefore it may have been useful in reducing the potentially large NCCCs required for finding all solutions of external relevance. The results showed that FC can generally improve performance but on other occasions, FC can lead to additional NCCCs. This is particularly the case when *DisPeL-1C* solves the problem in phase 1 before *SEBJ* finds all external solutions. This would appear to contradict previous studies on forward checking where it could always reduce NCCCs. However, the interaction between *SEBJ* with forward checking and the concurrent *DisPeL-1C* must be considered. Specifically, forward checking requires a large number of NCCCs near the top of the backtracking tree in order to prune branches of the tree earlier than is otherwise possible with backtracking search. This will ultimately lead to a reduction of the NCCCs once the whole tree has been explored. However, if a global solution to the problem is found quickly by *DisPeL-1C*, then the whole tree will not be explored. Consequently, the additional NCCCs to prune the tree by forward checking are wasted (because the whole tree is not explored) and forward checking results in additional NCCCs rather than less.

DisPeL-1C

We have developed DisPeL-1C which is a penalty-based distributed local search algorithm which is used to check the consistency of inter-agent constraints. *DisPeL-1C* substantially differs from Stoch-DisPeL as follows; (i) *DisPeL-1C* **continuously** imposes penalties when values are inconsistent without waiting until a quasi-local-minimum is detected; (ii) *DisPeL-1C*'s variables are **complex**, each representing all **externally relevant** variables for a complex local problem; (iii) In *DisPeL-1C* variable values are dynamically added to their domain (as *SEBJ* finds them) - for example ($OOP = 9AM$, $Databases = 10AM$)

would be added when the Computing agent's *SEBJ* discovers this solution to its local problem - note that the value of variable Web is obtained but not used for constraint checking [see point (v) below] since it has no **inter-agent constraints**; consequently, *DisPeL-1C* could solve a problem without knowing all of the local non-interchangeable solutions to the local problems that *SEBJ* instances will generate. This is somewhat similar to the open domain concept for open constraint satisfaction problems [26]; (iv) *DisPeL-1C* keeps track of the best solution (with fewest constraint violations) found so far; (v) *DisPeL-1C* only considers the **inter-agent constraints** and not the intra-agent constraints since the latter have already been checked by *SEBJ*; In our sample problem, this means that constraints such as $Databases \neq Web$ are ignored but constraints such as $OOP \neq Manag$ are now considered. *DisPeL-1C* may discover that the first solution for the Computing agent ($OOP = 9AM$, $Databases = 10AM$) extends with the second solution for the Business agent ($Manag = 10AM$) combined with the Art agent's first solution ($Sculpting = 9AM$) to form a global solution to the problem. If this is the case, *Multi-Hyb-Pen* terminates. Otherwise, *DisPeL-1C* terminates whenever *SEBJ* terminates.

Multi-Hyb-Pen combines a weight of 70% for the sum of *DisPeL-1C*'s incremental penalties on variable values (which is periodically reset) with a weight of 30% of the cumulative penalty count of all penalties imposed by *DisPeL-1C* on a variable (see below). This penalty information is used by *SynCBJ-CLP* (see below) to indicate the difficult areas of the problem.

SynCBJ-CLP

The SynCBJ algorithm [102] for complex local problems (*SynCBJ-CLP*) (see Algorithm 6) finds solutions to the inter-agent constraint problem. It uses **one complex variable per agent**, with each variable representing all the externally relevant variables of a complex local problem. The algorithm explores partial solutions generated by *SEBJ* such as ($OOP = 9AM$, $Databases = 10AM$) and ($Manag = 10AM$) to see if they extend to a global solution. Hence, *SynCBJ-CLP* only considers the **inter-agent constraints** (for example $OOP \neq Manag$) and ignores the intra-agent constraints, since these have already been checked by *SEBJ*. *SynCBJ-CLP* uses the following knowledge learnt by distributed local search: (i) difficult areas of the problem and; (ii) best 'solution' found so far. Those variables which are thought to represent difficult areas of the problem are ordered before "easier" variables. The variable values involved in the best 'solution' found by local search are tried first (value ordering). This knowledge sharing between a local and a systematic search algorithm is inspired by the *DisHyb* framework (see chapter 5).

Algorithm 6 procedure SynCBJ-CLP ($rankedDifficultVariables$, $bestValues$)

$ao \leftarrow$ list of agents sorted by max degree and $rankedDifficultVariables$
Prioritise best values ($bestValues_i$)
SynCBJ(ao)
if solution found by SynCBJ **then**
    return solution
else
    return "unsolvable problem"
end if

SynCBJ-CLP is efficient through: (i) its use of **complex variables**, aggregating all variables of the agent's complex local problem thereby having one complex variable per agent; (ii) only **inter-agent constraints** are given consideration (the same constraints considered by *DisPeL-1C*). Since SynCBJ is complete and variations introduced in *SynCBJ-CLP* only change the ordering of agents and first variable value, *SynCBJ-CLP* is completed and consequently when combined with the completeness of *SEBJ*, *Multi-Hyb-Pen* is complete.

Variations of *Multi-Hyb-Pen*

A series of experiments were conducted in order to measure the effectiveness of various agent orderings for *SynCBJ-CLP* using several heuristics based on the knowledge learnt by *DisPeL-1C*. For each heuristic, max degree is used with ties broken according to the specific heuristic rules. All orderings use the best 'solution' found by DisPeL as the first value in distributed systematic search. The following variations were considered:

1. ResetPen - Once all *SEBJ* searches have found all solutions, the *DisPeL-1C* search is stopped and starts a new *DisPeL-1C* search for a small number of cycles before switching to *SynCBJ-CLP*. The variable and value ordering in *SynCBJ-CLP* is

determined from the second *DisPeL-1C* search.

2. CumPen - The cumulative penalty count from *DisPeL-1C* is used for agent ordering.

3. DisPeLPen - This version uses *DisPeL-1C*'s current penalties for reordering variables rather than the cumulative penalty count.

4. RelCumPen (SL)/(SE) - RelCumPen (SL) gives higher relevance to penalties imposed later in the *DisPeL-1C* search. Specifically, the number of cycles which *DisPeL-1C* ran for is divided into three equal parts. The penalties imposed at the end of each of these parts is stored. For example, if *DisPeL-1C* ran for 60 cycles, the penalty count would be measured at 20, 40 and 60 cycles. The penalty count is reset at 20 and 40 cycles after it is measured. The cumulative penalty is then composed from 20% of the first penalty count, 30% of the second penalty count and 50% of the final penalty count. RelCumPen (SE) is the converse approach. Specifically, 50% of the first penalty count, 30% of the second penalty count and 20% of the final penalty count.

5. BothPens (50)/(CP)/(DP) - These versions modify the weightings of *DisPeL-1C*'s own penalties (DP) vs. the cumulative penalty count (CP) as follows: 50:50 (BothPens (50)), 30:70 (BothPens(CP)) and 70:30 (BothPens(DP)).

Results for the experiments on randomly generated DisCSPs with 80 variables, 8 domain values, 5 agents, 85% intra-agent constraints and 15% inter-agent constraints, 0.2 constraint density and 0.35 constraint tightness are shown in Table 6.3. The heuristics above were compared against a simple max-degree ordering for *SynCBJ-CLP*. The median and average values over 100 problems for number of messages and number of non-concurrent constraint checks (NCCCs) were measured.

Whilst starting a fresh version of *DisPeL-1C* for ordering (ResetPen) has some merit in reducing the number of non-concurrent constraint checks versus a simple max-degree ordering, it is considerably more costly than any of the other heuristics. This is caused by the longer period of *DisPeL-1C*'s execution. Indeed, in terms of messages, this approach is more costly than a simple max-degree ordering. All other variations improve on the simple

	Median		Average	
Heuristic	**Msgs**	**NCCCs**	**Msgs**	**NCCCs**
MaxDegree	255	118,996	376	166,647
ResetPen	285	115,832	407	139,982
CumPen	**139**	109,488	**172**	128,561
DisPeLPen	196	114,568	212	138,688
RelCumPen (SL)	158	111,247	217	131,894
RelCumPen (SE)	164	113,136	202	129,986
BothPens (50)	185	111,372	234	136,229
BothPens (DP)	**139**	**105,988**	175	**126,288**
BothPens (CP)	167	106,749	178	118,375

Table 6.3: Performance of different heuristics for Multi-Hyb-Pen.

max-degree ordering. The cumulative penalty (CumPen) performed better than *DisPeL-1C*'s own penalties (DisPeLPen), and consequently experiments were set up in order to determine whether this ordering could be improved by either weighting later penalties higher than earlier ones (RelCumPen(SL)) or the converse approach (RelCumPen(SE)). Imposing higher weights on penalties imposed later in *DisPeL-1C*'s search had a positive effect on the median but a negative effect on the average, with the converse approach having the opposite effect, although there is little difference in performance between both heuristics. A combination of the cumulative penalty and *DisPeL-1C*'s penalties was also explored. A combination of 70% of *DisPeL-1C*'s penalty and 30% of the cumulative penalty offered the best performance.

From the results, the two most efficient variations appear to be CumPen and BothPens (DP) in terms of best numbers of messages and constraint checks. Consequently, the data was normalised to determine whether the lower constraint checks made more difference than a small increase in average messages or vice versa. The BothPens (DP) variant was then the most efficient variation. Therefore, this is the heuristic used in *Multi-Hyb-Pen* for the experimental evaluation.

6.3.2 Multi-Hyb-DB

Multi-Hyb-DB is an implementation of *Multi-Hyb* which uses *SEBJ* (see section 6.3.1) as the centralised systematic solver, DisBO-wd (see below) as the distributed local search algorithm and *SynCBJ-CLP* (see section 6.3.1) as the distributed systematic search algorithm.

DisBO-wd

The Distributed Breakout Algorithm originally proposed in [45] was improved through the use of a weight decay mechanism in DisBO-wd [8]. In *Multi-Hyb-DB*, we make a number of modifications to DisBO-wd; (i) DisBO-wd's variables are complex so that one variable represents all **externally relevant** variables for a particular agent's complex local problem; (ii) Domain values are dynamically added as *SEBJ* finds them (in an identical way to *DisPeL-1C*); (iii) DisBO-wd maintains a record of the best solution found (with the fewest constraint violations); (iv) DisBO-wd only considers the **inter-agent constraints** so that constraint weights are only modified for those constraints between agents. If DisBO-wd discovers a solution to the global problem, *Multi-Hyb-DB* terminates. Otherwise, DisBO-wd terminates whenever all instances of *SEBJ* have terminated. *Multi-Hyb-DB* uses the highest constraint weight belonging to each complex variable (agent) from DisBO-wd to order the agents according to the difficult areas of the problem and passes this information to *SynCBJ-CLP*.

Variations of *Multi-Hyb-DB*

Experiments were also conducted in *Multi-Hyb-DB* to determine the best agent ordering heuristic for *SynCBJ-CLP* based on the knowledge learnt by DisBO-wd. For each heuristic, max degree is used with ties broken according to the specific heuristic rules. All orderings use the best 'solution' found by DisBO-wd as the first value in distributed systematic search. The following variations were considered:

1. Reset Weights - Once all *SEBJ* searches have found all solutions, this search stops the DisBO-wd search and starts a new DisBO-wd search for a small number of cycles before switching to *SynCBJ-CLP*. The variable and value ordering in *SynCBJ-CLP* is determined from the second DisBO-wd search.

2. Best Weights - This version uses the constraint weights at the time when the best solution (i.e. the set of values which minimised the number of constraint violations (excluding constraint weights)) occurred during the DisBO-wd run.

3. Last Weights - This version uses the constraint weight values upon termination of DisBO-wd.

Results for the experiments on randomly generated DisCSPs with 80 variables, 8 domain values, 5 agents, 80% intra-agent constraints and 20% inter-agent constraints, 0.2 constraint density and 0.35 constraint tightness are shown in Table 6.4. The heuristics above were compared against a simple max-degree ordering for *SynCBJ-CLP*. The median and average values over 100 problems for number of messages and number of non-concurrent constraint checks (NCCCs) were measured.

	Median		**Average**	
Heuristic	**Msgs**	**NCCCs**	**Msgs**	**NCCCs**
MaxDegree	185	271,107	369	483,614
Reset Weights	254	271,710	441	516,024
Best Weights	**158**	268,336	**350**	**472,720**
Last Weights	187	**262,508**	353	472,823

Table 6.4: Performance of different heuristics for Multi-Hyb-DB.

The reset weights heuristic is not competitive as it is outperformed by a simple max degree ordering. Whilst the last weights heuristic minimises the median number of NCCCs, the best weights heuristic outperforms the last weights heuristic in median number of messages as well as average number of messages and average number of NCCCs. We normalised the values for median messages and median NCCCs for both the best weights and last weights heuristic and found that the best weights' performance improvement in terms of messages was more significant than last weights' improvement in NCCCs. Consequently, the best weight heuristic is used in *Multi-Hyb-DB* in this chapter.

6.4 Experimental Evaluation

We evaluated the two implementations of our *Multi-Hyb* framework. *Multi-Hyb-Pen* uses *DisPeL-1C* as the local search solver and reoders complex variables in *SynCBJ-CLP* by max degree then penalties whereas *Multi-Hyb-DB* uses DisBO-wd as the local search solver and reorders complex variables in *SynCBJ-CLP* through max degree then constraint weights. We compared both implementations of *Multi-Hyb* with both systematic and local search algorithms designed for DisCSPs with complex local problems. For systematic search, *Multi-Hyb-Pen* and *Multi-Hyb-DB* were compared against Multi-ABT and Multi-AWCS. For local search, *Multi-Hyb-Pen* and *Multi-Hyb-DB* were compared against Multi-DisPeL and DisBO-wd. Note that Burke's work [13] concentrated on efficient algorithms for handling the complex local problems and as such does not present an overall algorithm for comparison. ABT-cf [23] forces the local solver to find all solutions to the complex local problem before the distributed search begins which then tries different combinations of these local solutions to find a global solution to the problem. As a result, their work is only evaluated on small and easy problems and so a comparison with their work is not possible [1]. We also did not consider DCDCOP [50] for evaluation as there is insufficient pseudo code currently available (since the algorithm was only presented in September 2009) to implement the algorithm. It also has not been evaluated against DisCSP algorithms as it has been designed as a Distributed Constraint Optimisation algorithm.

Experiments were run on **distributed randomly generated problems**, **distributed graph colouring problems**, **distributed meeting scheduling problems** and **distributed sensor network problems**. Our implementation of Multi-ABT was verified against the distributed graph colouring experiments in [46], our Multi-AWCS implementation was verified against the distributed graph colouring experiments from [96] and our Multi-DisPeL and DisBO-wd implementations were obtained from their authors. The results obtained were at least as good as those reported by the authors. We measured: (i) the number of Non-Concurrent Constraint Checks (NCCCs) performed and; (ii) the number of messages sent. Note that the number of messages required for termination detection is not counted for any of the algorithms as reported by other researchers [96]. Although CPU time is not an established measure for DisCSPs [56], we also measured it and the results obtained were consistent with the other measures used.

Extensive empirical evaluations were carried out while varying the number of variables (60-200), the domain sizes (5-10), the constraint tightness (0.35-0.5), the constraint densities (0.15-0.2) and the number of agents (5-25). Since the *Multi-Hyb* approach was

[1] Through a personal communication with the authors of ABT-cf, we obtained an implementation of ABT-cf. This version had not been tested by the authors above 5 variables per agent and our tests with the supplied implementation have shown that ABT-cf runs out of memory with more variables per agent.

developed for naturally distributed DisCSPs with complex local problems (with a higher proportion of intra-agent constraints than inter-agent constraints), the problems considered (except for distributed sensor network problems) contained between 70% and 90% intra-agent constraints with the remainder being inter-agent constraints. For each problem type (proportion of intra-agent and inter-agent constraints), 100 different problems were solved and average and median results calculated.

6.4.1 Solvable Problems

For solvable problems, the number of problems solved was also measured for Multi-DisPeL and DisBO-wd because they are incomplete algorithms. However, these generally solved the vast majority of problems. For Multi-DisPeL and DisBO-wd, a cut-off of $100n$ cycles (where n is the number of variables) and $200n$ cycles respectively (since 2 DisBO-wd cycles of *improve* and *ok*? equal one Multi-DisPeL cycle) were used. In the few cases where not all problems were solved (indicated by * in the results), the effort wasted (number of NCCCs and number of messages) was not included in the results.

Randomly Generated Problems

Median results for distributed randomly generated problems appear in Table 6.5. The number of variables ranged between 60 and 175. The number of agents was 5, the domain size was 8, constraint density was 0.2 and the constraint tightness was 0.35. The percentage of intra-agent constraints varied between 70% and 90% with the remainder being inter-agent constraints. The results of the best performing algorithm are shown in bold.

For large randomly generated problems (between 80 and 125 variables), *Multi-Hyb-DB* gives the best results. For very large problems (150 variables and above), *Multi-Hyb-Pen* gives the best results. There are a few occasions where Multi-DisPeL uses slightly less messages but the difference is very small when compared with the large difference in the number of NCCCs. For smaller problems, Multi-AWCS is best for NCCCs but uses substantially more messages. There is only one occasion (60 variables, 70:30 intra-agent to inter-agent constraints) where neither *Multi-Hyb-Pen* or *Multi-Hyb-DB* gives the best

		Median number of messages					
Num Vars	**% intra:inter constraints**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
60	90:10	399	**323**	842	4,834	536	1,150*
60	80:20	197	**158**	1,692	5,287	422	1,165
60	70:30	818	833	6,832	4,475	**496**	985
70	80:20	159	**96**	731	3,672	208	435
70	70:30	**112**	175	1,141	3,907	194	420
80	80:20	143	**60**	440	3,991	104	335
80	70:30	89	**60**	500	6,076	108	295
90	80:20	94	**60**	336	4,242	66	275
90	70:30	81	**60**	298	6,193	80	265
100	80:20	**56**	60	248	5,922	**56**	235
100	70:30	78	**60**	276	7,235	**60**	225
125	80:20	**20**	60	197	6,297	40	225
125	70:30	60	60	152	9,218	**40**	205
150	80:20	**20**	60	152	6,803	28	215
150	70:30	**30**	46	128	14,554	32	195
175	80:20	**20**	45	134	10,707	24	210
175	70:30	**20**	45	118	15,126	24	190
		Median number of NCCCs					
Num Vars	**% intra:inter constraints**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
60	90:10	**163,585**	170,093	314,067	165,118	118,735	469,162*
60	80:20	277,408	268,336	420,384	**194,432**	949,616	440,862
60	70:30	2,761,171	2,626,087	286,821	**182,936**	1,148,704	353,862
70	80:20	151,678	133,577	284,713	**124,238**	745,608	252,678
70	70:30	291,421	288,457	524,487	**135,090**	673,099	244,962
80	80:20	118,874	**114,283**	207,389	149,599	588,111	283,827
80	70:30	169,884	**153,848**	356,405	265,274	606,084	262,707
90	80:20	117,668	**105,869**	278,057	177,570	611,811	308,444
90	70:30	140,181	**130,355**	224,968	291,656	638,729	299,228
100	80:20	107,836	**101,792**	214,806	285,431	690,977	339,423
100	70:30	132,031	**125,176**	265,460	385,969	690,455	324,668
125	80:20	106,435	**104,718**	185,646	357,508	952,787	509,090
125	70:30	125,553	**121,680**	360,376	600,688	936,775	485,739
150	80:20	**100,020**	102,519	235,880	441,287	1362161	728427
150	70:30	**120,105**	128,039	268,777	1,302,570	1,281,866	682,116
175	80:20	**98,875**	103,143	155,900	885,339	1,926,771	976,712
175	70:30	**110,325**	124,838	235,168	1,453,996	1,831,216	908,710

Table 6.5: Results for solvable random problems.

results for either messages or NCCCs - this is because 60-variable problems are fairly small. *Multi-Hyb-Pen* in particular seems to be able to make best use of the information that agents have in the *Multi-Hyb* approach for the higher number of variables per agent problems (125 variables and above).

Graph Colouring Problems

Median results for distributed graph colouring problems are shown in Table 6.6. 150 and 200 nodes were used with 15 to 25 agents, 3 colours and a degree between 4.9 and 5.1. The percentage of intra-agent constraints varied between 70% and 90% with the remainder being inter-agent constraints.

For graph colouring problems, *Multi-Hyb-Pen* clearly gives the best results in terms of number of messages. Multi-ABT is often the better performing algorithm for NCCCs although *Multi-Hyb-Pen* is often better for large number of agents (i.e. smaller complex local problems). This is owing to the cost of searching for all local solutions until a global solution is found. *Multi-Hyb-DB* is not competitive when compared with *Multi-Hyb-Pen* suggesting that penalties provides the better knowledge in this case.

Meeting Scheduling Problems

Median results for solvable meeting scheduling problems (as described in section 2.3.3) are presented in Table 6.7. Our problems had 50-80 meetings, 5 departments (agents), a timeframe of 6 or 7 time units and a constraint density of 0.18. The percentage of intra-agent constraints varied between 70% and 90% with the remainder being inter-agent constraints. Two departments with common meetings have a random distance between 1 and 3 time units.

For scheduling problems, *Multi-Hyb-Pen* performed best for the majority of problems in terms of number of messages. *Multi-Hyb-DB* and Multi-DisPeL were also optimal for different problem parameters for number of messages. For NCCCs, Multi-ABT and Multi-AWCS were the most consistent algorithms. However, *Multi-Hyb-Pen* did outperform these algorithms for some problem parameters. Multi-DisPeL and DisBO-wd gave poorer

				Median number of messages					
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
150	15	4.9	90:10	**40**	155	490	1,281	595	855
150	15	5.1	90:10	**35**	163	608	1,437	714	840*
150	15	4.9	80:20	**21**	134	326	1,102	588	765*
150	15	5.1	80:20	**23**	143	350	1,248	616	900*
150	15	4.9	70:30	**31**	180	591	1,588	714	780*
150	15	5.1	70:30	**31**	185	629	1,909	735	900*
150	25	4.9	90:10	**35**	177	373	1,508	1,176	1,175*
150	25	5.1	90:10	**29**	179	399	1,534	1,176	1,200*
150	25	4.9	80:20	**53**	317	2,696	2,079	1,392	1,300*
150	25	5.1	80:20	**37**	245	1,053	2,423	1,368	1,325*
150	25	4.9	70:30	**42**	261	1,403	2,879	1,788	1,500*
150	25	5.1	70:30	**51**	338	3,642	3,362	1,680	1,275*
200	20	4.9	90:10	**62**	212	698	2,146	1,197	1,420*
200	20	5.1	90:10	**73**	223	938	2,328	1,216	1,300*
200	20	4.9	80:20	**31**	188	528	1,732	1,064	1,220*
200	20	5.1	80:20	**34**	196	544	1,851	1,140	1,340*
200	20	4.9	70:30	**59**	266	1,050	2,465	1,225	1,200*
200	20	5.1	70:30	**77**	289	1,278	2,668	1,282	1,440*
200	25	4.9	90:10	**51**	233	657	2,350	1,716	1,425*
200	25	5.1	90:10	**45**	232	869	2,092	1,800	1,575*
200	25	4.9	80:20	**57**	252	911	2,396	1,680	1,450
200	25	5.1	80:20	**44**	250	1,068	2,446	1,692	1,425*
200	25	4.9	70:30	**56**	309	2,048	3,148	1,848	1,625*
200	25	5.1	70:30	**62**	339	2,746	3,259	1,992*	1,825*
				Median number of NCCCs					
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
150	15	4.9	90:10	3,579	3,735	**1,266**	3,172	46,215	66,583
150	15	5.1	90:10	3,689	3,837	**1,589**	3,435	57,967	63,567*
150	15	4.9	80:20	1,314	1,611	**1,165**	3,123	49,008	63,739*
150	15	5.1	80:20	1,279	1,653	**1,278**	3,310	51,564	73,127*
150	15	4.9	70:30	1,882	2,659	**1,501**	3,535	53,692	57,495*
150	15	5.1	70:30	1,783	2,507	**1,535**	4,058	59,712	68,942*
150	25	4.9	90:10	**675**	775	689	1,454	30,961	53,242*
150	25	5.1	90:10	**633**	757	724	1,417	33,134	53,127*
150	25	4.9	80:20	**729**	1,223	1,532	1,651	35,018	53,604*
150	25	5.1	80:20	**549**	800	1,017	1,974	37,113	55,657*
150	25	4.9	70:30	**726**	1,253	1,802	2,265	43,369	46,600*
150	25	5.1	70:30	**534**	801	1,218	2,087	43,344	57,361*
200	20	4.9	90:10	4,195	4,561	**1,434**	3,836	71,275	104,597*
200	20	5.1	90:10	4,403	4,646	**1,716**	4,185	70,314	105,865*
200	20	4.9	80:20	1,439	1,900	**1,286**	3,637	65,360	99,080*
200	20	5.1	80:20	1,467	1,925	**1,273**	3,623	72,354	106,079*
200	20	4.9	70:30	2,369	3,403	**1,604**	4,180	73,351*	89,740*
200	20	5.1	70:30	2,348	3,484	**1,872**	4,405	77,346	107,339*
200	25	4.9	90:10	1,843	2,154	**1,014**	2,723	61,481	87,216*
200	25	5.1	90:10	1,703	2,046	**1,214**	2,499	68,940	99,001*
200	25	4.9	80:20	**972**	1,261	1,267	2,669	61,118	89,533
200	25	5.1	80:20	**878**	1,225	1,424	2,903	66,544	89,009*
200	25	4.9	70:30	**1,272**	2,089	1,727	2,975	63,778	86,824*
200	25	5.1	70:30	**1,372**	2,156	2,029	3,121	71,947*	101,275*

Table 6.6: Results for solvable graph colouring problems.

			Median number of messages					
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
50	7	90:10	**20**	54	81	340	68	295*
50	7	80:20	**40**	60	112	381	60*	405*
50	7	70:30	139	**75**	204	415	96	335*
50	6	90:10	**10**	45	64	269	52	155*
50	6	80:20	**20**	60	96	321	64	165*
50	6	70:30	184	102	161	362	**66**	215*
60	7	90:10	**20**	60	86	359	64	245*
60	7	80:20	80	**60**	136	396	76	275*
60	7	70:30	412	173	341	500	**72**	295*
60	6	90:10	**10**	45	78	288	32	145*
60	6	80:20	**10**	45	106	327	44	175*
60	6	70:30	**42**	60	149	409	56	225*
70	7	90:10	**20**	60	103	380	44	235*
70	7	80:20	**20**	60	128	428	56	255
70	7	70:30	228	90	205	514	**64**	315
70	6	90:10	**20**	45	91	274	40	165*
70	6	80:20	**20**	60	116	352	40	195
70	6	70:30	**40**	60	132	415	50	245
80	7	90:10	**20**	60	115	404	48	235
80	7	80:20	**20**	60	128	473	48	245
80	7	70:30	151	74	185	547	**60**	305
80	6	90:10	**20**	45	98	284	32	185
80	6	80:20	**20**	60	118	379	40	205
80	6	70:30	**20**	60	124	443	44	245
			Median number of NCCCs					
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
50	7	90:10	7,162	7,369	**6,988**	7,309	112,308	110,290*
50	7	80:20	10,852	13,139	9,488	**8,214**	130,639	138,306*
50	7	70:30	20,684	25,451	13,774	**8,605**	120,664	126,017*
50	6	90:10	**2,933**	3,503	3,793	5,534	73,805	57,262*
50	6	80:20	4,803	5,259	**4,411**	5,974	79,868	84,785*
50	6	70:30	7,451	9,632	**5,238**	6,382	74,451	71,166*
60	7	90:10	10,777	12,076	10,901	**10,613**	160,589	158,103*
60	7	80:20	16,251	16,367	11,413	**10,821**	163,578	183,771*
60	7	70:30	37,138	36,649	15,464	**12,513**	153,894	158,777*
60	6	90:10	**5,095**	5,700	5,490	7,894	89,497	91,349*
60	6	80:20	6,163	6,346	**5,981**	8,249	99,156	107,302*
60	6	70:30	11,334	11,654	**6,766**	9,628	100,219	201,621*
70	7	90:10	15,377	17,757	**13,044**	13,739	198,303	203,387*
70	7	80:20	20,174	21,380	**12,956**	14,696	199,104	240,856
70	7	70:30	38,453	45,164	**15,624**	16,365	214,783	241,370
70	6	90:10	**6,586**	7,573	6,906	10,373	131,723	136,478*
70	6	80:20	9,523	9,632	**6,880**	11,512	129,821	154,904
70	6	70:30	14,375	12,949	**7,354**	12,123	148,914	163,560
80	7	90:10	17,434	17,651	**14,685**	18,715	270,668	280,138
80	7	80:20	27,460	26,809	**13,708**	19,959	263,191	303,366
80	7	70:30	50,844	50,219	**17,888**	21,276	271,813	312,564
80	6	90:10	8,863	8,461	**7,432**	14,264	177,645	187,330
80	6	80:20	10,967	11,202	**7,687**	14,796	185,343	223,587
80	6	70:30	14,073	15,645	**8,512**	16,331	186,795	224,357

Table 6.7: Results for solvable meeting scheduling problems.

results.

Sensor Networks

Finally, the algorithms were evaluated on Grid-based SensorDCSP (see section 2.3.4). These problems are **not naturally distributed** since they have a large number of inter-agent constraints combined with relatively simple local problems for each agent. Consequently, the ratio is now 85% inter-agent constraints and 15% intra-agent constraints. The problems used had between 5 and 8 targets, between 25 and 64 sensors (grids of 5, 6, 7, 8), k-visibility of 2, k-compatibility of 1, probability of visibility of 0.9 and probability of compatibility of 0.6. Median results are shown in Table 6.8. In some cases, Multi-DisPeL was optimal in terms of number of messages but did not solve all problems. In these cases, the next optimal algorithm which solved all problems is shown in bold as well as Multi-DisPeL.

Performance varied between algorithms for number of messages with *Multi-Hyb-DB* and Multi-ABT offering the most consistent performances. For NCCCs, either *Multi-Hyb-Pen* and *Multi-Hyb-DB* are optimal for the majority of the problem combinations with Multi-AWCS also offering good performance. This is interesting since the *Multi-Hyb* approach was not originally designed for these types of problems.

6.4.2 Unsolvable Problems

Multi-Hyb-Pen and *Multi-Hyb-DB* were only compared against Multi-ABT and Multi-AWCS on unsolvable problems since Multi-DisPeL and DisBO-wd cannot detect unsolvability. We distinguish between two categories of unsolvable problems: (i) those where at least one complex local problem is unsolvable and; (ii) those where all complex local problems are solvable, but no overall solution exists.

Randomly Generated Problems: Median results for unsolvable randomly generated problems using 5 agents, a domain size of 8 and a constraint tightness of 0.35 are presented in table 6.9 for problems which have one or more complex local problems that are unsolvable. It should be noted in this case that the *SEBJ* part of *Multi-Hyb-Pen* and

		Median n. Messages					
Num Targets	**Num Sensors**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
5	25	69	**63**	204	299	80*	575*
5	36	50	**49**	52	185	40*	285*
5	49	25	42	**24**	94	40*	160*
5	64	**14**	34	19	101	28*	120*
6	25	1,649	**765**	1,390	1,166	417	1,938*
6	36	1,383	242	**145**	333	**105***	846
6	49	338	116	**60**	185	**60***	414
6	64	510	310	**31**	127	50*	306*
7	25	3,814	**2,300**	8,786	3,492	**1,161***	4,907*
7	36	3,868	**1,051**	1,164	955	225*	1960*
7	49	1,092	210	**128**	330	**126***	609*
7	64	482	196	**55**	216	93*	658*
8	25	16,471	**3,644**	108,882	16,155	3,979*	25,608*
8	36	5,522	3,842	5,087	1,693	**759***	**3,840***
8	49	2,753	1,100	**328**	693	**203***	1296*
8	64	1,175	411	**126**	473	143*	768*
		Median n. NCCCs					
Num Targets	**Num Sensors**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
5	25	**4,072**	6,599	8,859	5,959	40,031*	66,968*
5	36	**2,936**	5,353	4,329	3,888	25,707*	31,359*
5	49	2,708	3,431	2,755	**2,314**	18,280*	19,366*
5	64	2,541	2,759	2,294	**1,856**	14,432*	15,397*
6	25	**13,164**	49,144	27,603	19,024	194,721*	248,682*
6	36	7,819	**2,306**	9,159	5,645	47,195*	98,318*
6	49	5,706	**2,112**	4,544	3,474	29,704*	48,459*
6	64	18,774	**2,497**	3,230	2,588	24,140*	31,515*
7	25	120,789	133,882	114,529	**44,926**	453,891*	623,861*
7	36	**8,622**	23,240	27,975	12,062	112,370*	267,908*
7	49	21,124	**2,288**	7,149	4,886	53,062*	83,965*
7	64	16,961	**2,229**	4,420	3,596	37,510*	68,098*
8	25	1,395,619	595,777	970,639	**190,699**	1,335,327*	3,667,100*
8	36	21,999	133,809	75,134	**18,978**	281,020*	545,384*
8	49	**7,420**	36,938	11,884	8,217	81,203*	161,266*
8	64	19,316	**2,417**	7,726	6,110	56,022*	94,678*

Table 6.8: Results for solvable Grid-based Sensor Network problems.

Multi-Hyb-DB will detect unsolvability in phase 1 so the distributed local search will not run and therefore these two algorithms will perform identically on these types of problems. We found that *Multi-Hyb-Pen* and *Multi-Hyb-DB* substantially outperformed Multi-ABT and Multi-AWCS on these problems in both number of messages and number of NCCCs. There is only one case (90 variables, 80:20 intra:inter-agent constraints) where Multi-ABT outperforms *Multi-Hyb-Pen* and *Multi-Hyb-DB* slightly on messages but *Multi-Hyb-Pen* and *Multi-Hyb-DB* substantially outperform Multi-ABT on NCCCs.

We also conducted experiments with identical parameters for problems that had solutions to all complex local problems but no global solution, the results of which are shown in Table 6.10. *Multi-Hyb-Pen* performed best for messages whilst *Multi-Hyb-DB* and Multi-ABT were often the better algorithms for NCCCs.

Graph Colouring Problems: Median results for unsolvable 3-colour distributed graph colouring problems with 150 to 200 nodes, 15 to 25 agents and 4.9 to 5.1 degree where one or more agents had no solutions to their complex local problem are presented in table 6.11. In these cases, *Multi-Hyb-Pen* and *Multi-Hyb-DB* will perform identically since *SEBJ* will detect unsolvability in phase 1 of the Multi-Hyb approach so distributed local search will not run. We found that *Multi-Hyb-Pen* and *Multi-Hyb-DB* substantially outperformed Multi-ABT and Multi-AWCS on these problems for messages. Multi-ABT was occasionally better for NCCCs but the large difference in messages meant that *Multi-Hyb-Pen* and *Multi-Hyb-DB* were better overall. Median results in table 6.4.2 are for problems where all agents had solutions to their complex local problem but there was no global solution to the problem. *Multi-Hyb-Pen* performed best for number of messages whilst *Multi-Hyb-Pen* and Multi-ABT performed best for NCCCs depending on the problem parameters. However, for most but not all of the cases where Multi-ABT outperforms *Multi-Hyb-Pen* on NCCCs, the difference between the algorithms in terms of messages is larger. Consequently, in general, *Multi-Hyb-Pen* is the optimal algorithm.

Meeting Scheduling Problems: Unsolvable meeting scheduling problems with 50-80 meetings, 5 departments (agents), a timeframe of 6 or 7 time units and a constraint density of 0.18 were conducted. The percentage of intra-agent constraints varied between

			Median number of messages			
Num Vars	**% constraint density**	**% intra:inter constraints**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
60	0.2	90:10	**14**	**14**	647	38,169
70	0.2	80:20	**12**	**12**	420	46,792
70	0.2	70:30	**16**	**16**	682	48,959
80	0.2	80:20	**12**	**12**	285	53,343
80	0.2	70:30	**12**	**12**	353	56,070
90	0.18	80:20	12	12	**10**	58,800
90	0.18	70:30	**12**	**12**	292	62,809
100	0.16	80:20	**10**	**10**	**10**	64,706
100	0.16	70:30	**12**	**12**	371	69,132
125	0.14	80:20	**10**	**10**	**10**	80,695
125	0.14	70:30	**10**	**10**	**10**	86,454
150	0.12	80:20	**10**	**10**	**10**	95,779
150	0.12	70:30	**10**	**10**	**10**	102,960
175	0.1	80:20	**10**	**10**	**10**	111,218
175	0.1	70:30	**10**	**10**	**10**	119,188
			Median number of NCCCs			
Num Vars	**% constraint density**	**% intra:inter constraints**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
60	0.2	90:10	**52,826**	**52,826**	116,728	10,082,412
70	0.2	80:20	**42,530**	**42,530**	131,095	10,388,804
70	0.2	70:30	**52,179**	**52,179**	162,757	11,137,456
80	0.2	80:20	**43,799**	**43,779**	145,124	12,703,763
80	0.2	70:30	**51,542**	**51,542**	176,548	14,467,021
90	0.18	80:20	**45,684**	**45,684**	108,806	14,363,762
90	0.18	70:30	**61,117**	**61,117**	219,677	17,521,470
100	0.16	80:20	**54,195**	**54,195**	116,340	16,992,283
100	0.16	70:30	**83,499**	**83,499**	290,934	20,802,015
125	0.14	80:20	**67,445**	**67,445**	139,844	25,087,165
125	0.14	70:30	**104,296**	**104,296**	212,568	31,483,422
150	0.12	80:20	**117,291**	**117,291**	179,070	34,293,903
150	0.12	70:30	**181,334**	**181,334**	305,890	43,698,629
175	0.1	80:20	**227,126**	**227,126**	272,432	45,266,830
175	0.1	70:30	**365,401**	**365,401**	459,317	56,044,863

Table 6.9: Median results for unsolvable random problems with one or more agents having no solution to their local problem.

			Median number of messages			
Num Vars	% constraint density	% intra:inter constraints	Multi -Hyb -Pen	Multi -Hyb -DB	Multi -ABT	Multi -AWCS
60	0.2	80:20	**177**	194	762	33,930
60	0.2	70:30	**249**	319	3,950	41,712
70	0.18	70:30	**114**	166	1,266	48,433
80	0.16	70:30	**106**	129	1,242	55,324
90	0.14	70:30	**158**	262	1,968	61,541
100	0.13	70:30	**129**	157	840	68,524
			Median number of NCCCs			
Num Vars	**% constraint density**	**% intra:inter constraints**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
60	0.2	80:20	62,205	**59,641**	127,460	7,620,027
60	0.2	70:30	251,012	252,212	**226,011**	7,996,729
70	0.18	70:30	**136,748**	**136,748**	192,851	10,569,556
80	0.16	70:30	**174,461**	**174,461**	230,568	13,527,324
90	0.14	70:30	374,569	372,796	**333,709**	16,092,489
100	0.13	70:30	362,227	354,277	**347,370**	19,929,678

Table 6.10: Median results for unsolvable random problems with all agents having solutions to their local problem but no global solution.

				Median number of messages			
Num Nodes	Num Agents	Deg	intra: inter	Multi -Hyb -Pen	Multi -Hyb -DB	Multi -ABT	Multi -AWCS
150	15	4.9	80:20	**42**	**42**	860	6,307
150	15	5.1	80:20	**42**	**42**	947	6,456
150	15	4.9	70:30	**50**	**50**	2,911	9,356
150	15	5.1	70:30	**48**	**48**	1,899	9,474
150	25	4.9	70:30	**72**	**72**	1,576	13,728
150	25	5.1	70:30	**68**	**68**	1,630	14,031
200	20	4.9	80:20	**57**	**57**	1,277	9,163
200	20	5.1	80:20	**58**	**58**	1,497	9,195
200	20	4.9	70:30	**66**	**66**	2,296	14,107
200	20	5.1	70:30	**64**	**64**	1,956	14,680
200	25	4.9	80:20	**68**	**68**	1,398	10,195
200	25	5.1	80:20	**66**	**66**	1,234	10,321
200	25	4.9	70:30	**79**	**79**	1,816	16,277
200	25	5.1	70:30	**76**	**76**	1,883	17,021
				Median number of NCCCs			
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
150	15	4.9	80:20	1,525	1,525	**1,202**	11,590
150	15	5.1	80:20	1,421	1,421	**1,286**	11,924
150	15	4.9	70:30	**2,332**	**2,332**	2,395	15,259
150	15	5.1	70:30	2,114	2,114	**1,797**	15,255
150	25	4.9	70:30	**296**	**296**	767	11,580
150	25	5.1	70:30	**294**	**294**	758	11,725
200	20	4.9	80:20	1,415	1,415	**1,304**	11,910
200	20	5.1	80:20	1,717	1,717	**1,321**	11,812
200	20	4.9	70:30	2,512	2,512	**1,727**	15,300
200	20	5.1	70:30	2,253	2,253	**1,656**	15,854
200	25	4.9	80:20	**673**	**673**	900	9,807
200	25	5.1	80:20	**644**	**644**	845	9,693
200	25	4.9	70:30	**895**	**895**	1,053	12,411
200	25	5.1	70:30	**875**	**875**	1,095	12,990

Table 6.11: Median results for unsolvable graph colouring problems with one or more agents having no solution to their local problem.

				Median number of messages			
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
150	15	4.9	80:20	**144**	250	1,417	6,620
150	15	5.1	80:20	**187**	311	1,823	6,627
150	15	4.9	70:30	**388**	518	4,019	9,816
150	15	5.1	70:30	**208**	364	3,590	9,942
150	25	4.9	80:20	**48**	261	1,405	13,174
150	25	5.1	80:20	**27**	246	1,205	14,173
150	25	4.9	70:30	**61**	328	3,134	19,866
150	25	5.1	70:30	**48**	333	2,863	22,954
200	20	4.9	80:20	**266**	414	1,464	8,480
200	20	5.1	80:20	**176**	342	1,424	9,015
200	20	4.9	70:30	**1,324**	1,528	4,818	13,206
200	20	5.1	70:30	**744**	952	3,402	13,058
200	25	4.9	80:20	**186**	376	1,429	11,049
200	25	5.1	80:20	**116**	313	1,166	11,577
200	25	4.9	70:30	**354**	627	3,097	15,778
200	25	5.1	70:30	**204**	498	2,495	17,386
				Median number of NCCCs			
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
150	15	4.9	80:20	**2,184**	2,275	2,514	23,646
150	15	5.1	80:20	**2,166**	2,355	2,981	24,823
150	15	4.9	70:30	7,566	7,566	**4,571**	30,175
150	15	5.1	70:30	4,250	4,250	**4,150**	30,428
150	25	4.9	80:20	**439**	830	1,029	20,399
150	25	5.1	80:20	**394**	814	883	21,351
150	25	4.9	70:30	**558**	1,339	1,522	26,605
150	25	5.1	70:30	**514**	1,155	1,398	30,230
200	20	4.9	80:20	3,263	3,263	**2,333**	22,227
200	20	5.1	80:20	2,375	2,666	**2,132**	23,200
200	20	4.9	70:30	10,130	10,130	**4,201**	28,327
200	20	5.1	70:30	7,502	7,502	**3,195**	27,356
200	25	4.9	80:20	1,607	1,718	**1,503**	20,176
200	25	5.1	80:20	**1,126**	1,399	1,416	20,676
200	25	4.9	70:30	3,528	3,532	**2,104**	25,026
200	25	5.1	70:30	1,968	2,736	**1,895**	25,263

Table 6.12: Median results for unsolvable graph colouring problems with all agents having at least one solution to their local problem but no global solution.

70% and 80%. Two departments with common meetings have a distance of between 1 and 3 time units. Problems where one or more agents had no solution to their complex local problem are presented in table 6.13. For these problems, the *SEBJ* algorithm detected unsolvability and so both implementations of *Multi-Hyb* performed identically. These implementations substantially outperformed Multi-ABT and Multi-AWCS on both messages and NCCCs. Problems where all agents had solutions to their complex local problem but there was no global solution are presented in table 6.14.

			Median number of messages			
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
50	7	80:20	**13**	**13**	182	1,730
50	7	70:30	**14**	**14**	331	2,308
50	6	80:20	**12**	**12**	86	1,138
50	6	70:30	**14**	**14**	176	1,446
60	7	80:20	**12**	**12**	124	1,687
60	7	70:30	**14**	**14**	240	2,390
60	6	80:20	**11**	**11**	117	1,145
60	6	70:30	**12**	**12**	171	1,511
70	7	80:20	**10**	**10**	152	1,721
70	7	70:30	**12**	**12**	185	2,232
70	6	80:20	**12**	**12**	110	1,139
70	6	70:30	**12**	**12**	132	1,495
80	7	80:20	**10**	**10**	115	1,659
80	7	70:30	**10**	**10**	167	2,285
80	6	80:20	**10**	**10**	97	5,724
80	6	70:30	**12**	**12**	239	1,401
			Median number of NCCCs			
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
50	7	80:20	**3,051**	**3,051**	10,128	26,687
50	7	70:30	**3,174**	**3,174**	11,474	32,575
50	6	80:20	**2,315**	**2,315**	3,929	15,309
50	6	70:30	**1,916**	**1,916**	5,270	18,386
60	7	80:20	**3,055**	**3,055**	12,044	31,148
60	7	70:30	**3,476**	**3,476**	11,779	41,168
60	6	80:20	**2,211**	**2,211**	5,021	17,618
60	6	70:30	**1,980**	**1,980**	5,638	21,167
70	7	80:20	**3,395**	**3,395**	15,330	34,728
70	7	70:30	**4,343**	**4,343**	14,405	41,546
70	6	80:20	**2,275**	**2,275**	6,152	20,240
70	6	70:30	**2,576**	**2,576**	6,598	24,230
80	7	80:20	**4,637**	**4,637**	14,145	38,468
80	7	70:30	**3,941**	**3,941**	16,856	49,571
80	6	80:20	**2,210**	**2,210**	5,724	20,048
80	6	70:30	**2,890**	**2,890**	1,674	1,674

Table 6.13: Median results for unsolvable meeting scheduling problems with one or more agents having no solution to their local problem.

The results show that *Multi-Hyb-Pen* and *Multi-Hyb-DB* were the best performing algorithms where one or more agents had no solution to their complex local problem.

			Median number of messages			
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
50	7	80:20	344	**150**	197	4,926
50	7	70:30	624	517	**507**	5,177
50	6	80:20	222	**91**	107	3,502
50	6	70:30	204	**119**	151	4,226
60	7	80:20	320	**125**	132	4,991
60	7	70:30	284	**210**	306	5,106
60	6	80:20	**16**	45	62	3,488
60	6	70:30	190	**60**	85	4,158
70	7	80:20	248	89	**62**	4,950
70	7	70:30	242	**91**	115	5,099
70	6	80:20	146	**45**	61	3,525
70	6	70:30	94	**45**	62	4,159
80	7	80:20	196	83	**61**	4,939
80	7	70:30	162	**71**	112	5,077
80	6	80:20	118	**43**	58	3,587
80	6	70:30	86	**45**	58	4,207
			Median number of NCCCs			
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
50	7	80:20	14,345	18,004	**9,965**	76,384
50	7	70:30	33,084	38,739	**11,309**	81,379
50	6	80:20	5,318	7,223	**4,055**	47,704
50	6	70:30	9,480	10,630	**4,930**	56,673
60	7	80:20	17,819	19,668	**10,808**	92,333
60	7	70:30	33,445	37,229	**13,464**	96,062
60	6	80:20	5,860	6,891	**4,219**	57,171
60	6	70:30	7,599	9,114	**5,152**	63,441
70	7	80:20	17,692	20,279	**9,919**	102,431
70	7	70:30	27,350	29,213	**11,554**	109,482
70	6	80:20	7,089	8,841	**4,736**	66,217
70	6	70:30	8,791	9,461	**6,222**	75,706
80	7	80:20	23,671	26,352	**10,977**	118,078
80	7	70:30	33,516	36,516	**16,282**	124,501
80	6	80:20	8,602	9,664	**5,703**	75,665
80	6	70:30	10,999	12,072	**6,216**	84,881

Table 6.14: Median results for unsolvable meeting scheduling problems with all agents having at least one solution to their local problem but no global solution.

When all agents had solutions to their complex local problem but there was no solution to the global problem, *Multi-Hyb-Pen* and *Multi-Hyb-DB* were able to reduce the number of messages in most cases by ensuring that all agents had solutions before beginning the message exchange. However, the cost of having to search for all local solutions in order to prove global unsolvability, meant that Multi-ABT was optimal for NCCCs with *Multi-Hyb-Pen* in 2nd place and *Multi-Hyb-DB* in 3rd.

Sensor Network Problems: Table 6.15 shows median results for unsolvable sensor networks problems with 5 to 8 targets, 25-64 sensors (grids of 5, 6, 7 and 8), k-visibility of 2, k-compatibility of 1, probability of visibility of 0.9 and probability of compatibility of 0.6. The ratio of intra-agent to inter-agent constraints is 15% to 85%. Consequently, all agents had solutions to their complex local problem but there was no global solution.

		Median number of messages			
Num Targets	**Num Sensors**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**
5	25	1,293	**730**	2,309	5,524
5	36	875	**560**	864	4,657
5	49	1,006	**531**	680	3,346
5	64	554	**320**	381	3,043
6	25	2,771	**1,723**	16,002	14,968
6	36	7,819	**2,306**	3,469	20,989
6	49	176	**136**	320	2,365
6	64	1156	815	**514**	925
7	25	7,235	**4,047**	26,807	25,103
7	36	5,962	**2,775**	5,429	7,043
7	49	721	**574**	693	3,731
7	64	2,041	1,501	**503**	1,155
8	25	20,488	**13,809**	112,189	105,417
8	36	8,333	**5,098**	24,051	88,030
8	49	1,011	**641**	1,068	6,415
8	64	6,539	5,295	**932**	2,470
		Median number of NCCCs			
Num Targets	**Num Sensors**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**
5	25	**22,275**	29,873	49,663	92,273
5	36	**15,229**	20,391	24,703	77,773
5	49	22,827	24,551	**22,292**	64,488
5	64	**9,225**	9,787	13,227	61,675
6	25	**110,032**	131,431	198,139	225,797
6	36	821,636	821,633	**57,489**	288,281
6	49	**3,037**	3,364	9,802	33,164
6	64	37,684	38,626	12,827	**11,363**
7	25	331,460	431,012	**290,947**	347,372
7	36	65,204	**55,508**	76,948	78,664
7	49	9,608	11,516	16,481	44,210
7	64	30,609	39,313	11,938	**11,393**
8	25	1,556,926	2,071,355	**990,653**	1,321,611
8	36	**153,330**	226,993	299,894	935,045
8	49	**19,284**	20,455	25,350	57,949
8	64	337,895	379,813	**17,372**	25,679

Table 6.15: Median results on unsolvable Grid-based Sensor Network problems.

Multi-Hyb-DB offers the most consistent performance for number of messages. Multi-ABT also performs well and occasionally outperforms *Multi-Hyb-DB*. Each algorithm (*Multi-Hyb-Pen*, *Multi-Hyb-DB*, Multi-ABT and Multi-AWCS) is optimal for different problem combinations for NCCCs.

6.5 Evaluating Multi-Hyb's Components

In the four problem classes tested (randomly generated, graph colouring, scheduling and sensor network problems), *Multi-Hyb-Pen* and *Multi-Hyb-DB* often produced significantly better results than the other algorithms (Multi-ABT, Multi-AWCS, Multi-DisPeL and DisBO-wd) for both solvable and unsolvable problems.

In order to ascertain the reasons for the success of the *Multi-Hyb* approach, an analysis of the performance of each component of *Multi-Hyb-Pen* and *Multi-Hyb-DB* was conducted. Table 6.16 shows the breakdown for solvable randomly generated problems (60 variables, 8 domain values, 5 agents, 90% intra-agent constraints, 10% inter-agent constraints, 0.2 constraint density and 0.35 constraint tightness), solvable graph colouring problems (150 nodes, 3 colours, 15 agents, 85% intra-agent constraints, 15% inter-agent constraints and degree of 4.9) and for solvable meeting scheduling problems (60 meetings, 7 time units, 90% intra-agent constraints, 10% inter-agent constraints, 5 departments and 0.18 constraint density).

	Multi-Hyb-Pen				*Multi-Hyb-DB*			
	SEBJ	*DisPeL-1C*	*SynCBJ-CLP*	**Total**	*SEBJ*	**DisBO-wd**	*SynCBJ-CLP*	**Total**
Random DisCSPs								
Solved	-	29%	71%	100%	-	29%	71%	100%
Msgs	-	322	22	399	-	284	5	323
NCCCs	142,005	14,723	1,269	163,585	166,143	7,158	254	170,093
Graph Colouring								
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	10	52	88	-	176	49	226
NCCCs	4,914	348	1,051	7,011	4,914	2,437	1,007	8,002
Meeting Scheduling								
Solved	-	81%	19%	100%	-	65%	35%	100%
Msgs	-	20	5	20	-	60	5	60
NCCCs	10,761	107	32	10,777	12,050	128	32	12,076

Table 6.16: Median Phase Results.

SEBJ does not incur any messages but performs a high proportion of NCCCs due to the generation of possibly all externally relevant (non-interchangeable) solutions to the local problem. *SynCBJ-CLP* is able to solve problems with relatively few messages and NCCCs because of the knowledge learnt from local search's short execution and *SEBJ*'s partial solutions.

Comparing the two variants (*Multi-Hyb-Pen* and *Multi-Hyb-DB*), it is interesting to note that on random problems, DisBO-wd not only uses less messages and NCCCs than *DisPeL-1C* but also provides a better ordering for *SynCBJ-CLP*. It is only let down by the fact that those problems solved by DisBO-wd alone take longer than *DisPeL-1C* which contributes to a higher number of NCCCs for *SEBJ*. For graph colouring and meeting scheduling, DisBO-wd requires more effort which contributes to the higher values for *Multi-Hyb-DB*.

6.6 Contributions

The following contributions have been made:

1. The *Multi-Hyb* approach has been developed. This 2-phase approach finds the externally relevant solutions for each agent's complex local problem whilst participating in a distributed local search to find a global solution. If distributed local search fails to find a solution, a distributed systematic search uses the knowledge learned from all searches to find a global solution.

2. *SEBJ* which finds all externally relevant solutions to an agent's complex local problem.

3. *DisPeL-1C* which continually imposes penalties rather than waiting for 2 cycles of quasi-local-minima. In addition, *DisPeL-1C* uses complex variables (one per agent).

4. A revised DisBO-wd to use complex variables within our *Multi-Hyb-DB* implementation.

5. Two implementations of the *Multi-Hyb* approach have been presented: *Multi-Hyb-Pen* using the penalty-on-values local search strategy and *Multi-Hyb-DB* using the breakout local search strategy.

6.7 Summary

In this chapter, *Multi-Hyb*, a concurrent distributed complete approach for solving DisCSPs with complex local problems has been presented. Specifically, a centralised systematic search is run concurrently for each agent to find all externally relevant solutions for each agent's local problem whilst each agent also participates in a distributed local search to find a suitable combination of local problem solutions which does not violate any inter-agent constraint. If none of the centralised systematic searches detects unsolvability and all systematic searches have found all non-interchangeable solutions and the distributed local search does not find a solution to the problem, a distributed systematic search algorithm is run. This algorithm benefits from: (i) not having to check intra-agent constraints since it has the local problem solutions obtained by centralised systematic search; (ii) being able to use possible 'best values' found by distributed local search; (iii) using knowledge learnt about the difficult areas of the problem from distributed local search.

We have presented two implementations of our approach: *Multi-Hyb-Pen* and *Multi-Hyb-DB*. *Multi-Hyb-Pen* uses *SEBJ* as the centralised systematic search solver, *DisPeL-1C* as the distributed local search algorithm and *SynCBJ-CLP* as the distributed systematic search algorithm. *Multi-Hyb-DB* uses *SEBJ* as the centralised systematic search solver, DisBO-wd as the distributed local search algorithm and *SynCBJ-CLP* as the distributed systematic search algorithm. *Multi-Hyb-Pen* and *Multi-Hyb-DB* have been shown to often outperform Multi-ABT, Multi-AWCS, Multi-DisPeL and DisBO-wd on distributed randomly generated, distributed graph colouring, distributed meeting scheduling and distributed sensor network problems.

However, there is an issue with the *Multi-Hyb* approach which can cause its performance to degrade for particular types of problems. Specifically, there are some problems where local search is unable to find a solution but a global solution does exist. For these problems, *Multi-Hyb* must find all externally relevant solutions to local problems before it is able to find a solution through the *SynCBJ-CLP* algorithm. Conversely, if all local problems have many solutions but there is no global solution to a problem, *Multi-Hyb* cannot detect this until all externally relevant solutions are found and the *SynCBJ-CLP* algorithm runs. In both of these scenarios, *Multi-Hyb* could have wasted a large number of non-concurrent constraint checks as well as messages. In the next chapter, we present another novel hybrid approach for DisCSPs with Complex Local Problems which is specifically aimed at tackling this limitation of *Multi-Hyb*.

Chapter 7

Multi-HDCS - Solving DisCSPs With Complex Local Problems Cooperatively

7.1 Introduction

In this chapter, the Hybrid Decomposition Concurrent Search approach for Complex Local Problems (*Multi-HDCS*) is presented. *Multi-HDCS* revises *Multi-Hyb* (see 6) by running the distributed systematic search whilst the agents search for solutions to their local problem and participating in a distributed local search. Information learnt about difficult variables by a distributed local search algorithm is now synchronised on a regular basis by distributed systematic search. Whilst existing distributed local searches can be used, a new distributed systematic search algorithm is presented to ensure completeness whilst additional solutions to complex local problems are added. A diagram of the approach is shown in figure 7.1.

The principle differences with the Multi-Hyb diagram (see figure 6.1) are: (i) the approach has a single phase and consequently the distributed systematic search now runs at the same time as the distributed local search and centralised systematic searches; (ii) information is now synchronised from distributed local search to distributed systematic

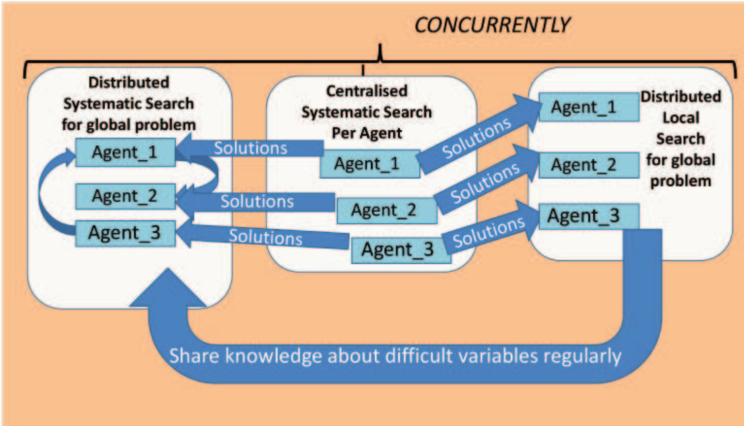

Figure 7.1: The Multi-HDCS approach.

search regularly rather than between the two phases in Multi-Hyb.

An overview of the approach and algorithms presented in this chapter is shown in table 7.1.

Approach	Algorithm	Local Search Strategy
Multi-HDCS	Multi-HDCS-Pen	Penalties on values
Multi-HDCS	Multi-HDCS-DB	Constraint weights ('Breakout')

Table 7.1: Chapter Overview.

7.2 Description of approach

Multi-HDCS is a novel distributed hybrid approach for solving DisCSPs with complex local problems. Our approach runs a centralised systematic search and two distributed search algorithms concurrently to solve the problem. In order to explain the approach, the very simple timetabling problem presented in section 6.2 is used.

In *Multi-HDCS* (see Algorithm 7), each agent attempts to solve their own local problem using a centralised systematic search (step 5). This search finds all solutions to an agent's complex local problem which are **externally relevant**, i.e. all **non-interchangeable solutions** which satisfy all intra-agent constraints and are distinguishable when looking only at externally relevant variables (involved in inter-agent constraints). In the simple timetabling example (see section 6.2 and figure 6.2), a centralised systematic search would be run for each department. If any department is unable to find a solution to its complex local problem, the problem is unsolvable. Otherwise, after all agents have found at least one solution to their local problem, two **additional distributed searches** (local search and systematic search) are **started concurrently** (steps 10 and 11), which attempt to find a solution which satisfies all inter-agent constraints. The centralised systematic search (step 5) continues to run and dynamically communicates new value combinations to both distributed searches. Meanwhile, the distributed local search (step 10) identifies difficult parts of the problem and passes this information to the distributed systematic search (step 11) at synchronisation points, to be used for dynamic variable ordering. *Multi-HDCS* does not exchange values from distributed local search to distributed systematic search as *Multi-Hyb* does, because of the dynamic addition of values to distributed systematic search.

Algorithm 7 *Multi-HDCS*

1: Initialise each agent with its complex local problem
2: *done* $\leftarrow false$
3: **while** not(*done*) **concurrently do**
4:     **for** each agent a_i with local problem lp_i **concurrently do**
5:         **Run a centralised systematic search** to find all **externally relevant** solutions to lp_i
6:     **end for**
7:     **if** A centralised systematic search finds no solutions to its local problem **then**
8:         *done* $\leftarrow true$
9:     **else if** Once all agents have found at least one solution to their local problem **then**
10:         **Run a distributed local search** combining local problem solutions (found in step 5), checking **inter-agent constraints** only. Regularly pass the knowledge learnt during search to the distributed systematic search below (step 8)
11:         **Run a distributed systematic search** combining local problem solutions (found in step 5) and using distributed local search's findings (from step 7) to dynamically order agents
12:         **if** Distributed local search or distributed systematic search finds a solution or distributed systematic search detects that the problem is unsolvable **then**
13:             *done* $\leftarrow true$
14:         **end if**
15:     **end if**
16: **end while**
17: **if** distributed local search or distributed systematic search has found solution S **then**
18:     return S
19: **else**
20:     Return "Unsolvable problem" as either the centralised systematic search (step 5) or the distributed systematic search (step 8) has detected unsolvability
21: **end if**

An overview of the properties of the different components of *Multi-HDCS* is given in table 7.2.

Component	Variables	Variable Ordering	Domains	Constraints Considered	Knowledge Exchanged
Centralised Systematic Searches	All variables in an agent	Static	Static	Intra-agent constraints	Solutions to complex local problems passed to distributed local search and distributed systematic search.
Distributed Local Search	All externally relevant variables in an agent	Static	Dynamic	Inter-agent constraints	Solutions to complex local problems from centralised systematic searches. Knowledge about difficult variables and best values passed to distributed systematic search.
Distributed Systematic Search	One complex variable per agent	Dynamic	Dynamic	Inter-agent constraints	Solutions to complex local problems from centralised systematic searches. Knowledge about difficult variables regularly synchronised from distributed local search.

Table 7.2: Overview of Multi-HDCS components.

7.2.1 Completeness

The centralised systematic searches (step 5) are guaranteed to find all non-interchangeable solutions to the complex local problems. If one of these problems does not have a solution, the centralised systematic search for that problem and consequently *Multi-HDCS* will detect unsolvability.

If, however, all complex local problems have at least one solution, the distributed systematic search (step 11) will either find a solution or detect unsolvability. Note that the distributed systematic search can complete its run whilst the centralised systematic searches are still running if a solution to the global problem is found.

In the case of unsolvable problems, we must make sure that the distributed systematic search cannot miss values. Therefore, nogoods contain a justification. These justifications can then be used to determine which nogoods can be removed when a new value is added - in a similar way to when a constraint is retracted in dynamic CSPs. When a new value is added to variable x_k , all nogoods containing variable x_k as a justification are removed. Should a new value be added to a variable, whilst it is choosing a value, then that value is considered when all other values have been considered (i.e. as the last value in the domain). Should a variable which has been assigned a value gain a new value, this new value will be attempted when that variable is backjumped or backtracked to (note if the variable is backjumped over then the new value for this variable is attempted). If a variable which has yet to be assigned gains a new value, then the new value will be attempted along with the others when the variable is processed. Additionally, the distributed systematic search's first agent cannot be changed dynamically whilst all other agents can change position in the ordering. Consequently, when the distributed systematic search returns to the first agent and the first agent has no more values to try, the distributed systematic search pauses. This is contrary to normal distributed systematic search which would terminate at this point. Whilst in pause mode, the next step of the distributed systematic search would either be: (i) once centralised systematic search detects a new solution for an agent and that agent is in one of the nogoods which caused the algorithm to enter pause mode, the algorithm tries these new values; (ii) if all centralised systematic searches for agents composed in the nogood which caused the algorithm to enter pause mode finish without finding more solutions, the distributed systematic search terminates.

Consequently, the distributed systematic search can only terminate if all values have been found and so no values are missed and the distributed systematic search is complete.

7.2.2 Termination

Each instance of the centralised systematic search terminates when either: (i) it has found all non-interchangeable solutions to its local problem; (ii) it detects the unsolvability of its local problem and has informed all other agents; (iii) receives a message from one of the agents stating that the problem is unsolvable; (iv) receives a message from either the distributed local search or the distributed systematic search stating that the problem has been solved. Since distributed local search and distributed systematic search only start once all agents have found at least one solution, the distributed local search and the distributed systematic search would not run if one or more agents had no solution to their local problem. Consequently, we now only need to consider cases where all agents have found at least one solution to their local problem. The distributed local search stops when either: (i) it has found a solution or; (ii) the distributed systematic search has either found a solution or detected unsolvability. The distributed systematic search terminates when either: (i) it has found a solution; (ii) it has detected that the problem is unsolvable once the centralised systematic searches have completed their search; (iii) the distributed local search has found a solution. Since the centralised systematic searches, the distributed local search and the distributed systematic search terminate, *Multi-HDCS* also terminates.

7.3 Implementations

We present two implementations of our approach: *Multi-HDCS-Pen* and *Multi-HDCS-DB*. The approaches differ in the strategy used for local search: penalties on values (*Multi-HDCS-Pen*) and weights on constraints (*Multi-HDCS-DB*).

7.3.1 Multi-HDCS-Pen

Multi-HDCS-Pen runs *SEBJ* (see chapter 6) as the centralised systematic search algorithm, *InterDisPeL* (see below) as the distributed local search algorithm and *InterPODS* (see below) as the distributed systematic search algorithm.

SEBJ finds all solutions to an agent's complex local problem which are externally relevant, i.e. all non-interchangeable solutions which satisfy all intra-agent constraints and are distinguishable when looking only at externally relevant variables (involved in inter-agent constraints). This *SEBJ* algorithm was already presented as part of *Multi-Hyb* in section 6.3.1 to which the reader is referred for a full description of the algorithm.

InterDisPeL is a penalty-based distributed local search algorithm inspired by Multi-DisPeL [8]. Unlike Multi-DisPeL, *InterDisPeL*: (i) considers only inter-agent constraints; (ii) maintains, for each agent, an overall count of the penalties it has imposed in the spirit of *PenDHyb* (see section 5.3.1). Thus, whenever a penalty is imposed on an agent's variable, the agent's penalty count is increased. This allows *InterDisPeL* to detect the complex local problems that are difficult to solve (i.e. with high penalties) and inform *InterPODS* (see below).

InterPODS (see Algorithms 8 and 9) is a new systematic algorithm for solving **inter-agent constraints** which uses **complex variables**. *InterPODS* is inspired by the much simpler PenDHyb algorithm (see section 5.3.1) with substantial differences: (i) each *In-*

terPODS agent knows only those value combinations which are compatible with the local problem's intra-agent constraints; (ii) *InterPODS* only considers inter-agent constraints; (iii) *InterPODS* uses complex variables; (iv) the next agent for processing is chosen dynamically based on the maximum degree heuristic, the minimum domain heuristic and each agent's penalty count obtained from the concurrent *InterDisPeL* search. For example, assuming that maximum degree and minimum domain were the same for the agents representing *computing* and *business* then if agent *art* has already selected a value for its complex variable and *computing* and *business* have penalties of 0 and 3, *InterPODS* will select the *business* agent for processing. The penalty information is synchronized with *InterDisPeL*'s current penalty counts regularly.

Algorithm 8 *InterPODS*

```
initialise agents with partial solutions from centralised systematic search as its domain
set first_agent and curr_agent to highest agent in ordering schema.
ChooseVal(curr_agent)
while messages exist do
   if receive backjumping message with backjumping_agent then
      ChooseVal (backjumping_agent)
   else if receive cpa message with next_agent then
      ChooseVal (next_agent)
   else if "solution found" then
      stop algorithm and return "solution found"
   else if "no solution found" then
      stop algorithm and return "no solution found"
   end if
end while
```

Determining the optimal variable ordering for *InterPODS* in *Multi-HDCS*-Pen

Experiments were conducted to measure the effectiveness of various dynamic orderings for *InterPODS* in the *Multi-HDCS-Pen* algorithm. If there remains a tie after considering all parts of the ordering, then agents are chosen lexicographically. The following orderings were considered:

1. PenCount - Choose the agent with the variable that has the highest penalty count as the next agent.

2. PenCount+MaxDeg - Choose the agent with the highest penalty as the next agent.

Algorithm 9 procedure ChooseVal(*curr_agent*)

```
for each value d_i in agent curr_agent's domain do
  if all higher priority constraints are satisfied then
    if all higher priority nogoods are not consistent with agent values then
      assign value d_i representing the chosen local solution for that agent's problem to agent
        curr_agent in cpa
      set next_agent to next agent dynamically chosen from ordering schema.
      if next_agent = last_agent then
        return "solution found"
      end if
      send message to next_agent with cpa
    end if
  else if higher priority constraints are violated then
    for each higher priority constraint which is violated do
      record the agent and value pair as part of a nogood value d_i to agent curr_agent
    end for
  end if
end for
if centralised systematic search has found new solutions then
  synchronize domain with centralised systematic search.
  remove nogoods containing agents who have new values.
  return chooseVal(curr_agent).
end if
if local search has updated penalty counts then
  synchronize information from local search.
end if
if curr_agent is first_agent and has no assigned value and centralised systematic search has
    terminated then
  return "unsolvable problem"
else if curr_agent is first_agent and has no assigned value but centralised systematic search
    has not terminated then
  pause algorithm and wait for centralised systematic search to retrieve new solutions or
    terminate.
else if curr_agent has no assigned value then
  Create a conflict set for agent curr_agent containing all agents involved in nogoods for values
    belonging to agent curr_agent
  if any agents between the lowest priority agent in the conflict set and this agent have gained
      new values then
    Set that agent to be curr_agent.
    return chooseVal(curr_agent)
  else
    Send a backjump message to the lowest priority agent in the conflict set.
  end if
end if
```

If there is a tie, choose the agent with the highest number of neighbours.

3. PenCount+MinDom - Choose the agent with the highest penalty as the next agent.

   If there is a tie, choose the agent with the minimum number of domain values.

4. MaxDeg+PenCount - Choose the agent with the highest number of neighbours as

the next agent. If there is a tie, choose the agent with the variable that has the highest penalty count.

5. MinDom+PenCount - Choose the agent with the minimum number of domain values as the next agent. If there is a tie, choose the agent with the variable that has the highest penalty count.

6. PenCount+MaxDeg+MinDom - Choose the agent with the variable that has the highest penalty count as the next agent. If there is a tie, choose the agent with the highest number of neighbours. If there remains a tie, choose the agent with the minimum number of domain values.

7. PenCount+MinDom+MaxDeg - Choose the agent with the variable that has the highest penalty count as the next agent. If there is a tie, choose the agent with the minimum number of domain values. If there remains a tie, choose the agent with the highest number of neighbours.

8. MaxDeg+PenCount+MinDom - Choose the agent with the highest number of neighbours as the next agent. If there is a tie, choose the agent with the variable that has the highest penalty count. If there remains a tie, choose the agent with the minimum number of domain values.

9. MaxDeg+MinDom+PenCount - Choose the agent with the highest number of neighbours as the next agent. If there is a tie, choose the agent with the minimum number of domain values. If there remains a tie, choose the agent with the variable that has the highest penalty count.

10. MinDom+PenCount+MaxDeg - Choose the agent with the minimum number of domain values as the next agent. If there is a tie, choose the agent with the variable that has the highest penalty count. If there remains a tie, choose the agent with the highest number of neighbours.

11. MinDom+MaxDeg+PenCount - Choose the agent with the minimum number of domain values as the next agent. If there is a tie, choose the agent with the highest

number of neighbours. If there remains a tie, choose the agent with the variable that has the highest penalty count.

We compared the 11 different orderings described above on randomly generated problems with 60 variables, 8 domain values, 5 agents, 75% intra-agent constraints and 25% inter-agent constraints, 0.2 constraint density and 0.35 constraint tightness. Table 7.3 shows a breakdown of the performance of each component of the *Multi-HDCS-Pen* algorithm since differences in the ordering can impact on each of the components.

	SEBJ	**InterDisPeL**	**InterPODS**	**Total**	**SEBJ**	**InterDisPeL**	**InterPODS**	**Total**
	PenCount				**PenCount+MaxDeg**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	1,738	441	3,162	-	2,066	387	3,329
NCCCs	266,672	157,602	214,495	586,856	267,462	196,784	212,261	598,744
	PenCount+MinDom				**MaxDeg+PenCount**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	2,440	410	3,758	-	1,978	363	3,352
NCCCs	259,580	204,151	251,006	692,831	267,200	195,383	280,419	650,655
	MinDom+PenCount				**PenCount+MaxDeg+MinDom**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	2,354	473	3,607	-	2,214	526	3,635
NCCCs	271,059	239,281	254,242	663,786	273,970	240,934	264,822	685,820
	PenCount+MinDom+MaxDeg				**MaxDeg+PenCount+MinDom**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	1,792	359	3,824	-	2,036	424	3,434
NCCCs	251,599	175,955	232,281	705,668	271,059	190,326	213,166	631,394
	MaxDeg+MinDom+PenCount				**MinDom+PenCount+MaxDeg**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	1,850	366	33,42	-	2,056	354	**2,929**
NCCCs	271,059	201,906	183,491	**578,160**	272,615	190,852	279,529	685,820
	MinDom+MaxDeg+PenCount							
Solved	-	0%	100%	100%				
Msgs	-	1,942	312	3,367				
NCCCs	273,970	213,788	253,397	663,140				

Table 7.3: Comparison of different orderings for InterPODS in the Multi-HDCS-Pen algorithm.

The best performing heuristic for messages is 'MinDom+PenCount+MaxDeg' whilst the best performing heuristic for NCCCs is 'MaxDeg+MinDom+PenCount'. A normalization of the results showed that the difference in constraint checks was more significant and therefore the heuristic recommended for determining the choice of next agent for *InterPODS* in the *Multi-HDCS-Pen* algorithm is 'MaxDeg+MinDom+PenCount'.

7.3.2 Multi-HDCS-DB

Multi-HDCS-DB runs *SEBJ* (see section 6) as the centralised systematic search algorithm, *InterDisBO-wd* (see below) as the distributed local search algorithm and *InterPODS* as the distributed systematic search algorithm.

InterDisBO-wd is inspired by the breakout-based algorithm DisBO-wd [8]. Unlike DisBO-wd, *InterDisBO-wd*: (i) checks only inter-agent constraints; (ii) considers only variable-value combinations approved by *SEBJ*; (iii) maintains, for each agent, a cumulative constraint-weight counter, i.e. the sum of the weights on all constraints which involve one of the agent's variables. These counters enable the indentification of complex local problems which are difficult to solve (i.e. with high constraint weights) to guide the *InterPODS* systematic search.

InterPODS has already been presented above for *Multi-HDCS-Pen*. The version of *InterPODS* used in *Multi-HDCS-DB* differs only in that the next agent for processing is now chosen dynamically based on each agent's constraint weight from the concurrent *InterDisBO-wd* search with ties broken by minimum domain and maximum degree heuristics. The constraint weight information is synchronized with *InterDisBO-wd*'s current constraint weights regularly.

Determining the optimal variable ordering for *InterPODS* in *Multi-HDCS*-DB

Experiments were also conducted to measure the effectiveness of various dynamic orderings for *InterPODS* in the *Multi-HDCS-DB* algorithm. We assume agents are chosen lexicographically if there remains a tie after considering all parts of the ordering. The following orderings were considered:

1. ConWeight - Choose the agent with the highest constraint weight as the next agent.
2. ConWeight+MaxDeg - Choose the agent with the highest constraint weight as the next agent. If there is a tie, choose the agent with the highest number of neighbours.
3. ConWeight+MinDom - Choose the agent with the highest constraint weight as the next agent. If there is a tie, choose the agent with the minimum number of domain

values.

4. MaxDeg+ConWeight - Choose the agent with the highest number of neighbours as the next agent. If there is a tie, choose the agent with the highest constraint weight.

5. MinDom+ConWeight - Choose the agent with the minimum number of domain values as the next agent. If there is a tie, choose the agent with the variable that has the highest constraint weight.

6. ConWeight+MaxDeg+MinDom - Choose the agent with the highest constraint weight as the next agent. If there is a tie, choose the agent with the highest number of neighbours. If there remains a tie, choose the agent with the minimum number of domain values.

7. ConWeight+MinDom+MaxDeg - Choose the agent with the highest constraint weight as the next agent. If there is a tie, choose the agent with the minimum number of domain values. If there remains a tie, choose the agent with the highest number of neighbours.

8. MaxDeg+ConWeight+MinDom - Choose the agent with the highest number of neighbours as the next agent. If there is a tie, choose the agent with the highest constraint weight. If there remains a tie, choose the agent with the minimum number of domain values.

9. MaxDeg+MinDom+ConWeight - Choose the agent with the highest number of neighbours as the next agent. If there is a tie, choose the agent with the minimum number of domain values. If there remains a tie, choose the agent with the highest constraint weight.

10. MinDom+ConWeight+MaxDeg - Choose the agent with the minimum number of domain values as the next agent. If there is a tie, choose the agent with the highest constraint weight. If there remains a tie, choose the agent with the highest number of neighbours.

11. MinDom+MaxDeg+ConWeight - Choose the agent with the minimum number of domain values as the next agent. If there is a tie, choose the agent with the highest number of neighbours. If there remains a tie, choose the agent with the highest constraint weight.

These orderings were compared on randomly generated problems with 60 variables, 8 domain values, 5 agents, 75% intra-agent constraints and 25% inter-agent constraints, 0.2 constraint density and 0.35 constraint tightness. Table 7.4 show a breakdown of the performance of each component of the *Multi-HDCS-DB* algorithm since differences in the ordering can impact on each of the components.

	SEBJ	**InterDisBO-wd**	**InterPODS**	**Total**	**SEBJ**	**InterDisBO-wd**	**InterPODS**	**Total**
	ConWeight				**ConWeight+MaxDeg**			
Solved	-	3%	97%	100%	-	0%	100%	100%
Msgs	-	256	461	934	-	301	352	825
NCCCs	247,659	87,069	327,053	442,620	244,794	91,079	159,456	455,162
	ConWeight+MinDom				**MaxDeg+ConWeight**			
Solved	-	0%	100%	100%	-	1%	99%	100%
Msgs	-	271	65	407	-	295	338	829
NCCCs	254,449	89,948	87,292	332,431	241,733	92,172	177,845	454,175
	MinDom+ConWeight				**ConWeight+MaxDeg+MinDom**			
Solved	-	2%	98%	100%	-	0%	100%	100%
Msgs	-	306	59	428	-	265	282	838
NCCCs	267,462	99,167	82,713	354,079	248,666	84,515	170,357	393,921
	ConWeight+MinDom+MaxDeg				**MaxDeg+ConWeight+MinDom**			
Solved	-	2%	98%	100%	-	0%	100%	100%
Msgs	-	177	72	**345**	-	254	261	752
NCCCs	236,915	58,568	81,599	**299,226**	251,859	82,716	146,823	371,538
	MaxDeg+MinDom+ConWeight				**MinDom+ConWeight+MaxDeg**			
Solved	-	1%	99%	100%	-	2%	98%	100%
Msgs	-	244	350	811	-	264	63	446
NCCCs	231,302	80,754	164,482	382,864	256,759	85,713	82,448	344,894
	MinDom+MaxDeg+ConWeight							
Solved	-	2%	98%	100%				
Msgs	-	322	50	488				
NCCCs	289,564	110,573	75,620	361,578				

Table 7.4: Comparison of different orderings for InterPODS in the Multi-HDCS-DB algorithm.

For *Multi-HDCS-DB*, the 'ConWeight+MinDom+MaxDeg' ordering performs significantly better than all other orderings and is therefore the recommended ordering for *Multi-HDCS-DB*.

7.3.3 Determining the Optimal Synchronisation Interval

We also conducted experiments to determine the optimal synchronisation interval. For *Multi-HDCS-Pen*, we tried intervals of $\in 1, 2, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70$. We chose 70 as the upper limit since *InterDisPeL* never ran longer than 70 cycles. For *Multi-HDCS-DB*, we used the same intervals but doubled them so as to reflect the 2 cycles of *InterDisBO-wd* which equal 1 cycle of *InterDisPeL*. Therefore, the intervals were $\in 2, 4, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140$. 140 was the upper limit of *InterDisBO-wd*'s runs and therefore this was chosen as the upper limit. Median results on randomly-generated problems with 60 variables, 8 domain values, 5 agents, 75% intra-agent constraints and 25% inter-agent constraints, 0.2 constraint density and 0.35 constraint tightness are shown in table 7.5 for *Multi-HDCS-Pen* and table 7.6 for *Multi-HDCS-DB*.

	SEBJ	**InterDisPeL**	**InterPODS**	**Total**	**SEBJ**	**InterDisPeL**	**InterPODS**	**Total**
	1 Cycle				**2 Cycles**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	4,680	324	6,513	-	2214	526	3,635
NCCCs	291,892	360,668	262,949	844,147	273,970	240,934	264,822	**685,820**
	5 Cycles				**10 Cycles**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	2,460	333	**3,586**	-	2,654	477	3,953
NCCCs	271,059	261,448	218,012	692,536	273,970	275,379	285,727	705,286
	15 Cycles				**20 Cycles**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	2,594	390	4,765	-	2,492	413	4,240
NCCCs	280,052	346,553	285,090	761,118	256,946	294,755	194,816	761,888
	25 Cycles				**30 Cycles**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	3,066	459	4,548	-	2,832	470	4230
NCCCs	271,059	355,424	259,247	737,233	289,564	323,978	268,296	829,660
	35 Cycles				**40 Cycles**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	2,780	401	3,766	-	2,350	430	4,172
NCCCs	281,608	332,933	263,459	965,591	267,462	293,981	240,015	723,144
	45 Cycles				**50 Cycles**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	2,800	343	4,519	-	2,546	572	4,467
NCCCs	269,441	402,770	225,569	719,630	252,319	286,036	305,741	859,690
	55 Cycles				**60 Cycles**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	2,352	350	3,845	-	2,504	453	4,546
NCCCs	256,759	281,803	254,741	861,027	252,988	294,246	238,827	882,954
	65 Cycles				**70 Cycles**			
Solved	-	0%	100%	100%	-	0%	100%	100%
Msgs	-	2,752	368	4,687	-	2,550	368	4,171
NCCCs	273,970	336,760	292,014	856,126	271,059	308,131	227,352	907,079

Table 7.5: Comparison of synchronisation intervals for the Multi-HDCS-Pen algorithm.

	SEBJ	**InterDisBO-wd**	**InterPODS**	**Total**	**SEBJ**	**InterDisBO-wd**	**InterPODS**	**Total**
	2 Cycles				**4 Cycles**			
Solved	-	3%	97%	100%	-	1%	99%	100%
Msgs	-	855	119	1,390	-	307	61	445
NCCCs	289,564	209,341	183,983	494,221	271,059	100,440	81,660	356,640
	10 Cycles				**20 Cycles**			
Solved	-	2%	98%	100%	-	1%	99%	100%
Msgs	-	274	85	477	-	315	56	439
NCCCs	255,709	87,290	87,935	377,484	273,970	108,141	78,612	405,125
	30 Cycles				**40 Cycles**			
Solved	-	2%	98%	100%	-	1%	99%	100%
Msgs	-	328	58	441	-	366	52	474
NCCCs	273,970	101,326	90,473	375,701	289,564	116,197	81,858	371,930
	50 Cycles				**60 Cycles**			
Solved	-	4%	96%	100%	-	2%	98%	100%
Msgs	-	288	59	434	-	177	72	**345**
NCCCs	267,462	87,368	85,398	366,930	236,915	58,568	81,599	**299,226**
	70 Cycles				**80 Cycles**			
Solved	-	4%	96%	100%	-	2%	98%	100%
Msgs	-	288	59	434	-	257	61	389
NCCCs	267,462	87,368	85,398	366,930	259,896	73,587	85,060	330,311
	90 Cycles				**100 Cycles**			
Solved	-	0%	100%	100%	-	3%	97%	100%
Msgs	-	315	63	407	-	275	68	382
NCCCs	268817	106633	88443	364377	255709	88830	94429	348226
	110 Cycles				**120 Cycles**			
Solved	-	3%	97%	100%	-	0%	100%	100%
Msgs	-	296	62	439	-	358	60	473
NCCCs	239,845	88,376	92,288	319,583	269,745	121,373	86,017	383,369
	130 Cycles				**140 Cycles**			
Solved	-	1%	99%	100%	-	1%	99%	100%
Msgs	-	375	56	498	-	314	59	453
NCCCs	289,564	120,276	79,115	376,104	281,608	106,368	81,871	371,930

Table 7.6: Comparison of synchronisation intervals for the Multi-HDCS-DB algorithm.

It is interesting to note that the synchronisation intervals for *Multi-HDCS-Pen* and *Multi-HDCS-DB* are different. *Multi-HDCS-DB* benefits from longer synchronisation intervals whilst *Multi-HDCS-Pen* prefers shorter synchronisation intervals. However, changing the synchronisation interval does not have a large effect on performance. The optimal synchronisation interval for *Multi-HDCS-DB* is 60 to minimise both messages and constraint checks. The optimal synchronisation interval for *Multi-HDCS-Pen* is 5 to minimise messages and 2 to minimise constraint checks. When the results are normalised, the difference in constraint checks is larger so the optimal synchronisation interval is 2. We did consider synchronisation intervals of 3 and 4 but these produced worse results than 2 and 5. We use these synchronisation intervals in the experimental evaluation below.

7.4 Experimental Evaluation

An extensive experimental evaluation of two implementations of *Multi-HDCS* on both solvable and unsolvable problems has been carried out. The experimental evaluation compared *Multi-HDCS-Pen* and *Multi-HDCS-DB* against Multi-ABT, Multi-AWCS, *Multi-Hyb-Pen* and *Multi-Hyb-DB*. In addition, Multi-DisPeL and DisBO-wd were compared for solvable problems only. The reader is referred to section 6.4 for the verification of our implementations of these algorithms.

Multi-HDCS-Pen and *Multi-HDCS-DB* were evaluated on **distributed randomly generated problems**, **distributed 3-colour graph colouring problems** and **distributed meeting scheduling problems** measuring: (i) the number of messages sent between agents; (ii) the number of non-concurrent constraint checks (NCCCs) performed. Note that the number of messages/NCCCs required for termination detection are not included in the results for any of the algorithms as reported by other researchers [96]. Whilst CPU time is not an established measure for comparing DisCSP algorithms [56], the CPU time results matched the trends of other measures. The experiments focused on naturally distributed problems i.e. problems which have a high ratio of intra-agent to inter-agent constraint: between 90:10 and 70:30. The number of variables used ranged from 25 to 200. 100 different instances for each problem type (proportion of intra-agent to inter-agent constraints) were solved, with average and median results calculated.

An evaluation of *Multi-HDCS-Pen* and *Multi-HDCS-DB* on distributed sensor network problems is also presented. These problems are not naturally distributed since they have a relatively simple local problem within each agent and many inter-agent constraints. However, they are included in the comparison to determine the limitations of the *Multi-HDCS* approach.

7.4.1 Solvable Problems

An identical cutoff from the *Multi-Hyb* experiments in section 6.4 of $100n$ iterations for Multi-DisPeL and $200n$ iterations for DisBO-wd was used. Cases where Multi-DisPeL or DisBO-wd did not solve all problems are indicated in the results by "*".

Randomly Generated Problems

Table 7.7 presents the median for solvable randomly generated problems using 5 agents, a domain size of 8, a constraint density of 0.2 and constraint tightness of 0.35 for *Multi-HDCS-Pen* and *Multi-HDCS-DB* compared with systematic, hybrid search and local search algorithms.

When comparing the two implementations of the *Multi-HDCS* approach, *Multi-HDCS-DB* outperforms *Multi-HDCS-Pen* in general with the later occasionally performing better for NCCCs but never for messages. For number of messages with medium-sized problems (60 to 125 variables), *Multi-HDCS-DB* performed best with *Multi-Hyb-DB* in 2nd place and *Multi-Hyb-Pen* in 3rd. For problems with 125 or more variables, *Multi-Hyb-Pen* and Multi-DisPeL outperform *Multi-HDCS-DB* although the difference is very small. For NCCCs, *Multi-HDCS-DB* gives best results, followed by *Multi-HDCS-Pen* and *Multi-Hyb-Pen*. Consequently, whilst *Multi-Hyb-DB* and Multi-DisPeL do outperform *Multi-HDCS-DB* for large problems on messages, *Multi-HDCS-DB* gives the best results for NCCCs.

		Median number of messages							
Num Vars	**% intra:inter constraints**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
60	90:10	234	**60**	399	323	842	4,834	536	1,150*
60	80:20	344	**85**	197	158	1,692	5,287	422	1,165
60	70:30	278	**156**	818	833	6,832	4,475	496	985
70	80:20	130	**45**	159	96	731	3,672	208	435
70	70:30	264	**60**	112	175	1,141	3,907	194	420
80	80:20	70	**42**	143	60	440	3,991	104	335
80	70:30	117	**38**	89	60	500	6,076	108	295
90	80:20	70	**35**	94	60	336	4,242	66	275
90	70:30	125	**35**	81	60	298	6,193	80	265
100	80:20	70	**35**	56	60	248	5,922	56	235
100	70:30	70	**35**	78	60	276	7,235	60	225
125	80:20	70	35	**20**	60	197	6,297	40	225
125	70:30	70	**35**	60	60	152	9,218	40	205
150	80:20	70	35	**20**	60	152	6,803	28	215
150	70:30	70	35	**30**	46	128	14,554	32	195
175	80:20	70	35	**20**	45	134	10,707	24	210
175	70:30	70	35	**20**	45	118	15,126	24	190
		Median number of NCCCs							
Num Vars	**% intra:inter constraints**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
60	90:10	**59,560**	60,088	163,585	170,093	314,067	165,118	1,187,335	469,162*
60	80:20	75,413	**71,387**	277,408	268,336	420,384	277,408	949,616	440,862
60	70:30	1,012,213	537,988	2,761,171	2,626,087	286,821	**182,936**	1,148,704	353,862
70	80:20	50,698	**49,960**	151,678	133,577	284,713	124,238	745,608	252,678
70	70:30	88,373	**85,467**	291,421	288,457	524,487	135,090	673,099	244,962
80	80:20	**48,123**	49,126	118,874	114,283	207,389	149,599	588,111	283,827
80	70:30	56,643	**56,339**	169,884	153,848	356,405	265,274	606,084	262,707
90	80:20	46,855	**45,307**	117,668	105,869	278,057	177,570	611,811	308,444
90	70:30	52,380	**51,510**	140,181	130,355	224,968	291,656	638,729	299,228
100	80:20	44,687	**44,571**	107,836	101,792	214,806	285,431	690,977	339,423
100	70:30	**50,638**	52,368	132,031	125,176	265,460	385,969	690,455	324,668
125	80:20	46,992	**46,706**	106,435	104,718	185,646	357,508	952,787	509,090
125	70:30	51,280	**50,360**	125,553	121,680	360,376	600,688	936,775	485,739
150	80:20	45,587	**45,250**	100,020	102,519	235,880	441,287	1,362,161	728,427
150	70:30	54,756	**52,613**	120,105	128,039	268,777	1,302,570	1,281,866	682,116
175	80:20	45,774	**45,613**	98,875	103,143	155,900	885,339	1,926,771	976,712
175	70:30	51,805	**50,468**	110,325	124,838	235,168	1,453,996	1,831,216	908,710

Table 7.7: Median results for solvable randomly generated problems.

Graph Colouring Problems

Median results comparing *Multi-HDCS-Pen* and *Multi-HDCS-DB* against systematic, hybrid and local search algorithms for 3-colour distributed graph colouring problems with 150 to 200 nodes, 15 to 25 agents and 4.9 to 5.1 degree are presented in table 7.8.

Multi-Hyb-Pen gives best results for the number of messages with *Multi-HDCS-DB* in 2nd place and *Multi-Hyb-DB* in 3rd place. *Multi-HDCS-DB* offers consistent performance for NCCCs being optimal in the majority of cases but not all. For the other cases, *Multi-Hyb-Pen*, *Multi-HDCS-Pen* and Multi-ABT are optimal for different problem settings. Specifically, *Multi-HDCS-Pen* and *Multi-HDCS-DB* appear to improve the NCCCs for *Multi-Hyb-Pen* and *Multi-Hyb-DB* for problems in a number of cases owing to the increased concurrency but with a substantial increase in the number of messages. In particular, *Multi-HDCS-DB* would appear, in general, to be the better implementation of the *Multi-HDCS* approach for graph colouring problems.

Meeting Scheduling Problems

The meeting scheduling problems formulisation is described in section 2.3.3. Table 7.9 compares median results for *Multi-HDCS-Pen* and *Multi-HDCS-DB* with other leading algorithms for solvable meeting scheduling problems with 50-80 meetings, 5 departments (agents), timeframe of 6 or 7 units and constraint density of 0.18. The percentage of intra-agent constraints varied between 70% to 90%. Two departments with common meetings had a distance of between 1 and 3 time units.

Multi-Hyb-Pen performs best for number of messages with *Multi-HDCS-DB* in 2nd place. Occasionally, *Multi-HDCS-DB* outperforms *Multi-Hyb-Pen*. For NCCCs, Multi-ABT performs best in the majority of cases with *Multi-Hyb-Pen* and Multi-AWCS also performing best in some cases. For the *Multi-HDCS* approach, the computational effort of running two distributed algorithms in parallel is too high for this type of problem versus the simpler approach of *Multi-Hyb-Pen*, *Multi-Hyb-DB*, Multi-ABT and Multi-AWCS.

				Median number of messages							
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
150	15	4.9	90:10	486	120	**40**	155	490	1,281	595	855
150	15	5.1	90:10	481	120	**35**	163	608	1,437	714	840*
150	15	4.9	80:20	481	120	**21**	134	326	1,102	588	765*
150	15	5.1	80:20	481	128	**23**	143	350	1,248	616	900*
150	15	4.9	70:30	495	146	**31**	180	591	1,588	714	780*
150	15	5.1	70:30	467	122	**31**	185	629	1,909	735	900*
150	25	4.9	90:10	1,205	200	**35**	177	373	1,508	1,176	1,175*
150	25	5.1	90:10	1,182	200	**29**	179	399	1,534	1,176	1,200*
150	25	4.9	80:20	1,286	188	**53**	317	2,696	2,079	1,392	1,300*
150	25	5.1	80:20	996	200	**37**	245	1,053	2,423	1,368	1,325*
150	25	4.9	70:30	1,014	200	**42**	261	1,403	2,879	1,788	1,500*
150	25	5.1	70:30	1,102	204	**51**	338	3,642	3,362	1,680	1,275*
200	20	4.9	90:10	842	160	**62**	212	698	2,146	1,197	1,420*
200	20	5.1	90:10	832	160	**73**	223	938	2,328	1,216	1,300*
200	20	4.9	80:20	844	180	**31**	188	528	1,732	1,064	1,220*
200	20	5.1	80:20	803	141	**34**	196	544	1,851	1,140	1,340*
200	20	4.9	70:30	842	160	**59**	266	1,050	2,465	1,225*	1,200*
200	20	5.1	70:30	656	162	**77**	289	1,278	2,668	1,282	1,440*
200	25	4.9	90:10	1,253	200	**51**	233	657	2,350	1,716	1,425*
200	25	5.1	90:10	1,253	200	**45**	232	869	2,092	1,800	1,575*
200	25	4.9	80:20	1,253	200	**57**	252	911	2,396	1,680	1,450
200	25	5.1	80:20	1,040	204	**44**	250	1,068	2,446	1,692	1,425*
200	25	4.9	70:30	977	200	**56**	309	2,048	3,148	1,848	1,625*
200	25	5.1	70:30	1,277	172	**62**	339	2,746	3,259	1,992*	1,825*
				Median number of NCCCs							
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
150	15	4.9	90:10	1,387	**1,185**	3,579	3,735	1,266	3,172	46,215	66,583
150	15	5.1	90:10	1,449	**1,255**	3,689	3,837	1,589	3,435	57,967	63,567*
150	15	4.9	80:20	1,098	**1,081**	1,314	1,611	1,165	3,123	49,008	63,739*
150	15	5.1	80:20	**1,105**	1,107	1,279	1,653	1,278	3,310	51,564	73,127*
150	15	4.9	70:30	1,511	1,521	1,882	2,659	**1,501**	3,535	53,692	57,495*
150	15	5.1	70:30	**1,479**	1,500	1,783	2,507	1,535	4,058	59,712	68,942*
150	25	4.9	90:10	570	**423**	675	775	689	1,454	30,961	53,242*
150	25	5.1	90:10	541	**459**	633	757	724	1,417	33,134	53,127*
150	25	4.9	80:20	1,643	842	**729**	1,223	1,532	1,651	35,018	53,604*
150	25	5.1	80:20	632	800	**549**	800	1,017	1,974	37,113	55,657*
150	25	4.9	70:30	1,015	1,404	**726**	1,253	1,802	2,265	43,369	46,600*
150	25	5.1	70:30	572	854	**534**	801	1,218	2,087	43,344	57,361*
200	20	4.9	90:10	1,605	1,461	41,95	4,561	**1,434**	3,836	71,275	104,597*
200	20	5.1	90:10	1,530	**1,438**	4,403	4,646	1,716	4,185	70,314	105,865*
200	20	4.9	80:20	1,391	**1,272**	1,439	1,900	1,286	3,637	65,360	99,080*
200	20	5.1	80:20	1,319	**1,167**	1,467	1,925	1,273	3,623	72,354	106,079*
200	20	4.9	70:30	2,084	1,820	2,369	3,403	**1,604**	4,180	73,351*	89,740*
200	20	5.1	70:30	**1,670**	1,770	2,348	3,484	1,872	4,405	77,346	107,339*
200	25	4.9	90:10	997	**751**	1,843	2,154	1,014	2,723	61,481	87,216*
200	25	5.1	90:10	998	**769**	1,703	2,046	1,214	2,499	68,940	99,001*
200	25	4.9	80:20	1,106	**780**	972	1,261	1,267	2,669	61,118	89,533
200	25	5.1	80:20	**877**	939	878	1,225	1,424	2,903	66,544	89,009*
200	25	4.9	70:30	**1,033**	1,253	1,272	2,089	1,727	2,975	63,778	86,824*
200	25	5.1	70:30	1,619	**879**	1,372	2,156	2,029	3,121	71,947*	101,275*

Table 7.8: Median results for solvable graph colouring problems.

			Median number of messages							
Num Meetings	**Num Times**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
50	7	90:10	65	50	**20**	54	81	340	68	295*
50	7	80:20	71	**45**	139	75	204	415	96	335*
50	7	70:30	221	**73**	460	328	453	464	90	405*
50	6	90:10	65	50	**10**	45	64	269	52	155*
50	6	80:20	73	35	**20**	60	96	321	64	165*
50	6	70:30	70	**42**	184	102	161	362	66	215*
60	7	90:10	65	50	**20**	60	86	359	64	245*
60	7	80:20	70	**45**	80	60	136	396	76	275*
60	7	70:30	140	**49**	412	173	341	500	72	295*
60	6	90:10	65	50	**10**	45	78	288	32	145*
60	6	80:20	65	35	**10**	45	106	327	44	175*
60	6	70:30	173	**35**	42	60	149	409	56	225*
70	7	90:10	68	50	**20**	60	103	380	44	235*
70	7	80:20	70	42	**20**	60	128	428	56	255
70	7	70:30	74	**38**	228	90	205	514	64	315
70	6	90:10	65	40	**20**	45	91	274	40	165*
70	6	80:20	65	35	**20**	60	116	352	40	195
70	6	70:30	70	**35**	40	60	132	415	50	245
80	7	90:10	70	50	**20**	60	115	404	48	235
80	7	80:20	70	37	**20**	60	128	473	48	245
80	7	70:30	130	**36**	151	74	185	547	60	305
80	6	90:10	65	40	**20**	45	98	284	32	185
80	6	80:20	68	35	**20**	60	118	379	40	205
80	6	70:30	70	35	**20**	60	124	443	44	245
			Median number of NCCCs							
Num Meetings	**Num Times**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
50	7	90:10	8,623	7,571	7,162	7,369	**6,988**	7,309	112,308	110,290*
50	7	80:20	13,147	13,460	10,852	13,139	9,488	**8,214**	130,639	138,306*
50	7	70:30	19,763	19,847	20,684	25,451	13,774	**8,605**	120,664	126,017*
50	6	90:10	3,956	3,592	**2,933**	3,503	3,793	5,534	73,805	57,262*
50	6	80:20	5,881	5,146	4,803	5,259	**4,411**	5,974	79,868	84,785*
50	6	70:30	7,757	7,738	7,451	9,632	**5,238**	6,382	74,751	71,166*
60	7	90:10	15,114	12,833	10,777	12,076	10,901	**10,613**	160,589	158,103*
60	7	80:20	18,113	18,050	16,251	16,367	11,413	**10,821**	163,578	183,771*
60	7	70:30	33,811	33,999	37,138	36,649	15,464	**12,513**	153,894	158,777*
60	6	90:10	6,634	5,948	**5,095**	5,700	5,490	7,894	89,497	91,349*
60	6	80:20	6,353	6,428	6,163	6,346	5,981	8,249	99,156	107,302*
60	6	70:30	16,639	12,236	11,334	11,654	**6,766**	9,628	100,219	201,621*
70	7	90:10	18,496	18,255	15,377	17,757	**13,044**	13,739	198,303	203,387*
70	7	80:20	23,920	24,287	20,174	21,380	**12,956**	14,696	199,104	240,856
70	7	70:30	34,708	35,181	38,453	45,164	**15,624**	16,365	214,783	241,370
70	6	90:10	8,194	7,585	**6,586**	7,573	6,906	10,373	131,723	136,478*
70	6	80:20	9,627	9,827	9,523	9,632	**6,880**	11,512	129,821	154,904
70	6	70:30	14,191	14,768	14,375	12,949	**7,354**	12,123	148,914	163,560
80	7	90:10	20,834	20,432	17,434	17,651	**14,685**	18,715	270,668	280,138
80	7	80:20	30,384	30,172	27,460	26,809	**13,708**	19,959	263,191	303,366
80	7	70:30	47,197	49,563	50,844	50,219	17,888	21,276	271,813	312,564
80	6	90:10	8,587	8,407	8,863	8,461	**7,432**	14,264	177,645	187,330
80	6	80:20	13,515	11,258	10,967	11,202	7,687	14,796	185,343	223,587
80	6	70:30	15,926	16,750	14,073	15,645	8,512	16,331	186,795	224,357

Table 7.9: Median results for solvable meeting scheduling problems.

Sensor Network Problems

Finally, *Multi-HDCS-Pen* and *Multi-HDCS-DB* were evaluated against systematic, hybrid and local search algorithms on Grid-based SensorDCSP [100]. These problems are not naturally distributed since they have a large number of inter-agent constraints combined with relatively simple local problems for each agent. Consequently, the ratio is now 85% inter-agent constraints and 15% intra-agent constraints. They provide an interesting case to determine whether the *Multi-HDCS* approach also functions for problems which are not naturally distributed. The problems used had 5 targets, between 25 and 64 sensors (grids of 5, 6, 7, 8), k-visibility of 2, k-compatibility of 1, probability of visibility of 0.9 and probability of compatibility of 0.6. Median results for *Multi-HDCS-Pen* and *Multi-HDCS-DB* are shown in Table 7.10.

Whilst *Multi-Hyb-DB* and Multi-ABT are also optimal for some problems (and Multi-DisPeL but it does not solve all problems), *Multi-HDCS-DB* offers the most consistent performance for number of messages. For NCCCs, all algorithms except Multi-DisPeL and DisBO-wd are optimal for different problem combinations, but *Multi-HDCS-Pen* offers the most consistent performance.

7.4.2 Unsolvable Problems

Our experiments with unsolvable problems distinguish between two categories of unsolvable problems: (i) those where at least one complex local problem is unsolvable and; (ii) those where all complex local problems are solvable, but no overall solution exists.

Randomly Generated Problems: Median results for unsolvable randomly generated problems using 5 agents, a domain size of 8 and a constraint tightness of 0.35 are presented in table 7.11 for problems which have one or more complex local problems that are unsolvable. In these cases, *SEBJ* detects unsolvability and therefore *Multi-Hyb-Pen*, *Multi-Hyb-DB*, *Multi-HDCS-Pen* and *Multi-HDCS-DB* all perform identically. We found that the *Multi-Hyb* and *Multi-HDCS* implementations outperformed Multi-ABT and Multi-AWCS on both messages and NCCCs.

We also conducted experiments for problems that had solutions to all complex local

		Median n. Messages							
Num Targets	**Num Sensors**	**Multi HDCS-Pen**	**Multi -HDCS-DB**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
5	25	145	**50**	69	63	204	299	80*	575*
5	36	145	**40**	50	49	52	185	40*	285*
5	49	85	40	25	42	**24**	94	40*	160*
5	64	85	40	**14**	34	19	101	28*	120*
6	25	595	**121**	1,649	765	1,390	1,166	417*	1938*
6	36	210	**54**	1,383	242	145	333	105*	846*
6	49	210	**54**	338	116	60	185	60*	414*
6	64	120	54	510	310	**31**	127	50*	306*
7	25	15,737	**1,568**	3,814	2,300	8,786	3,492	**1,161***	4,907*
7	36	502	**100**	3,868	1,051	1,164	955	225*	1,960*
7	49	161	**63**	1,092	210	128	330	126*	609*
7	64	161	63	482	196	**55**	216	93*	658*
8	25	90,892	15,253	16,471	**3,644**	108,882	16,155	3,979*	25,608*
8	36	1,083	**318**	5,522	3,847	5,087	1,693	759*	3,840*
8	49	379	**76**	2,753	1,100	328	693	203*	1,296*
8	64	208	**74**	1,175	411	126	473	143*	768*
		Median n. NCCCs							
Num Targets	**Num Sensors**	**Multi HDCS-Pen**	**Multi -HDCS-DB**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**	**Multi -DisPeL**	**DisBO -WD**
5	25	**2,727**	4,716	4,072	6,599	8,859	5,959	40,031*	66,968*
5	36	**2,337**	2,512	2,936	5,353	4,329	3,888	25,707*	31,359*
5	49	**2,254**	2,374	2,708	3,431	2,755	2,314	18,280*	19,366*
5	64	2,371	2,373	2,541	2,759	2,294	**1,856**	14,432*	15,397*
6	25	13,266	17,087	**13,164**	49,144	27,603	19,024	194,721*	248,682*
6	36	2,782	5,436	7,819	**2,306**	9,159	5,645	47,195*	98,318*
6	49	2,406	2,651	5,706	**2,112**	4,544	3,474	29,704*	48,459*
6	64	2,512	2,594	18,774	**2,497**	3,230	2,588	24,140*	31,515*
7	25	263,885	253,031	120,789	133,882	114,529	**44,926**	453,891*	623,861*
7	36	**7,596**	12,321	8,622	23,240	27,975	12,062	112,370*	267,908*
7	49	2,570	4,662	21,124	**2,288**	7,149	4,886	53,062*	83,965*
7	64	**3,045**	6,737	7,420	36,938	11,884	8,217	81,203*	161,266*
8	25	2,477,556	2,678,899	1,395,619	**595,777**	970,639	190,699	1,335,327*	3,667,100*
8	36	36,801	39,546	21,999	133,809	75,134	**18,978**	281,020*	545,384*
8	49	**3,045**	6,737	7,420	36,938	11,884	8,217	81,203*	161,266*
8	64	2,854	3,713	19,316	**2,417**	7,726	6,110	56,022*	94,678*

Table 7.10: Median results for solvable Grid-based Sensor Network problems.

problems but no global solution with identical parameters. The results of these experiments are shown in Table 7.12. *Multi-HDCS-DB* significantly outperforms all other algorithms both for number of messages and for NCCCs. It would appear that *InterDisBO-wd* is able to give a very good ordering to *InterPODS* very early in the search that enables *InterPODS* to quickly determine that there is no global solution once all relevant *SEBJ* searches have finished.

			Median number of messages					
Num Vars	**% constraint density**	**% intra:inter constraints**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
60	0.2	90:10	**14**	**14**	**14**	**14**	647	38,169
70	0.2	80:20	**12**	**12**	**12**	**12**	420	46,792
70	0.2	70:30	**16**	**16**	**16**	**16**	682	48,959
80	0.2	80:20	**12**	**12**	**12**	**12**	285	53,343
80	0.2	70:30	**12**	**12**	**12**	**12**	353	56,070
90	0.18	80:20	12	12	12	12	**10**	58,800
90	0.18	70:30	**12**	**12**	**12**	**12**	292	62,809
100	0.16	80:20	**10**	**10**	**10**	**10**	**10**	64,706
100	0.16	70:30	**12**	**12**	**12**	**12**	371	69,132
125	0.2	80:20	**10**	**10**	**10**	**10**	**10**	80,695
125	0.2	70:30	**10**	**10**	**10**	**10**	**10**	86,454
150	0.2	80:20	**10**	**10**	**10**	**10**	**10**	95,779
150	0.2	70:30	**10**	**10**	**10**	**10**	**10**	102,960
175	0.2	80:20	**10**	**10**	**10**	**10**	**10**	111,218
175	0.2	70:30	**10**	**10**	**10**	**10**	**10**	119,188
			Median number of NCCCs					
Num Vars	**% constraint density**	**% intra:inter constraints**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
60	0.2	90:10	**52,826**	**52,826**	**52,826**	**52,826**	116,728	10,082,412
70	0.2	80:20	**42,530**	**42,530**	**42,530**	**42,530**	131,095	10,388,804
70	0.2	70:30	**52,179**	**52,179**	**52,179**	**52,179**	162,757	11,137,456
80	0.2	80:20	**43,799**	**43,799**	**43,799**	**43,799**	145,124	12,703,763
80	0.2	70:30	**51,542**	**51,542**	**51,542**	**51,542**	176,548	14,467,021
90	0.18	80:20	**45,684**	**45,684**	**45,684**	**45,684**	108,806	14,363,762
90	0.18	70:30	**61,117**	**61,117**	**61,117**	**61,117**	219,677	17,521,470
100	0.16	80:20	**54,195**	**54,195**	**54,195**	**54,195**	116,340	16,992,283
100	0.16	70:30	**83,499**	**83,499**	**83,499**	**83,499**	290,934	20,802,015
125	0.2	80:20	**67,445**	**67,445**	**67,445**	**67,445**	139,844	25,087,165
125	0.2	70:30	**104,296**	**104,296**	**104,296**	**104,296**	212,568	31,483,422
150	0.2	80:20	**117,291**	**117,291**	**117,291**	**117,291**	179,070	34,293,903
150	0.2	70:30	**181,334**	**181,334**	**181,334**	**181,334**	305,890	43,698,629
175	0.2	80:20	**227,126**	**227,126**	**227,126**	**227,126**	272,432	45,266,830
175	0.2	70:30	**365,401**	**365,401**	**365,401**	**365,401**	459,317	56,044,863

Table 7.11: Median results for unsolvable random problems with one or more agents having no solution to their local problem.

Graph Colouring Problems: Median results for unsolvable 3-colour distributed graph colouring problems with 150 to 200 nodes, 15 to 25 agents and 4.9 to 5.1 degree where one or more agents had no solutions to their complex local problem are presented in table 7.13. *Multi-Hyb-Pen*, *Multi-Hyb-DB*, *Multi-HDCS-Pen* and *Multi-HDCS-DB* will

			Median number of messages					
Num Vars	**% constraint density**	**% intra:inter constraints**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
60	0.2	80:20	703	**69**	177	194	762	33,930
60	0.2	70:30	480	**69**	249	319	3,950	41,712
70	0.18	70:30	418	**54**	114	166	1,266	48,433
80	0.16	70:30	823	**49**	106	129	1,242	55,324
90	0.14	70:30	500	**56**	158	262	1,968	61,541
100	0.13	70:30	674	**49**	129	157	840	68,524
			Median number of NCCCs					
Num Vars	**% constraint density**	**% intra:inter constraints**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
60	0.2	80:20	53,186	**52,648**	62,205	59,641	127,460	7,620,027
60	0.2	70:30	113,114	**83,564**	251,012	252,212	226,011	7,996,729
70	0.18	70:30	102,594	**91,343**	136,748	136,748	192,851	10,569,556
80	0.16	70:30	135,582	**124,409**	174,461	174,461	230,568	13,527,324
90	0.14	70:30	254,904	**238,437**	374,569	372,796	333,709	16,092,489
100	0.13	70:30	330,553	**298,966**	362,227	354,277	347,370	19,929,678

Table 7.12: Median results for unsolvable random problems with all agents having solutions to their local problem but no global solution.

all perform identically in this situation since *SEBJ* detects unsolvability. We found that these algorithms outperformed Multi-ABT and Multi-AWCS substantially on messages. For NCCCs, Multi-ABT was occasionally better but the smaller difference in NCCCs meant that the *Multi-Hyb* and *Multi-HDCS* implementations were better overall. Median results in table 7.14 are for problems where all agents had solutions to their complex local problem but there was no global solution to the problem.

Multi-HDCS-Pen and *Multi-HDCS-DB* suffer from excessive running of local search which continues beyond the point of centralised systematic search finishing until distributed systematic search finishes. This causes a substantial increase in both messages and NCCCs. *Multi-Hyb-Pen* was the most consistent algorithm for graph colouring problems where there was no global solution. There are however 3 problem settings where *Multi-HDCS-DB* performs well in terms of messages and may therefore be a suitable algorithm for these problems.

Meeting Scheduling Problems: Unsolvable meeting scheduling problems with 50-80 meetings, 5 departments (agents), a timeframe of 6 or 7 time units and a constraint density of 0.18 were conducted. The percentage of intra-agent constraints varied between 70% and 90%. Two departments with common meetings have a distance of between 1 and 3 time units. Problems where one or more agents had no solution to their complex

				Median number of messages					
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
150	15	4.9	80:20	**42**	**42**	**42**	**42**	860	6,307
150	15	5.1	80:20	**42**	**42**	**42**	**42**	947	6,456
150	15	4.9	70:30	**50**	**50**	**50**	**50**	2,911	9,356
150	15	5.1	70:30	**48**	**48**	**48**	**48**	1,899	9,474
150	25	4.9	70:30	**72**	**72**	**72**	**72**	1,576	13,728
150	25	5.1	70:30	**68**	**68**	**68**	**68**	1,630	14,031
200	20	4.9	80:20	**57**	**57**	**57**	**57**	1,277	9,163
200	20	5.1	80:20	**58**	**58**	**58**	**58**	1,497	9,195
200	20	4.9	70:30	**66**	**66**	**66**	**66**	2,296	14,107
200	20	5.1	70:30	**64**	**64**	**64**	**64**	1,956	14,680
200	25	4.9	80:20	**68**	**68**	**68**	**68**	1,398	10,195
200	25	5.1	80:20	**66**	**66**	**66**	**66**	1,234	10,321
200	25	4.9	70:30	**79**	**79**	**79**	**79**	1,816	16,277
200	25	5.1	70:30	**76**	**76**	**76**	**76**	1,883	17,021
				Median number of NCCCs					
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
150	15	4.9	80:20	1,525	1,525	1,525	1,525	**1,202**	11,590
150	15	5.1	80:20	1,421	1,421	1,421	1,421	**1,286**	11,924
150	15	4.9	70:30	**2,332**	**2,332**	**2,332**	**2,332**	2,395	15,259
150	15	5.1	70:30	2,114	2,114	2,114	2,114	**1,797**	15,255
150	25	4.9	70:30	**296**	**296**	**296**	**296**	767	11,580
150	25	5.1	70:30	**294**	**294**	**294**	**294**	758	11,725
200	20	4.9	80:20	1,415	1,415	1,415	1,415	**1,304**	11,910
200	20	5.1	80:20	1,717	1,717	1,717	1,717	**1,321**	11,812
200	20	4.9	70:30	2,512	2,512	2,512	2,512	**1,727**	15,300
200	20	5.1	70:30	2,253	2,253	2,253	2,253	**1,656**	15,854
200	25	4.9	80:20	**673**	**673**	**673**	**673**	900	9,807
200	25	5.1	80:20	**644**	**644**	**644**	**644**	845	9,693
200	25	4.9	70:30	**895**	**895**	**895**	**895**	1,053	12,411
200	25	5.1	70:30	**875**	**875**	**875**	**875**	1,095	12,990

Table 7.13: Median results for unsolvable graph colouring problems with one or more agents having no solution to their local problem.

				Median number of messages					
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
150	15	4.9	80:20	2,300	484	**144**	250	1,417	6,620
150	15	5.1	80:20	1,927	676	**187**	311	1,823	6,627
150	15	4.9	70:30	2,346	**387**	388	518	4,019	9,816
150	15	5.1	70:30	2,718	367	**208**	364	3,590	9,942
150	25	4.9	80:20	4,131	446	**48**	261	1,405	13,174
150	25	5.1	80:20	3,879	361	**27**	246	1,205	14,173
150	25	4.9	70:30	3,046	309	**61**	328	3,134	19,866
150	25	5.1	70:30	4,806	331	**48**	333	2,863	22,954
200	20	4.9	80:20	3,677	452	**266**	414	1,464	8,480
200	20	5.1	80:20	3,895	431	**176**	342	1,424	9,015
200	20	4.9	70:30	3,650	**429**	1,324	1,528	4,818	13,206
200	20	5.1	70:30	4,280	**317**	744	952	3,402	13,058
200	25	4.9	80:20	4,740	292	**186**	376	1,429	11,049
200	25	5.1	80:20	4,913	279	**116**	313	1,166	11,577
200	25	4.9	70:30	5,899	361	**354**	627	3,097	15,778
200	25	5.1	70:30	4,752	333	**204**	498	2,495	17,386
				Median number of NCCCs					
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
150	15	4.9	80:20	3,316	4,431	**2,184**	2,275	2,514	23,646
150	15	5.1	80:20	2,719	6,644	**2,166**	2,355	2,981	24,823
150	15	4.9	70:30	5,524	5,332	7,566	7,566	**4,571**	30,175
150	15	5.1	70:30	5,838	5,457	4,250	4,250	**4,150**	30,428
150	25	4.9	80:20	3,017	3,613	**439**	830	1,029	20,399
150	25	5.1	80:20	2,694	3,236	**394**	814	883	21,351
150	25	4.9	70:30	3,738	3,596	**558**	1,339	1,522	26,605
150	25	5.1	70:30	5,266	4,112	**514**	1,155	1,398	30,230
200	20	4.9	80:20	4,175	4,036	3,263	3,263	**2,333**	22,227
200	20	5.1	80:20	3,605	4,100	2,375	2,666	**2,132**	23,200
200	20	4.9	70:30	8,473	6,320	10,130	10,130	**4,201**	28,327
200	20	5.1	70:30	8,037	4,669	7,502	7,502	**3,195**	27,356
200	25	4.9	80:20	3,847	2,017	1,607	1,718	**1,503**	20,176
200	25	5.1	80:20	3,649	1,967	**1,126**	1,399	1,416	20,676
200	25	4.9	70:30	7,140	4,000	3,528	3,532	**2,104**	25,026
200	25	5.1	70:30	6,233	4,224	1,968	2,736	**1,895**	25,263

Table 7.14: Median results for unsolvable graph colouring problems with all agents having at least one solution to their local problem but no global solution.

local problem are presented in table 7.15. Since *SEBJ* detects unsolvability, both implementations of *Multi-Hyb* and *Multi-HDCS* will perform identically. The *Multi-Hyb* and *Multi-HDCS* implementations substantially outperform Multi-ABT and Multi-AWCS for messages and NCCCs. Problems where all agents had solutions to their complex local problem but there was no global solution are presented in table 7.16.

			Median number of messages					
Num Meetings	**Num Times**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
50	7	80:20	**13**	**13**	**13**	**13**	182	1,730
50	7	70:30	**14**	**14**	**14**	**14**	331	2,308
50	6	80:20	**12**	**12**	**12**	**12**	86	1,138
50	6	70:30	**14**	**14**	**14**	**14**	176	1,446
60	7	80:20	**12**	**12**	**12**	**12**	124	1,687
60	7	70:30	**14**	**14**	**14**	**14**	240	2,390
60	6	80:20	**11**	**11**	**11**	**11**	117	1,145
60	6	70:30	**12**	**12**	**12**	**12**	171	1,511
70	7	80:20	**10**	**10**	**10**	**10**	152	1,721
70	7	70:30	**12**	**12**	**12**	**12**	185	2,232
70	6	80:20	**12**	**12**	**12**	**12**	110	1,139
70	6	70:30	**12**	**12**	**12**	**12**	132	1,495
80	7	80:20	**10**	**10**	**10**	**10**	115	1,659
80	7	70:30	**10**	**10**	**10**	**10**	167	2,285
80	6	80:20	**10**	**10**	**10**	**10**	97	1,032
80	6	70:30	**12**	**12**	**12**	**12**	239	1,401
			Median number of NCCCs					
Num Meetings	**Num Times**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
50	7	80:20	**3,051**	**3,051**	**3,051**	**3,051**	10,128	26,687
50	7	70:30	**3,174**	**3,174**	**3,174**	**3,174**	11,474	32,575
50	6	80:20	**2,315**	**2,315**	**2,315**	**2,315**	3,929	15,309
50	6	70:30	**1,916**	**1,916**	**1,916**	**1,916**	5,270	18,386
60	7	80:20	**3,055**	**3,055**	**3,055**	**3,055**	12,044	31,148
60	7	70:30	**3,476**	**3,476**	**3,476**	**3,476**	11,779	41,168
60	6	80:20	**2,211**	**2,211**	**2,211**	**2,211**	5,021	17,618
60	6	70:30	**1,980**	**1,980**	**1,980**	**1,980**	5,638	21,167
70	7	80:20	**3,395**	**3,395**	**3,395**	**3,395**	15,330	34,728
70	7	70:30	**4,343**	**4,343**	**4,343**	**4,343**	14,405	41,546
70	6	80:20	**2,275**	**2,275**	**2,275**	**2,275**	6,152	20,240
70	6	70:30	**2,576**	**2,576**	**2,576**	**2,576**	6,598	24,230
80	7	80:20	**4,637**	**4,637**	**4,637**	**4,637**	14,145	38,468
80	7	70:30	**3,941**	**3,941**	**3,941**	**3,941**	16,856	49,571
80	6	80:20	**2,210**	**2,210**	**2,210**	**2,210**	5,724	20,048
80	6	70:30	**2,890**	**2,890**	**2,890**	**2,890**	10,522	24,303

Table 7.15: Median results for meeting scheduling problems where one or more agents had no solution to their complex local problem.

For problems where all agents had solutions but there was no global solution, *Multi-Hyb-DB* was optimal for most cases for number of messages whilst Multi-ABT was optimal for the remainder and for all problem settings with NCCCs. Particularly, it would appear that the *Multi-HDCS* approach is costly to detect global unsolvability because of the large

			Median number of messages					
Num Meetings	**Num Times**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
50	7	80:20	1,029	2,613	344	**150**	197	4,926
50	7	70:30	686	730	624	517	**507**	5,177
50	6	80:20	661	2,764	222	**91**	107	3,502
50	6	70:30	575	959	204	**119**	151	4,226
60	7	80:20	765	989	320	**125**	132	4,991
60	7	70:30	547	332	284	**210**	306	5,106
60	6	80:20	522	1154	**16**	45	62	3,488
60	6	70:30	450	487	190	**60**	85	4,158
70	7	80:20	554	492	248	89	**62**	4,950
70	7	70:30	517	180	242	**91**	115	5,099
70	6	80:20	446	537	146	**45**	61	3,525
70	6	70:30	326	211	94	**45**	62	4,159
80	7	80:20	577	239	196	83	**61**	4,939
80	7	70:30	430	97	162	**71**	112	5,077
80	6	80:20	358	273	118	**43**	58	3,587
80	6	70:30	318	123	86	**45**	58	4,207
			Median number of NCCCs					
Num Meetings	**Num Times**	**intra: inter**	**Multi -HDCS -Pen**	**Multi -HDCS -DB**	**Multi -Hyb -Pen**	**Multi -Hyb -DB**	**Multi -ABT**	**Multi -AWCS**
50	7	80:20	24,749	14,709	14,345	18,004	**9,965**	76,384
50	7	70:30	36,899	24,253	33,084	38,739	**11,309**	81,379
50	6	80:20	12,884	6,185	5,318	7,223	**4,055**	47,704
50	6	70:30	19,681	7,615	9,480	10,630	**4,930**	56,673
60	7	80:20	23,816	18,745	17,819	19,668	**10,808**	92,333
60	7	70:30	36,205	25,679	33,445	37,229	**13,464**	96,062
60	6	80:20	13,498	6,194	5,860	6,891	**4,219**	57,171
60	6	70:30	17,772	7,441	7,599	9,114	**5,152**	63,441
70	7	80:20	24,275	18,907	17,692	20,279	**9,919**	102,431
70	7	70:30	33,241	25,162	27,350	29,213	**11,554**	109,482
70	6	80:20	14,621	7,032	7,089	8,841	**4,736**	66,217
70	6	70:30	18,238	8,801	8,791	9,461	**6,222**	75,706
80	7	80:20	31,380	24,388	23,671	26,352	**10,977**	118,078
80	7	70:30	38,205	31,133	33,516	36,516	**16,282**	124,501
80	6	80:20	15,678	8,667	8,602	9,664	**5,703**	75,665
80	6	70:30	20,603	10,898	10,999	12,072	**6,216**	84,881

Table 7.16: Median results for meeting scheduling problems where all agents had solutions to their complex local problem but there was no global solution.

costs associated with local search running to provide information to systematic search which are not offset by detecting unsolvability quicker.

Sensor Network Problems: Table 7.17 shows median results for unsolvable sensor networks problems with 5 targets, 25-64 sensors (grids of 5, 6, 7 and 8), k-visibility of 2, k-compatibility of 1, probability of visibility of 0.9 and probability of compatibility of 0.6. The ratio of intra-agent to inter-agent constraints is 15% to 85%. Consequently, all agents had solutions to their complex local problem but there was no global solution.

		Median number of messages					
Num Targets	**Num Sensors**	**Multi HDCS-Pen**	**Multi -HDCS-DB**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**
5	25	1,733	**262**	1,293	730	2,309	5,524
5	36	2,505	**331**	875	560	864	4,657
5	49	2,349	**300**	1,006	531	680	3,346
5	64	2,052	**265**	554	320	381	3,043
6	25	13,910	5,641	2,771	**1,723**	16,002	14,968
6	36	3,550	5,882	14,643	7,069	**3,469**	20,989
6	49	1,345	3,518	176	**136**	320	2,365
6	64	2,930	6,056	1,156	815	**514**	925
7	25	32,035	5,767	7,235	**4,047**	26,807	25,103
7	36	**2,386**	3,486	5,962	2,775	5,429	7,043
7	49	**484**	3,098	721	574	693	3,731
7	64	1,495	7,045	2,041	1,501	**503**	1,155
8	25	52,480	**12,580**	20,488	13,809	112,189	105,417
8	36	5,177	**4,786**	8,333	5,098	24,051	88,030
8	49	1,361	3,131	1,011	**641**	1,068	6,415
8	64	3,966	7,041	6,539	5,295	**932**	2,470
		Median number of NCCCs					
Num Targets	**Num Sensors**	**Multi HDCS-Pen**	**Multi -HDCS-DB**	**Multi -Hyb-Pen**	**Multi -Hyb-DB**	**Multi -ABT**	**Multi -AWCS**
5	25	**13,536**	21,870	22,275	29,873	49,663	92,273
5	36	**11,340**	12,587	15,229	20,391	24,703	77,773
5	49	10,917	**8,393**	22,827	24,551	22,292	64,488
5	64	10,170	**4,887**	9,225	9,787	13,227	61,675
6	25	205,965	421,110	**110,032**	131,431	198,139	225,797
6	36	**17,568**	194,603	821,636	821,633	57,489	288,281
6	49	8,123	**2,712**	3,037	3,364	9,802	33,164
6	64	17,136	126,000	37,684	38,626	12,827	**11,363**
7	25	381,672	624,456	331,460	431,012	**290,947**	347,372
7	36	**14,283**	50,121	65,204	55,508	76,948	78,664
7	49	2,813	**2,787**	9,608	11,516	16,481	44,210
7	64	15,219	189,248	30,609	39,313	11,938	**11,393**
8	25	1,782,819	1,814,238	1,556,956	2,071,355	**990,653**	1,321,611
8	36	**62,316**	232,983	153,330	226,993	299,894	935,045
8	49	7,794	**2,921**	19,284	20,455	25,350	57,949
8	64	**13,968**	50,949	337,895	379,813	18,345	25,679

Table 7.17: Median results on unsolvable Grid-based Sensor Network problems.

Multi-HDCS-DB performs well on problems with 5 targets for both number of messages and NCCCs. In addition, it also performs well on some problem combinations with a higher number of targets. Multi-ABT, Multi-AWCS, *Multi-HDCS-Pen* and Multi-Hyb-DB all perform well on different problem combinations for number of messages and NCCCs.

7.5 Comparing Multi-HDCS and Multi-Hyb

Both *Multi-HDCS* and *Multi-Hyb* (see section 6) use one centralised systematic search per agent, one distributed local search and one distributed systematic search. However, their overall approaches are substantially different as follows: (i) In *Multi-HDCS* all three types of searches run concurrently whereas in *Multi-Hyb* a two-phase strategy is used; (ii) In *Multi-HDCS*, the knowledge discovered during the distributed local search is regularly passed to the distributed systematic search; (iii) *Multi-Hyb* uses a fixed-order distributed systematic search whereas *Multi-HDCS* dynamically orders its agents in its distributed systematic search; (iv) *Multi-Hyb* adds solutions dynamically only to distributed local search whilst solutions are added dynamically to distributed local search and distributed systematic search in *Multi-HDCS*; (v) the distributed local search and distributed systematic search use complex variables in *Multi-Hyb* (i.e. one variable per agent containing all possible solutions for that agent) whilst only the distributed systematic search uses complex variables in *Multi-HDCS*. *Multi-HDCS* uses distributed local search for coarse-grained DisCSP algorithms so that the distributed local search has agents consisting of the number of variables which are externally relevant (i.e. have inter-agent constraints) for that agent. This was found to reduce the number of constraint checks over distributed local search with complex variables as there were less constraint checks performed when choosing a new value as the potential domain for the variable was much reduced. This wasn't the case for distributed systematic search and so this still uses complex variables.

In terms of our problem areas, we have shown that *Multi-HDCS* is primarily effective at reducing NCCCs for randomly generated and sensor network problems. In addition, it can frequently also reduce the number of messages. Our graph colouring problems and scheduling problems tend to have symmetrical solutions (i.e. solutions spread equally over a large search space) in which case *Multi-Hyb* is the more effective algorithm. *Multi-Hyb* can find these solutions quickly without the additional overhead of concurrent searches which *Multi-HDCS* has.

7.6 Contributions

The following contributions have been made:

1. The *Multi-HDCS* approach which finds the externally relevant solutions for each agent's complex local problem whilst participating in a distributed local search and a distributed systematic search to find a global solution. The distributed local search periodically shares knowledge with the distributed systematic search.

2. Two implementations of the *Multi-HDCS* approach: *Multi-HDCS-Pen* using the penalty-on-values local search strategy and *Multi-HDCS-DB* using the breakout local search strategy.

3. *InterDisPeL* which revises the Multi-DisPeL approach specifically for considering inter-agent constraints and uses only solutions dynamically supplied by centralised systematic searches.

4. *InterDisBO-wd* which revises the DisBO-wd approach for inter-agent constraints and only considering solutions supplied dynamically by centralised systematic searches.

5. *InterPODS*, a dynamically ordered systematic search algorithm using complex variables which dynamically receives solutions from centralised systematic searches and knowledge from distributed local search.

7.7 Summary

Multi-HDCS is a new hybrid approach for solving DisCSPs with complex local problems where the problem solving is carried out by concurrent cooperative searches: (i) a set of centralised systematic searches (one per agent) finds all non-interchangeable solutions to each agent's local problem; (ii) a distributed local search attempts to solve the inter-agent constraints using variable-value combinations approved by the centralised systematic searches. It also identifies local problems which are difficult to solve and passes this information to a distributed systematic search (see below); (iii) a distributed systematic search attempts to find a solution satisfying the inter-agent constraints using only variable-value combinations approved by centralised systematic searches whilst dynamically prioritising agents acording to the level of difficulty of their local problems assigned by the distributed local search.

We have presented two implementations of our approach: *Multi-HDCS-Pen* and *Multi-HDCS-DB*. These approaches differ mainly in the algorithm used for distributed local search: *Multi-HDCS-Pen* uses a penalty-based algorithm (*InterDisPeL*) whereas *Multi-HDCS-DB* uses a breakout-based (i.e. weights on constraints) algorithm (*Inter-DisBO-wd*). Both algorithms use *SEBJ* to solve the agent's local problem and *InterPODS* as the distributed systematic search algorithm.

Substantial empirical results on several problem classes demonstrate that the *Multi-HDCS* approach (particularly in the *Multi-HDCS-DB* implementation) is generally competitive when compared to leading DisCSPs with complex local problems algorithms on both solvable problems and unsolvable problems.

Chapter 8

Conclusions and Future Work

This thesis has researched and developed hybrid algorithms for Distributed Constraint Satisfaction and their applicability on a number of problem classes. A number of new contributions to the Distributed Constraint Satisfaction community in the field of hybrid algorithms have been made through this thesis. This chapter outlines the contributions of this thesis and possible avenues for future work.

8.1 Contributions

A number of contributions have been made in this thesis (table 8.1 summarises the contributions):

1. In chapter 5, the *DisHyb* approach was presented. This novel hybrid approach for DisCSPs with one variable per agent runs a distributed local search algorithm for a bounded number of cycles. This local search algorithm learns important knowledge about difficult variables and the best values for those variables. If the distributed local search algorithm fails to solve the problem, a distributed systematic search runs guided by the knowledge learnt by distributed local search. Two implementations of this approach have been presented in this thesis: *PenDHyb* and *DBHyb*. We have derived a formula to predict the best number of cycles for distributed local search and also shown that much longer executions of local search may be beneficial for harder solvable problems. We have shown that *PenDHyb* and *DBHyb* outperform

systematic search on three problem classes (randomly generated problems, graph colouring problems and meeting scheduling problems).

2. In chapter 6, the *Multi-Hyb* approach was presented. This is a novel two-phase hybrid approach for DisCSPs with complex local problems. In the first phase, a centralised systematic search algorithm runs concurrently for each agent to find solutions for that agent's complex local problem. Concurrently, a distributed local search algorithm runs which only considers constraints between agents and attempts to combine solutions to an agent's complex local problem (i.e. partial solutions to the global problem) in order to find a global solution to the problem. This local search also learns about difficult variable and value combinations. If all solutions to an agent's complex local problem are found before distributed local search finds a solution, a distributed systematic search runs which finds a solution or detects unsolvability. Two implementations of this approach have been presented in this thesis: *Multi-Hyb-Pen* and *Multi-Hyb-DB*. We have shown that *Multi-Hyb-Pen* and *Multi-Hyb-DB* often outperform leading algorithms for DisCSPs with complex local problems (Multi-ABT, Multi-AWCS, Multi-DisPeL, DisBO-wd) on randomly generated problems, graph colouring problems, meeting scheduling problems (for number of messages) and sensor network problems.

3. In chapter 7, the *Multi-HDCS* approach was presented. This is a novel hybrid approach for DisCSPs with complex local problems. This algorithm also runs concurrently a centralised systematic search for each agent to detect all solutions to an agent's complex local problem. In addition, two concurrent searches are run: (i) a distributed local search algorithm which learns about difficult variables and values in addition to attempting to finding a global solution to the problem; (ii) a distributed systematic search which is guided by the distributed local search through synchronisation of information and finds a global solution to the problem or detects unsolvability. Two implementations of this approach have been presented in this thesis: *Multi-HDCS-Pen* and *Multi-HDCS-DB*. We have shown that *Multi-HDCS* is an important revision to *Multi-Hyb* which outperforms *Multi-Hyb* on randomly

generated and sensor network problems.

Approach	Number of Phases	Algorithms	Variables	Domains	Constraints Considered	Knowledge Exchanged
DisHyb	2	PenDHyb and DB-Hyb	Single variable per agent	Static	Inter-agent constraints (no intra-agent constraints in fine-grained DisCSPs)	Distributed local search exchanges difficult variables and best values information with distributed systematic search.
Multi-Hyb	2	Multi-Hyb-Pen and Multi-Hyb-DB	Centralised systematic searches uses all variables in an agent, distributed local search and distributed systematic search use complex variables	Static for centralised systematic searches and distributed systematic search. Dynamic for distributed local search	Intra-agent constraints considered by centralised systematic searches. Inter-agent constraints considered by distributed local search and distributed systematic search	Centralised systematic searches pass solutions to complex local problems to distributed local search and distributed systematic search. Distributed local search passes knowledge of difficult variables and best values from distributed local search to distributed systematic search.
Multi-HDCS	1	Multi-HDCS-Pen and Multi-HDCS-DB	Centralised systematic searches uses all variables in an agent, distributed local search uses all externally relevant variables in an agent and distributed systematic search use complex variables	Static for centralised systematic searches. Dynamic for distributed local search and distributed systematic search	Intra-agent constraints considered by centralised systematic searches. Inter-agent constraints considered by distributed local search and distributed systematic search	Centralised systematic searches pass solutions to complex local problems to distributed local search and distributed systematic search. Distributed local search passes knowledge of difficult variables regularly to distributed systematic search.

Table 8.1: Overview of Thesis Contributions.

8.2 Future Work

There are a number of possible avenues for future work which could extend the work presented in this thesis.

8.2.1 Alternative Implementations of DisHyb

Other implementations of the *DisHyb* framework in chapter 5 could be considered. Specifically, the Distributed Stochastic Algorithm (DSA) [100] uses a probabilistic strategy to escape local minima which allows variables to change values if they reduce the number of constraint violations or with a certain probability if they do not increase the number of constraint violations. It would be worth exploring if the difficulty of the variable could be determined by the amount of times a variable changes its value.

Alternatively, asynchronous systematic search algorithms (e.g. ABT [97]) could be used as the distributed systematic search algorithm. There are very few asynchronous local search algorithms so this part of the framework may have to remain synchronous. Note that an algorithm such as AWCS would not be appropriate in the framework as it would reorder the agents according to its own schema thereby removing the benefit of learning knowledge from local search.

8.2.2 Different Centralised Systematic Searches in Multi-Hyb/Multi-HDCS

Since the concurrent centralised systematic searches only require to find all non-interchangeable solutions for an agent's complex local problem, there is nothing to preclude the use of different search strategies for different agents. Therefore, an agent could detect certain properties about its own local problem (e.g. highly connected variables, low constraint density) to choose the best search algorithm and heuristic for that particular type of problem whilst maintaining the overall *Multi-Hyb* or *Multi-HDCS* framework.

8.2.3 Running Distributed Local Search after Centralised Systematic Searches in Multi-Hyb

In *Multi-Hyb*, the distributed local search could be left running for a number of cycles after all concurrent centralised systematic searches finish before starting the distributed systematic search. Initial experiments in this area were outlined in the *Multi-Hyb* variants section in chapter 6 which concluded that the effort was not worthwhile but for very large and difficult problems, it may be appropriate as has been evidenced for the longer executions of distributed local search in *DisHyb* (see section 5.5.2).

8.2.4 Bi-directional Feedback in Multi-HDCS

In our *Multi-HDCS* framework, distributed local search synchronises information about difficult variables to the distributed systematic search on a regular basis. It may be beneficial to extend this so that the distributed systematic search can give feedback to the distributed local search algorithm on the current parts of the search space it is exploring so that distributed local search can provide more targeted information. There will be a need to allow distributed local search to explore other areas of the search space periodically to ensure that the algorithms can escape from a part of the search space which does not contain solutions.

8.2.5 Using Multi-Hyb and Multi-HDCS for Optimisation

It is possible to adapt *Multi-Hyb* and *Multi-HDCS* for Distributed Constraint Optimisation problems. These problems contain a cost function which allows the most desirable solution to be found from a number of possible solutions. There would be a need to modify the algorithms in *Multi-Hyb* and *Multi-HDCS* to take account of this cost function.

8.2.6 Heterogeneous and Dynamic DisCSPs

The vast majority of experiments concerning DisCSPs with complex local problems assume that the number of variables in each agent is identical. However, many realistic scenarios have different number of variables for different agents. Consequently, there is a need to explore how our approaches and other approaches for DisCSPs with complex local problems deal with this scenario.

In addition, problems may change during execution of the solution. At the moment, our approaches would have to be re-run in order to cope with these changes. It would be interesting to explore if changes could be made to improve our approaches and remove this requirement to re-run.

8.3 Summary

This thesis has examined hybrid algorithms combining backtracking and local search properties for distributed constraint satisfaction. The primary aim of this thesis (as stated in section 1.1) has been to speed-up distributed problem solving through using local search as a learning tool which can be used to guide backtracking, particularly for naturally distributed problems. Our research objectives were therefore as follows:

1. Investigate techniques for making local search complete.
2. Making systematic search faster through the use of local search information.
3. Take advantage of agent idle time in order to carry out additional computation and thereby minimise overall problem cost.

We have presented new three hybrid approaches which meet these objectives: *DisHyb*, *Multi-Hyb* and *Multi-HDCS*.

DisHyb is a successful hybrid algorithm for fine-grained DisCSPs which uses knowledge learnt during the local search phase to guide the backtracking phase of the algorithm. Therefore, it makes a local search algorithm complete and makes systematic search faster through the use of local search information.

Multi-Hyb extends this approach for DisCSPs with complex local problems through using knowledge learnt from centralised systematic search and distributed local search to guide a distributed systematic search algorithm. This approach also makes distributed local search complete through the combination with distributed systematic search and makes systematic search faster through the use of local search information. Agent idle time is used to participate in a distributed local search.

Multi-HDCS further extends *Multi-Hyb* by introducing concurrent distributed local search and distributed systematic search algorithms with the distributed local search periodically sharing knowledge with the distributed systematic search. This approach meets all three objectives and particularly improves the use of agent idle time through participation in both a distributed systematic search and distributed local search.

Each of our approaches has been implemented with the breakout local search strategy and the penalty-based local search strategy. These algorithms have been shown to outperform the leading systematic and local search DisCSP algorithms on a number of problem classes (randomly generated, graph colouring, meeting scheduling and sensor networks).

In summary, three hybrid approaches for DisCSPs have been presented, one for fine-grained DisCSPs and two for DisCSPs with complex local problems. Two implementations of each of the approaches have been described and an extensive empirical evaluation on several problem classes has demonstrated the effectiveness of our approaches for these types of problems.

Bibliography

[1] S. Anand, W. N. Chin, and S. C. Khoo. A Lazy Divide & Conquer Approach to Constraint Solving. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '02)*, pages 91–98, 2002.

[2] Muhammad Arshad and Marius C. Silaghi. Distributed Simulated Annealing. In *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, volume 112 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2004.

[3] Fahiem Bacchus and Adam Grove. On The Forward Checking Algorithm. In Ugo Montanari and Francesca Rossi, editors, *Proceedings of the First International Conference on Constraint Programming*, pages 292–309. Springer-Verlag, 1995.

[4] Fahiem Bacchus and Peter van Beek. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 98)*, pages 311–318, 1998.

[5] Fahiem Bacchus and Paul van Run. Dynamic Variable Ordering In CSPs. In Ugo Montanari and Francesca Rossi, editors, *Proceedings of the First International Conference on Constraint Programming*, pages 258–275. Springer-Verlag, 1995.

[6] Nicolas Barnier and Pascal Brisset. Combine & Conquer: Genetic Algorithm and CP for Optimization. In *Poster at the Fourth Conference on Principles and Practice of Constraint Programming*, page 436, 1998.

[7] R. Bartak. Constraint Programming - What is Behind? In J. Figwer, editor,

Proceedings of the Workshop on Constraint Programming in Decision and Control, pages 7–15, Gliwice, June 1999.

[8] Muhammed Basharu. *Modifying Landscapes with Penalties in Iterative Improvement for Solving Distributed Constraint Satisfaction Problems.* PhD thesis, School of Computing, The Robert Gordon University, Aberdeen, April 2006.

[9] Christian Bessiere. Non-binary Constraints. In *Proceedings of Principles and Practice of Constraint Programming, CP 99, Invited Lecture*, pages 24–27, 1999.

[10] Christian Bessière, Arnold Maestre, Ismel Brito, and Pedro Meseguer. Asynchronous Backtracking without Adding Links: A New Member in the ABT Family. *Artificial Intelligence*, 161(1–2):7–24, 2005.

[11] Ismel Brito. *Distributed Constraint Satisfaction.* PhD thesis, Institut d'Investigacio en Intel.ligencia Artificial Consejo Superior de Investigaciones Cientificas, 2007.

[12] Ismel Brito and Pedro Meseguer. Synchronous, Asynchronous and Hybrid Algorithms for DisCSPs. In P. Modi, editor, *Proceedings of the 5th International Workshop on Distributed Constraint Reasoning (DCR-04)*, pages 80–94, Toronto, Canada, September 2004.

[13] David Burke. *Exploiting Problem Structure in Distributed Constraint Optimization with Complex Local Problems.* PhD thesis, National University of Ireland, Cork, 2008.

[14] Y. Caseau and F. Laburthe. Heuristics for Large Constrained Vehicle Routing Problems. *Journal of Heuristics*, 5:281–303, 1999.

[15] Yves Caseau, Glenn Silverstein, and Francois Laburthe. Learning Hybrid Algorithms for Vehicle Routing Problems. *Theory and Practice of Logic Programming*, 1(6):779–806, November 2001.

[16] Carlos Cotta, Ivan Dotu, Antonio J. Fernandez, and Pascal van Hentenryck. Local

Search-based Hybrid Algorithms for Finding Golomb Rulers. *Constraints*, 12(3):263–291, 2007.

[17] J. Crawford. Solving Satisfiability Problems Using a Combination of Systematic and Local Search. Second DIMACS Challenge: Cliques, Coloring, and Satisfiability, October 1993.

[18] James M. Crawford and Andrew B. Baker. Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1092–1097, Seattle, Washington, USA, July/August 1994. AAAI Press/MIT Press.

[19] Philippe David. A Constraint-Based Approach for Examination Timetabling Using Local Repair Techniques. *Lecture Notes in Computer Science*, 1408:169–186, August 1998.

[20] Bruno DeBacker, Vincent Furnon, Philip Kilby, Patrick Prosser, and Paul Shaw. Local Search in Constraint Programming: Application to the Vehicle Routing Problem. In *Proceedings of Workshop on Industrial Constraint-Directed Scheduling (1997), Constraint Programming 97*, 1997. Constraint Programming 97,.

[21] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, 2003.

[22] Carlos Eisenberg. *Distributed Constraint Satisfaction for Coordinating and Integrating a Large-Scale Heterogeneous Enterprise*. PhD thesis, Ecole Polytechnique Federale De Lausanne, 2003.

[23] R. Ezzahir, C. Bessiere, E. H. Bouyakhf, and M. Belaissaoui. Asynchronous Backtracking with Compilation Formulation for handling complex local problems. *ICGST International Journal on Artificial Intelligence and Machine Learning*, 8:45–53, 2008.

[24] Marko Fabiunke. Parallel Distributed Constraint Satisfaction. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 99)*, pages 1585–1591, Las Vegas, June 1999.

[25] Boi Faltings. *Handbook of Constraint Programming*, chapter 20, pages 699–729. Elsevier, 2006.

[26] Boi Faltings and Santiago Macho-Gonzalez. Open Constraint Programming. *Artificial Intelligence*, 161:181–208, 2005.

[27] Hai Fang and Wheeler Ruml. Complete Local Search for Propositional Satisfiability. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, pages 161–166, July 2004.

[28] Cesar Fernandez, Ramon Bejar, Bhaskar Krishnamachari, and Carla Gomes. Communication and Computation in Distributed CSP Algorithms. In *CP '02: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 664–679, Itacha, NY, USA, July 2002. Springer-Verlag.

[29] Stephen Fitzpatrick and Lambert Meertens. An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs. In Kathleen Steinhofel, editor, *1st Syposium on Stochastic Algorithms*, volume 2264 of *Lecture Notes in Computer Science*, pages 49–64, Berlin, December 2001. Springer-Verlag.

[30] Eugene C. Freuder, Rina Dechter, Bart Ginsberg, Bart Selman, and Edward P. K. Tsang. Systematic Versus Stochastic Constraint Satisfaction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 95)*, volume 2, pages 2027–2032, San Mateo, CA, 1995. Morgan Kaufmann.

[31] Eugene C. Freuder and Paul D. Hubbe. Extracting Constraint Satisfaction Subproblems. In *Proceedings from the 14th International Joint Conference on Artificial Intelligence*, pages 548–555, 1995.

[32] Daniel Frost and Rina Dechter. In Search of the Best Constraint Satisfaction Search. In *National Conference on Artificial Intelligence (AAAI '94)*, pages 301–306, 1994.

[33] John Gary Gaschnig. Performance Measurement and Analysis of Certain Search Algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, Pittsburgh, PA, 1979.

[34] Matthew L. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[35] Matthew L. Ginsberg and David A. McAllester. GSAT and Dynamic Backtracking. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, pages 226–237, Bonn, Germany, May 24-27 1994. Morgan Kaufmann.

[36] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[37] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 431–437, Madison, Wisconsin, July 1998. AAAI Press.

[38] Eric Gregoire, Bertrand Mazure, and Cedric Piette. Local-search Extraction of MUSes. *Constraints*, 12(3):325–344, 2007.

[39] Youssef Hamadi. Optimal Distributed Arc-Consistency. In *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming*, pages 219–233, 1999.

[40] Youssef Hamadi. Conflicting Agents in Distributed Search. *International Journal on Artificial Intelligence Tools*, 14(3-4):459–476, 2005.

[41] Youssef Hamadi, Christian Bessière, and Joel Quinqueton. Backtracking in Distributed Constraint Networks. In H. Prade, editor, *13th European Conference on Artificial Intelligence (ECAI '98)*, pages 219–223, Chichester, August 1998. John Wiley and Sons.

[42] Pierre Hansen and Nenad Mladenovic. Variable Neighborhood Search. In P. Pardalos and M. Resende, editors, *Handbook of Applied Optimization*, pages 221–234. Oxford University Press, New York, 2002.

[43] Jin-Kao Hao and Raphael Dorne. Empirical Studies of Heuristic Local Search for Constraint Solving. In *Proceedings of Principles and Practice of Constraint Programming (CP-96)*, number 1118 in Lecture Notes in Computer Science, pages 194–208, Cambridge, MA, USA, 1996.

[44] Peter Harvey, Chee Fon Chang, and Aditya Ghose. Support-based Distributed Search: A new approach for multiagent constraint processing. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 377–383, Hakodate, Japan, 2006. ACM Press.

[45] Katsutoshi Hirayama and Makoto Yokoo. The Distributed Breakout Algorithms. *Artificial Intelligence*, 161(1–2):89–115, January 2005.

[46] Katsutoshi Hirayama, Makoto Yokoo, and Katia Sycara. An Easy-Hard-Easy Cost Profile in Distributed Constraint Satisfaction. *Transactions of Information Processing Society of Japan*, 45(9):2217–2225, September 2004.

[47] Tad Hogg and Colin P. Williams. Solving the Really Hard Problems with Cooperative Search. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, pages 231–236, Washington, DC, July 1993. AAAI Press.

[48] N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, 2002.

[49] Olli Kamarainen and Hani El Sakkout. Local Probing Applied to Scheduling. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - Proceedings of the 8th International Conference on Constraint Programming*, pages 155–171, Ithaca, NY, USA, 2002.

[50] Sankalp Khanna, Abdul Sattar, David Hansen, and Bela Stantic. An Efficient Algorithm for Solving Dynamic Complex Local DCOP Problems. In *In Proceedings of 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 339–346, 2009.

[51] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 13 May 1983.

[52] J Lever. A Local Search/Constraint Propagation Hybrid for a Network Routing Problem. *International Journal on Artificial Intelligence Tools*, 14(1-2):43–60, 2005.

[53] Arnold Maestre and Christian Bessiere. Improving Asynchronous Backtracking for Dealing with Complex Local Problems. In *Proceedings of ECAI-04*, pages 206–210, Valencia, Spain, 2004.

[54] Roger Mailler and Victor Lesser. Using Cooperative Mediation to Solve Distributed Constraint Satisfaction Problems. In *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, volume 1, pages 446–453, New York, 2004. IEEE Computer Society.

[55] Bertrand Mazure, Lakhdar Sais, and Eric Gregoire. Boosting Complete Techniques Thanks to Local Search Methods. *Annals of Mathematics and Artificial Intelligence*, 22(3–4):319–331, 1998.

[56] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing Performance of Distributed Constraints Processing Algorithms. In *Proceedings of the AAMAS-2002 Workshop on Distributed Constraint Reasoning*, pages 86–93, Bologna, July 2002.

[57] A. Meisels and R. Zivan. Asynchronous Forward-Checking for DisCSPs. *Constraints*, 12(1):131–150, 2007.

[58] I. Miguel and Q. Shen. Solution Techniques for Constraint Satisfaction Problems: Advanced Approaches. *Artificial Intelligence Review*, 15(4):269–293, June 2001.

[59] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58(1–3):161–205, 1992.

[60] D. Mitra and H. rae Kim. A New Approach for Heterogeneous Hybridization of Constraint Satisfaction Search Algorithms. In *Proceedings of the Seventeenth In-*

ternational Florida Artificial Intelligence Research Society Conference. AAAI Press, 2004.

[61] Pierre Monier, Sylvain Piechowiak, and Rene Mandiau. A complete algorithm for DisCSP: Distributed Backtracking with Sessions (DBS). In *Proceedings of Second International Workshop on Optimisation in Multi-Agent Systems*, 2009.

[62] Paul Morris. The Breakout Method for Escaping from Local Minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 40–45, 1993.

[63] Alexander Nareyek, Stephen F. Smith, and Christian M. Ohler. Integration of a Refinement Solver and a Local-Search Solver. Technical Report TR-RI-04-33, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, August 2004.

[64] Viet Nguyen, Djamila Sam-Haroud, and Boi Faltings. Dynamic Distributed BackJumping. In *Proceedings of the 5th Workshop on Distributed Constraints Reasoning (DCR-04)*, pages 51–65, Toronto, Canada, September 2004.

[65] Eugeniusz Nowicki and Czeslaw Smutnicki. A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science*, 42(6):797–813, June 1996.

[66] Edgar M. Palmer. *Graphical Evolution: An Introduction to the Theory of Random Graphs*. John Wiley and Sons, Inc., 1985.

[67] Gilles Pesant and Michel Gendreau. A Constraint Programming Framework for Local Search Methods. *Journal of Heuristics*, 5(3):255–279, 1999.

[68] Adrian Petcu and Boi Faltings. A Value Ordering Heuristic for Local Search in Distributed Resource Allocation. In B. Faltings, A. Petcu, F. Rossi, and F. Fages, editors, *LNAI 3419 - CSCLP04*, pages 86–97. Springer Verlag, Lausanne, Switzerland, February 2004.

[69] Adrian Petcu and Boi Faltings. A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, Edinburgh, Scotland, August 2005.

[70] Adrian Petcu and Boi Faltings. A Hybrid of Inference and Local Search for Distributed Combinatorial Optimization. In *Proceedings of 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 342–348. IEEE Computer Society, 2007.

[71] S. Prestwich. Combining the Scalability of Local Search with the Pruning Techniques of Systematic Search. *Annals of Operations Research.*, 115(1-4):51–72, September 2002.

[72] Steve Prestwich. A Hybrid Search Architecture Applied to Hard Random 3-SAT and Low-Autocorrelation Binary Sequences. In *The Sixth International Conference on Principles and Practice of Constraint Programming (CP-2000)*, pages 337–352. Springer-Verlag, 2000.

[73] Steve Prestwich. Local Search and Backtracking vs Non-Systematic Backtracking. In *AAAI 2001 Fall Symposium on Using Uncertainty within Computation*, pages 109–115. AAAI Press, 2001.

[74] Patrick Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9(3):268–299, 1993.

[75] E. Thomas Richards and Barry Richards. Non-systematic Search and Learning: An Empirical Study. *Lecture Notes in Computer Science*, 1520:370–384, October 1998.

[76] Georg Ringwelski. An Arc-Consistency Algorithm for Dynamic and Distributed Constraint Satisfaction Problems. *Artificial Intelligence Review*, 24(3-4):431–454, November 2005.

[77] Georg Ringwelski and Youssef Hamadi. Boosting Distributed Constraint Satisfaction. In Peter van Beek, editor, *Proceedings of the 11th International Conference Principles and Practice of Constraint Programming û CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 549–562. Springer, September 2005.

[78] Nico Roos, Yongping Ran, and Jaap van den Herik. Combining Local Search and

Constraint Propagation to Find a Minimal Change Solution for a Dynamic CSP. In *Artificial Intelligence: Methodology, Systems, Applications*, pages 272–282, 2000.

[79] Hani El Sakkout and Mark Wallace. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 5(4):359–388, 2000.

[80] Andrea Schaerf. Combining Local Search and Look-Ahead for Scheduling and Constraint Satisfaction Problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 1254–1259. Morgan-Kaufmann, 1997.

[81] Uri Shapen and Amnon Meisels. Cooperative Dynamic Multi-CBJ Search for DisCSPs. In *Proceedings of the 9th International Workshop on Distributed Constraint Reasoning*, 2007.

[82] Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. *Springer-Verlag Lecture Notes in Computer Science*, 1520:417–431, 1998.

[83] Marius-Calin Silaghi, Djamila Sam-Haroud, and Boi Faltings. Asynchronous Search with Aggregations. In *Proceedings of AAAI/IAAI 2000*, pages 917–922, Austin, TX, 2000.

[84] Marius-Calin Silaghi, Djamila Sam-Haroud, and Boi Faltings. Consistency Maintenance for ABT. In *Proceedings 7th National Conference on Principles and Practice of Constraint Programming, CP'01*, pages 271–285, Paphos, Cyprus, 2001.

[85] Marius-Calin Silaghi, Djamila Sam-Haroud, and Boi Faltings. Hybridizing ABT and AWC into a polynomial space, complete protocol with reordering. Technical Report EPFL-TR-01/364, Swiss Federal Institute of Technology Lausanne, Swiss Federal Institute of Technology Lausanne, May 2001.

[86] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed Constrained Heuristic Search. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1446–1461, 1991.

[87] Peter van Beek. *Handbook of Constraint Programming*, chapter 4, pages 85–134. Elsevier, 2006.

[88] Gerard Verfaillie and Thomas Schiex. Solution Reuse in Dynamic Constraint Satisfaction Problems. In *National Conference on Artificial Intelligence (AAAI '94)*, pages 307–312, 1994.

[89] Chris Voudouris and Edward Tsang. Guided Local Search. Technical Report CSM-247, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, August 1995.

[90] Toby Walsh. Search on High Degree Graphs. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 266–271. Morgan Kaufmann, August 2001 2001.

[91] M. Yokoo. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer, 2000.

[92] Makoto Yokoo. Weak-Commitment Search for Solving Constraint Satisfaction Problems. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94);Vol. 1*, pages 313–318, Seattle, WA, USA, July 31 - August 4 1994. AAAI Press, 1994.

[93] Makoto Yokoo. Asynchronous Weak-Commitment Search for Solving Distributed Constraint Satisfaction Problems. In Ugo Montanari and Francesca Rossi, editors, *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP-95)*, volume 976 of *Lecture Notes in Computer Science*, pages 88–102. Springer, 1995.

[94] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.

[95] Makoto Yokoo and Katsutoshi Hirayama. Distributed Breakout Algorithm for Solving Distributed Constraint Satisfaction Problems. In M. Tokoro, editor, *Second In-*

ternational Conference on Multiagent Systems (ICMAS-96), pages 401–408, 'Kyoto, Japan, December 1996.

[96] Makoto Yokoo and Katsutoshi Hirayama. Distributed Constraint Satisfaction Algorithm for Complex Local Problems. In *ICMAS*, pages 372–379, 1998.

[97] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for Distributed Constraint Satisfaction: A Review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.

[98] Masazumi Yoshikawa, Kazuya Kaneko, Toru Yamanouchi, and Masanobu Watanabe. A Constraint-Based High School Scheduling System. *IEEE Expert: Intelligent Systems and Their Applications*, 11(1):63–72, February 1996.

[99] Jian Zhang and Hantao Zhang. Combining Local Search and Backtracking Techniques for Constraint Satisfaction. In *Proceedings of the 13th AAAI/8th IAAI, Vol. 1*, pages 369–374, August 1996.

[100] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1–2):55–87, January 2005.

[101] R. Zivan and A. Meisels. Message delay and DisCSP search algorithms. In *Proc. 5th workshop on Distributed Constraints Reasoning, DCR-04*, Toronto, 2004.

[102] Roie Zivan and Amnon Meisels. Synchronous vs Asynchronous search on DisCSPs. In *Proceedings of the First European Workshop on Multi-Agent Systems (EUMA)*, Oxford, December 2003.

[103] Roie Zivan and Amnon Meisels. Concurrent Dynamic Backtracking for Distributed CSPs. In *CP*, pages 782–787, 2004.

[104] Roie Zivan and Amnon Meisels. Dynamic Ordering for Asynchronous Backtracking on DisCSPs. *Constraints*, 11(2-3):179–197, 2006.

[105] Roie Zivan, Moshe Zazone, and Amnon Meisels. Min-Domain Ordering for Asynchronous Backtracking. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume Lecture Notes in Computer Science, pages 758–772. Springer-Verlag, 2007.

Published Papers

- David Lee, Ines Arana, Hatem Ahriz and Kit-Ying Hui, 2008. A Hybrid Approach to Distributed Constraint Satisfaction. In: Danail Dochev, Paolo Traverso and Marco Pistore, ed. *Artificial Intelligence: Methodology, Systems and Applications. 13th International Conference, AIMSA 2008 Varna, Bulgaria, September 4-6, 2008 Proceedings.* pages 375-379. 4th-6th September 2008. Varna, Bulgaria.

- David Lee, Ines Arana, Hatem Ahriz and Kit-Ying Hui, 2009. A Hybrid Approach to Solving Coarse-grained DisCSPs. In: *Proceedings of the Eighth International Conference on Autonomous Agents and Multi Agent Systems (AAMAS 09)* pages 1235-1236. 10th-15th May 2009. Budapest, Hungary.

- David Lee, Ines Arana, Hatem Ahriz and Kit-Ying Hui, 2009. Multi-Hyb: A Hybrid Algorithm for Solving DisCSPs with Complex Local Problems. In: *Proceedings of 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2009)* pages 379-382. 15th-18th September 2009. Milan, Italy.

Glossary of Terms

Constraint Satisfaction Problem	A Constraint Satisfaction Problem consists of variables, domains and constraints. A formal definition is given in section 3.1.
Complex Variables	A single complex variable represents all variables in a complex local problem. The domain is all possible solutions for that complex local problem.
Constraint	An expression between one or more variables which restricts the values that those variables can take simultaneously.
Domain	A set of values that a variable can take.
Distributed Constraint Satisfaction	A Distributed Constraint Satisfaction problem consists of variables, domains, constraints and agents which represent the variables in the problem. A formal definition is given in section 2.2.
DisCSP with Complex Local Problems	A Distributed Constraint Satisfaction problem where each agent represents a CSP containing more than one variable.
Externally Relevant Variables	Those variables in a complex local problem which have inter-agent constraints.
Fine-grained DisCSP	A Distributed Constraint Satisfaction problem where each agent has only one variable per agent.
Intra-agent constraint	A constraint between two or more variables where all variables belong to the same agent.
Inter-agent constraint	A constraint between two or more variables where at least two of the variables belong to different agents.

Interchangeable solutions	Two solutions to a complex local problem are said to be interchangeable if all values for the externally relevant variables are identical.
Local Optima	A neighbourhood is said to be in local optima if there are no changes to a single variable which can reduce the number of constraint violations.
Neighbour	Any variable which shares a constraint with another variable is said to be neighbours with that variable.
Neighbourhood	All of the variables sharing a constraint with a particular variable is said to form that variable's neighbourhood.

Appendix A

Distributed Penalty-Based Backjumping Algorithm (DisPBJ)

A.1 Introduction

In this section, the Distributed Penalty-Based Backjumping Algorithm (*DisPBJ*) is presented, combining local search and backjumping to permit practical completeness for bigger problems. DisPeL is run for a number of cycles and distributed backjumping is run if DisPeL is unsuccessful in solving the problem using DisPeL's penalty information to guide variable ordering and value selection.

Our contribution is therefore two-fold: a distributed hybrid algorithm extending completeness to local search, and an ordering heuristic which is able to find solutions quicker than backjumping. In Jussien's classification of hybrid algorithms [48], *DisPBJ* would be classified in the performing local search before/after systematic search category.

A.2 Algorithm Description

The *DisPBJ* algorithm, shown in Algorithm 10, runs DisPeL for a bounded number of cycles. If DisPeL has not solved the problem, DisPeL's penalty information guides variable ordering and value selection. DisPeL was modified so that it counts the overall number of penalties assigned to each agent (as discussed in section 5.3.1 for PenDHyb).

Algorithm 10 Our approach, the DisPBJ Algorithm

```
initialise
repeat
   dispel_agent_main_loop(termination_condition)
until termination condition met
if DisPeL did not find a solution then
   sort agents using the maximum degree heuristic using the agent's penalty count to
   break ties.
   set each variable's current value to the last value assigned by DisPeL
   repeat
      distributed_backjumping(new_agent_order)
   until solution found or no solution detected
end if
```

After initialisation of the agents, a standard DisPeL search is performed. A penalty counter for each variable was added to each agent maintaining penalties accrued by each variable [1]. A penalty counter is incremented whenever DisPeL imposes a penalty on that variable's value. This counter is not reset when DisPeL resets its penalties. A penalty counter highlights repeated penalisation of a variable for an agent throughout the whole search, thereby indicating that the variable is difficult to solve. DisPeL's standard penalty mechanism only accrues this information between penalty resets.

We conducted experiments on **solvable randomly generated DisCSPs** to determine an optimal bound of cycles for DisPeL with 30-60 variables (n) in steps of 10, 10 domain values, $3n$ constraints and constraint tightness of 0.5. Cut-offs were used between $0.5n$ and $7n$ in steps of 0.5. The results of these experiments are shown in table A.1. The optimal cut-off varies according to the number of variables with the cut-off increasing as the number of variables increase. For unsolvable problems, a small bound is always required as DisPeL cannot detect unsolvability.

If DisPeL does not solve the problem within the bounded number of cycles, a distributed version of backjumping [74] either solves the problem or determines that the problem is unsolvable. Our backjumping algorithm (*DisBJ*) uses descending Distributed Agent Ordering [41] with max degree and the penalty count of each variable breaking ties. A "Sticking Values" heuristic (originally proposed in [32]) initialises the first value for each

[1] Note that DisPeL does not keep track of overall penalties as these are reset periodically.

Cut-off	30	40	50	60
Number of Messages				
DisBJ	11,682	68,002	559,820	4,712,013
0.5n	**9,278**	33,528	384,596	1,493,332
1n	11,019	39,870	237,779	1,207,309
1.5n	13,043	36,100	183,141	1,040,986
2n	13,925	43,922	176,301	1,159,978
2.5n	15,584	**33,254**	142,291	630,009
3n	18,773	47,376	130,339	813,628
3.5n	21,628	41,883	111,685	251,633
4n	22,379	47,257	76,390	325,449
4.5n	24,450	44,842	116,840	262,365
5n	26,838	50,919	**76,856**	176,174
5.5n	29,897	53,480	121,434	**119,261**
6n	32,453	58,259	111,257	130,605
6.5n	29,250	58,560	97,719	143,515
7n	38,640	62,280	105,332	151,519
Number of Constraint Checks				
DisBJ	176,730	885,760	7,736,641	65,097,333
0.5n	**114,594**	402,984	5,369,603	21,281,579
1n	128,245	494,449	3,143,463	16,415,687
1.5n	148,583	437,667	2,416,938	14,443,569
2n	155,990	526,635	2,200,473	15,525,753
2.5n	171,424	**371,354**	1,792,222	8,283,567
3n	202,257	543,657	16,03,866	11,295,968
3.5n	229,911	455,163	1,346,886	3,071,316
4n	237,374	517,911	830,421	3,976,873
4.5n	257,455	477,003	1,301,327	3,324,933
5n	282,011	541,341	**810,018**	2,038,639
5.5n	314,317	564,166	1,331,341	**1,254,159**
6n	341,082	611,999	1,235,571	1,373,138
6.5n	307,709	615,892	1,027,384	1,512,688
7n	408,590	654,930	1,107,212	1,592,441

Table A.1: Determining the optimal cut-off value for DisPBJ for $3n$ constraints and constraint tightness of 0.5.

agent with DisPeL's last value used for that variable.

The *DisPBJ* algorithm is complete since either DisPeL will report a solution or cause backjumping to run which will determine whether the problem is solvable. Termination occurs if DisPeL finds a solution in the allocated number of cycles or at backjumping termination points guaranteeing that the algorithm will conclude in finite time. The algorithm is sound since solutions are generated by DisPeL or backjumping which have previously been proven to be sound [8, 74].

A.2.1 Determining the best version of DisPBJ

We ran a number of different versions of *DisPBJ* to determine the effectiveness of the sticking values heuristic. *DisPBJ* (Sticking Values/No Penalties) uses our sticking values heuristic but uses lexicographical ordering to break ties rather than penalties. *DisPBJ* (No Sticking Values/Penalties) omits sticking values but uses penalties to break ties. Table A.2 presents the median results for 100 solvable randomly generated DisCSPs with 40 variables (n), 10 domain values (d), constraint density ($p1$) of 0.15 and constraint tightness ($p2$) of 0.5. Median results are presented since averages unfairly penalise backjumping on harder problems.

Algorithm	**Messages**	**Constraint Checks**
DisBJ	68,002	885,760
DisPBJ (Sticking Values/No Penalties)	76,320	802,334
DisPBJ (No Sticking Values/Penalties)	89,160	937,886
DisPBJ	**50,040**	**526,323**

Table A.2: Determining the effectiveness of Sticking Values with different variants of DisPBJ for $< n{=}40, d{=}10, p1{=}0.15, p2{=}0.5 >$ on distributed random problems.

The penalty and value information from DisPeL appears beneficial in allowing backjumping to process the most difficult variables at the start of the search with less constrained variables near the end of the search. This combination of penalty and value information, whilst individually worse, yields considerably fewer messages and constraint checks than DisBJ.

A.3 Experimental Evaluation

Our hybrid algorithm, *DisPBJ*, is first evaluated against local search, DisPeL. Furthermore, we evaluate *DisPBJ* against distributed backjumping (*DisBJ*) to determine the effectiveness of our approach and the benefits that our approach has on both algorithms.

We evaluated *DisPBJ* against DisPeL on solvable randomly generated problems with 40, 50 and 60 variables, 10 domain values, a constraint density of 0.15 and on constraint tightness of 0.5 and averaged the results over 100 runs. We show the average results since DisPeL solves the vast majority of problems and therefore the median results for DisPeL and *DisPBJ* are identical. DisPeL has been shown to perform well on these problems but does not solve all problems [8]. Table A.3 lists the number of messages and constraint checks.

Number of Variables	**40**	**50**	**60**
Percentage of Problems Solved			
DisPeL	97	92	94
DisPBJ	**100**	**100**	**100**
Number of Messages			
DisPeL	**104,830**	**203,070**	**238,651**
DisPBJ	106,088	246,933	621,977
Number of Constraint Checks			
DisPeL	**1,102,416**	**2,135,261**	**2,509,272**
DisPBJ	1,119,100	2,727,034	8,012,461

Table A.3: DisPeL and DisPBJ Algorithms by Number of Messages and Constraint Checks.

In these experiments, DisPeL solved a very high percentage of problems, but a few problems remained unsolved, whilst *DisPBJ* obtains practical completeness in solving all problems. Naturally, *DisPBJ* incurs more messages and constraint checks than DisPeL since it incurs DisPeL's messages and constraint checks until the bounded number of cycles is reached and then the messages and constraint checks associated with backjumping. However, since *DisPBJ* gives practical completeness, this increase in evaluation metrics appears cost effective.

We evaluated *DisPBJ* against our *DisBJ* algorithm (a distributed version of Prosser's centralised backjumping algorithm [74] with max degree ordering) to determine whether DisPeL's ordering technique on backjumping was beneficial. Our comparison is for randomly generated problems with 30, 40, 50 and 60 variables, 10 domain values, constraint density of 0.15 and constraint tightness of 0.5. We ran the algorithms on 100 solvable problems and removed those problems which DisPeL solved. Most problems with less than 40 variables are easily solved by DisPeL, leaving too few to be solved through backjumping to be able to conduct an analysis. In Table A.4, we present the median results for 40, 50 and 60 variables on solvable problems. We do not count the messages and constraint checks incurred during the DisPeL phase of the *DisPBJ* algorithm. In Table A.5, we present median results for 40, 50 and 60 variables on unsolvable problems where the DisPeL phase in *DisPBJ* cannot detect that the problem is unsolvable so the backjumping phase must always run.

Number of Variables	**40**	**50**	**60**
Number of Messages			
DisBJ	68,002	599,884	4,712,013
DisPBJ	**50,040**	**315,317**	**2,557,046**
Number of Constraint Checks			
DisBJ	885,760	8,450,089	65,097,333
DisPBJ	**526,323**	**4,251,953**	**35,990,609**

Table A.4: DisBJ and DisPBJ Algorithms by Number of Messages and Constraint Checks for Solvable Problems.

Number of Variables	**40**	**50**	**60**
Number of Messages			
DisBJ	103,502	1,496,214	7,717,549
DisPBJ	**98,415**	**1,185,036**	**5,889,355**
Number of Constraint Checks			
DisBJ	1,438,582	20,682,093	109,817,618
DisPBJ	**1,278,583**	**15,592,142**	**80,024,587**

Table A.5: DisPeL and DisPBJ Algorithms by Number of Messages and Constraint Checks for Unsolvable Problems.

DisPBJ uses about half the number of messages and constraint checks than *DisBJ* at 60 variables. The difference is less profound for smaller problems and for unsolvable problems but DisPeL remains an effective ordering heuristic for distributed backjumping.

A.4 Discussion

We compared our synchronous hybrid algorithm, *DisPBJ*, with synchronous *DisBJ* and synchronous SynCBJ [102] on solvable randomly-generated problems with 30, 40, 50 and 60 variables (n), 10 domain values, $3n$ constraints and constraint tightness of 0.5. SynCBJ is considered to be a more efficient algorithm than standard backtracking and backjumping algorithms. Table A.6 presents median results.

Number of Variables	30	40	50	60
Number of Messages				
DisBJ	11,682	68,002	559,820	4,712,013
DisPBJ	32,130	63,840	128,550	1,803,134
SynCBJ	**7,166**	**29,334**	**114,625**	**434,935**
Number of Constraint Checks				
DisBJ	176,730	885,760	7,736,641	65,097,333
DisPBJ	337,992	671,205	1,351,619	1,567,039
SynCBJ	**40,123**	**149,163**	**547,700**	**1,828,217**

Table A.6: DisBJ, DisPBJ and SyncCBJ Algorithms by Number of Messages and Constraint Checks.

Whilst *DisPBJ* always significantly outperforms *DisBJ*, conflict-directed backjumping (SynCBJ) outperforms *DisPBJ*. *DisPBJ* remains an important contribution since small problems can be efficiently solved with *DisPBJ* without incurring the additional nogood storage requirements of SynCBJ.

As a direct consequence of this preliminary study, the *PenDHyb* algorithm was developed and is presented in section 5.3.1. The specific differences between *PenDHyb* and *DisPBJ* are: (i) *PenDHyb* uses SynCBJ to find a solution or detect that a problem is unsolvable when DisPeL fails whilst *DisPBJ* uses *DisBJ*; (ii) *PenDHyb* uses DisPeL's best variable values which minimized constraint violations as the initial variable values for SynCBJ whilst *DisPBJ* uses DisPeL's last variable values as the initial variable values for *DisBJ*.

A.5 Summary

In this appendix, the *DisPBJ* algorithm, a hybrid algorithm which combines penalty-based local search with backjumping systematic search has been presented. *DisPBJ* uses information from DisPeL's penalties and values to guide *DisBJ* (a distributed backjumping algorithm) if DisPeL is unable to solve the problem within a bounded number of cycles.

Appendix B

Evaluating the Cost of Forward Checking in the SEBJ algorithm

Forward checking [3] can be an effective method for reducing the computational effort required within backtracking algorithms. When assigning a value to a variable, forward checking makes sure that no future variables will have no possible values in the domain if the assignment of the current variable's value goes ahead. If a variable has no possible values, then another value is chosen for the current variable. The idea is to minimise the amount of backjumps which will be required. The *SEBJ* algorithm which is an integral part of *Multi-Hyb-Pen* (see chapter 6), *Multi-Hyb-DB* (see chapter 6), *Multi-HDCS-Pen* (see chapter 7) and *Multi-HDCS-DB* (see chapter 7) may benefit from the addition of forward checking. Consequently, we modified the *SEBJ* algorithm to do forward checking during assignment of external variables. We do not do forward checking on internal variables since we would perform additional computation and potentially redundant forward checking as we only need one interchangeable solution for the internal variables. We evaluated *Multi-Hyb-Pen*, *Multi-Hyb-DB*, *Multi-HDCS-Pen* and *Multi-HDCS-DB* with and without forward checking on **distributed randomly generated problems**, **distributed graph colouring problems**, **distributed meeting scheduling problems** and **distributed sensor network problems**.

B.1 Randomly Generated Problems

B.1.1 Solvable Problems

The number of agents was 5, the domain size was 8, constraint density was 0.2 and constraint tightness was 0.35. The percentage of intra-agent constraints varied between 70% and 90% with the remainder being inter-agent constraints. Median results for solvable randomly generated problems are shown in Table B.1.

		Median messages							
Num Vars	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
60	90:10	**399**	2334	**323**	2873	**234**	3,762	**60**	954
60	80:20	**197**	843	**158**	1657	**344**	4,462	**85**	857
60	70:30	818	**802**	**833**	1767	**278**	2,721	**156**	337
70	80:20	**159**	574	**96**	305	**130**	246	**45**	55
70	70:30	**112**	300	**175**	481	**264**	1,428	**60**	104
80	80:20	**143**	311	**60**	**60**	**70**	74	**42**	45
80	70:30	**89**	180	**60**	100	**117**	138	**38**	45
90	80:20	**94**	207	**60**	**60**	**70**	**70**	**35**	37
90	70:30	**81**	167	**60**	**60**	**125**	130	**35**	37
100	80:20	56	**45**	**60**	**60**	**70**	**70**	**35**	**35**
100	70:30	**78**	166	**60**	**60**	**70**	72	**35**	**35**
125	80:20	**20**	**20**	**60**	**60**	**70**	**70**	**35**	**35**
125	70:30	**60**	213	**60**	**60**	**70**	**70**	**35**	**35**
150	80:20	**20**	**20**	**60**	**60**	**70**	**70**	**35**	**35**
150	70:30	**30**	305	**46**	60	**70**	**70**	**35**	**35**
175	80:20	**20**	**20**	**45**	60	**70**	**70**	**35**	**35**
175	70:30	**20**	53	**45**	60	70	**65**	**35**	**35**
		Median NCCCs							
Num Vars	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
60	90:10	**163,585**	332,351	**170,093**	228,539	59,650	175,441	60,088	200,623
60	80:20	**277,408**	424,969	**268,336**	332,784	75,413	320,041	71,387	254,169
60	70:30	**2,761,171**	3,195,619	**2,626,087**	2,982,260	1,012,213	3,056,384	537,988	878,401
70	80:20	**151,678**	155,732	133,577	**105,299**	**50,698**	72,540	**49,960**	75,537
70	70:30	291,421	**287,706**	288,457	**252,790**	**88,373**	169,830	**85,467**	155,008
80	80:20	118,874	**86,706**	114,283	**60,197**	**48,123**	51,687	**49,126**	53,156
80	70:30	169,884	**117,823**	153,848	**88,854**	**56,643**	70,924	**56,339**	73,938
90	80:20	117,668	**59,951**	105,869	**48,391**	**46,855**	47,209	**45,307**	46,956
90	70:30	140,181	**86,475**	130,355	**62,901**	**52,380**	57,722	**51,510**	56,034
100	80:20	107,836	**48,532**	101,792	**46,228**	**44,687**	45,383	**44,571**	45,120
100	70:30	132,031	**79,724**	125,176	**57,312**	**50,638**	55,368	**52,368**	55,741
125	80:20	106,435	**46,117**	104,718	**46,815**	46,992	**46,024**	**46,706**	46,794
125	70:30	125,553	**81,054**	121,680	**54,071**	**51,280**	53,099	**50,360**	53,202
150	80:20	100,020	**49,255**	102,519	**49,523**	**45,587**	49,868	**45,250**	49,377
150	70:30	120105	**102,107**	128,039	**56,539**	**54,756**	56,204	**52,613**	55,343
175	80:20	98,875	**53,850**	103,143	**54,054**	**45,774**	53,992	**45,613**	53,491
175	70:30	110,325	**71,081**	124,838	**60,359**	**51,805**	58,286	**50,468**	58,778

Table B.1: Measuring the effectiveness of Forward Checking on SEBJ for solvable random problems.

For *Multi-Hyb-Pen* and *Multi-Hyb-DB*, forward checking is beneficial for the vast majority of larger problems (80 variables and above) whereas forward checking incurs more constraint checks rather than less for *Multi-HDCS-Pen* and *Multi-HDCS-DB*.

B.1.2 Unsolvable Problems

The number of agents was 5, the domain size was 8 and constraint tightness was 0.35. The percentage of intra-agent constraints varied between 70% and 90% with the remainder being inter-agent constraints. We consider median results for problems where at least one agent has no local solution in table B.2. In these problems, *SEBJ* detects unsolvability and so the number of messages is identical regardless of forward checking since these are only termination detection messages. For NCCCs, forward checking always produce more NCCCs than without. Median results for problems where all agents have local solutions but there is no global solution are presented in table B.3. In these problems, we found that forward checking reduced the number of messages sent but increased the number of NCCCs. The reduction in messages is caused as forward checking takes longer to find the first solution and therefore less messages require to be sent by local search whilst *SEBJ* is running.

B.2 Graph Colouring Problems

B.2.1 Solvable Problems

For distributed graph colouring problems, 150 and 200 nodes were used with 15 to 25 agents, 3 colours and a degree of between 4.9 and 5.1. The percentage of intra-agent constraints varied between 70% and 90% with the remainder being inter-agent constraints. Median results for solvable graph colouring problems are shown in Table B.4.

Forward checking can often reduce the number of messages within the context of our hybrid algorithms. Since forward checking requires additional computation at the top of the tree to enable pruning, there is often a shorter timeframe between finding the first solution and finding all solutions. Therefore, distributed local search runs for a shorter period of time and therefore incurs less messages. Whilst for some problem combinations

			Median messages							
Num Vars	**Constraint Density**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
60	0.2	90:10	14	14	14	14	14	14	14	14
70	0.2	80:20	12	12	12	12	12	12	12	12
70	0.2	70:30	16	16	16	16	16	16	16	16
80	0.2	80:20	12	12	12	12	12	12	12	12
80	0.2	70:30	12	12	12	12	12	12	12	12
90	0.18	80:20	12	12	12	12	12	12	12	12
90	0.18	70:30	12	12	12	12	12	12	12	12
100	0.16	80:20	10	10	10	10	10	10	10	10
100	0.16	70:30	12	12	12	12	12	12	12	12
125	0.14	80:20	10	10	10	10	10	10	10	10
125	0.14	70:30	10	10	10	10	10	10	10	10
150	0.12	80:20	10	10	10	10	10	10	10	10
150	0.12	70:30	10	10	10	10	10	10	10	10
175	0.1	80:20	10	10	10	10	10	10	10	10
175	0.1	70:30	10	10	10	10	10	10	10	10
			Median NCCCs							
Num Vars	**Constraint Density**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
60	0.2	90:10	**52,826**	174,240	**52,826**	174,240	**52,826**	174,240	**52,826**	174,240
70	0.2	80:20	**42,530**	81,719	**42,530**	81,719	**42,530**	81,719	**42,530**	81,719
70	0.2	70:30	**52,179**	89,715	**52,179**	89,715	**52,179**	89,715	**52,179**	89,715
80	0.2	80:20	**43,799**	66,769	**43,799**	66,769	**43,799**	66,769	**43,799**	66,769
80	0.2	70:30	**51,542**	71,058	**51,542**	71,058	**51,542**	71,058	**51,542**	71,058
90	0.18	80:20	**45,684**	57,134	**45,684**	57,134	**45,684**	57,134	**45,684**	57,134
90	0.18	70:30	**61,117**	89,118	**61,117**	89,118	**61,117**	89,118	**61,117**	89,118
100	0.16	80:20	**54,195**	68,184	**54,195**	68,184	**54,195**	68,184	**54,195**	68,184
100	0.16	70:30	**83,499**	134,548	**83,499**	134,548	**83,499**	134,548	**83,499**	134,548
125	0.14	80:20	**67,445**	77,563	**67,445**	77,563	**67,445**	77,563	**67,445**	77,563
125	0.14	70:30	**104,296**	129,868	**104,296**	129,868	**104,296**	129,868	**104,296**	129,868
150	0.12	80:20	**117,291**	134,940	**117,291**	134,940	**117,291**	134,940	**117,291**	134,940
150	0.12	70:30	**181,334**	236,331	**181,334**	236,331	**181,334**	236,331	**181,334**	236,331
175	0.1	80:20	**227,126**	253,182	**227,126**	253,182	**227,126**	253,182	**227,126**	253,182
175	0.1	70:30	**365,401**	415,737	**365,401**	415,737	**365,401**	415,737	**365,401**	415,737

Table B.2: Measuring the effectiveness of Forward Checking on SEBJ for unsolvable random problems where one or more agents has no local solution.

			Median messages							
Num Vars	**Constraint Density**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
60	0.2	80:20	177	**16**	194	**56**	703	**324**	69	**45**
60	0.2	70:30	249	**207**	319	**247**	480	**70**	69	**50**
70	0.18	70:30	114	**52**	166	**92**	418	**70**	54	**40**
80	0.16	70:30	106	**44**	129	**84**	823	**165**	49	**40**
90	0.14	70:30	158	**87**	262	**127**	500	**100**	56	**48**
100	0.13	70:30	**129**	713	**157**	753	674	**379**	**49**	260
			Median NCCCs							
Num Vars	**Constraint Density**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
60	0.2	80:20	**62,205**	172,452	**59,641**	172,452	**53,186**	166,554	**53,186**	166,554
60	0.2	70:30	**251,012**	512,896	**252,212**	512,896	**113,114**	247,778	**83,564**	238,365
70	0.18	70:30	**136,748**	409,832	**136,748**	409,382	**102,594**	344,360	**91,343**	344,360
80	0.16	70:30	**174,461**	480,168	**174,461**	480,168	**135,582**	362,447	**124,409**	362,447
90	0.14	70:30	**374,569**	1,050,722	**372,796**	1,050,722	**254,904**	875,936	**238,437**	842,274
100	0.13	70:30	**362,227**	19,168,433	**354,277**	19,168,433	**330,553**	8,637,679	**298,966**	11,387,490

Table B.3: Measuring the effectiveness of Forward Checking on SEBJ for unsolvable random problems where all agents have local solutions but there are no global solutions.

for some algorithms (particularly *Multi-Hyb-DB* and *Multi-HDCS-DB*) forward checking does reduce NCCCs, in general forward checking substantially increases the number of NCCCs as the benefits of pruning are not outweighed by the additional computation required.

B.2.2 Unsolvable Problems

For distributed graph colouring problems, 150 and 200 nodes were used with 15 to 25 agents, 3 colours and a degree of between 4.9 and 5.1. The percentage of intra-agent constraints varied between 70% and 80% with the remainder being inter-agent constraints. Median results for unsolvable graph colouring problems where one or more agents have no local solutions are shown in Table B.5. For these problems, *SEBJ* detects unsolvability and so the number of messages is only for termination detection and so is identical regardless of forward checking. Forward checking does however increase the number of NCCCs performed. Median results for unsolvable graph colouring problems where all agents have local solutions but there is no global solution are shown in Table B.6. Whilst for *Multi-HDCS-Pen* and *Multi-HDCS-DB*, forward checking did occasionally reduce NCCCs, it substantially increased the number of NCCCs for the other algorithms with only a very

				Median mssgs							
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
150	15	4.9	90:10	**40**	225	**155**	175	**486**	702	**120**	135
150	15	5.1	90:10	**35**	202	**163**	206	**481**	692	**120**	134
150	15	4.9	80:20	**21**	**21**	**134**	**134**	481	**473**	**120**	**120**
150	15	5.1	80:20	**23**	**23**	**143**	**143**	**481**	**481**	128	**120**
150	15	4.9	70:30	**31**	**31**	180	**179**	**495**	**495**	146	**122**
150	15	5.1	70:30	**31**	**31**	**185**	**185**	**467**	495	**122**	123
150	25	4.9	90:10	**35**	428	**177**	180	**1,205**	1,829	**200**	250
150	25	5.1	90:10	**29**	205	**179**	181	1,182	**1,170**	**200**	250
150	25	4.9	80:20	53	**37**	317	**245**	1286	**920**	**188**	200
150	25	5.1	80:20	**37**	42	**245**	261	996	**917**	**200**	**200**
150	25	4.9	70:30	**42**	53	**261**	317	**1014**	1275	200	**172**
150	25	5.1	70:30	**51**	**51**	**338**	**338**	**1102**	1286	204	**169**
200	20	4.9	90:10	**62**	144	**212**	222	**842**	861	**160**	205
200	20	5.1	90:10	**73**	181	**223**	225	**832**	861	**160**	208
200	20	4.9	80:20	**31**	**31**	**188**	**188**	844	**803**	180	**160**
200	20	5.1	80:20	**34**	**34**	**196**	**196**	803	**642**	**141**	160
200	20	4.9	70:30	**59**	**59**	**266**	**266**	842	**663**	**160**	**160**
200	20	5.1	70:30	**77**	**77**	**289**	**289**	**656**	841	162	**142**
200	25	4.9	90:10	**51**	88	**233**	**233**	**1,253**	1,254	**200**	250
200	25	5.1	90:10	**45**	191	**232**	235	**1,253**	1,297	**200**	202
200	25	4.9	80:20	**57**	**57**	**252**	**252**	1253	**1223**	**200**	**185**
200	25	5.1	80:20	**44**	**44**	**250**	**250**	1040	**970**	204	**200**
200	25	4.9	70:30	**56**	**56**	**309**	**309**	**977**	1296	200	**178**
200	25	5.1	70:30	**62**	**62**	**339**	**339**	**1277**	1281	172	**171**

				Median NCCCs							
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
150	15	4.9	90:10	**3,579**	5,039	3,735	**3,369**	**1,387**	2,640	**1,185**	2,351
150	15	5.1	90:10	**3,689**	5,139	3,837	**3,492**	**1,449**	2,469	**1,255**	2,426
150	15	4.9	80:20	**1,314**	2,225	**1,611**	2,599	**1,098**	2,016	**1,081**	1,981
150	15	5.1	80:20	**1,279**	2,323	**1,653**	2,663	**1,105**	1,977	**1,107**	1,977
150	15	4.9	70:30	**1,882**	3,566	**2,659**	4,127	**1,511**	2,938	**1,521**	2,938
150	15	5.1	70:30	**1,783**	3,425	**2,507**	4,127	**1,479**	2,923	**1,500**	2,923
150	25	4.9	90:10	**675**	1,962	775	**622**	**570**	705	**423**	503
150	25	5.1	90:10	**633**	1,384	757	**622**	**541**	545	**459**	501
150	25	4.9	80:20	**729**	744	1,223	**1,012**	1,643	**611**	842	**777**
150	25	5.1	80:20	**549**	737	**800**	997	**632**	647	**800**	870
150	25	4.9	70:30	**726**	1,003	**1,253**	1,498	**1,015**	1,599	1,404	**685**
150	25	5.1	70:30	**534**	1,002	**801**	1,490	**572**	1,441	854	**687**
200	20	4.9	90:10	**4,195**	4,209	4,561	**4,081**	**1,605**	2,961	**1,461**	2,887
200	20	5.1	90:10	**4,403**	4,778	4,646	**3,993**	**1,530**	2,976	**1,438**	2,956
200	20	4.9	80:20	**1,439**	2,478	**1,900**	2,890	**1,391**	2,195	**1,272**	2,195
200	20	5.1	80:20	**1,467**	2,650	**1,925**	3,063	**1,319**	2,299	**1,167**	2,299
200	20	4.9	70:30	**2,369**	4,445	**3,403**	5,138	**1,604**	3,312	**2,084**	3,351
200	20	5.1	70:30	**2,348**	4,412	**3,484**	5,032	**1,872**	3,411	**1,670**	3,411
200	25	4.9	90:10	**1,843**	2,015	2,154	**1,838**	**997**	1,224	**751**	1,281
200	25	5.1	90:10	**1,703**	2,264	2,046	**1,718**	**998**	1,262	**769**	1238
200	25	4.9	80:20	**972**	1,523	**1,261**	1,748	**1,106**	1,229	**1,106**	1,163
200	25	5.1	80:20	**878**	1,393	**1,225**	1,707	**877**	1,143	**939**	1,171
200	25	4.9	70:30	**1,272**	2,112	**2,089**	2,738	**1,033**	1,790	**1,253**	1,554
200	25	5.1	70:30	**1,372**	2,215	**2,156**	2,841	**1,619**	1,908	**879**	1,533

Table B.4: Measuring the effectiveness of Forward Checking on SEBJ for solvable graph colouring problems.

small decrease in messages.

				Median mssgs							
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
150	15	4.9	80:20	42	42	42	42	42	42	42	42
150	15	5.1	80:20	42	42	42	42	42	42	42	42
150	15	4.9	70:30	50	50	50	50	50	50	50	50
150	15	5.1	70:30	48	48	48	48	48	48	48	48
150	25	4.9	70:30	72	72	72	72	72	72	72	72
150	25	5.1	70:30	68	68	68	68	68	68	68	68
200	20	4.9	80:20	57	57	57	57	57	57	57	57
200	20	5.1	80:20	58	58	58	58	58	58	58	58
200	20	4.9	70:30	66	66	66	66	66	66	66	66
200	20	5.1	70:30	64	64	64	64	64	64	64	64
200	25	4.9	80:20	68	68	68	68	68	68	68	68
200	25	5.1	80:20	66	66	66	66	66	66	66	66
200	25	4.9	70:30	79	79	79	79	79	79	79	79
200	25	5.1	70:30	76	76	76	76	76	76	76	76
				Median NCCCs							
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
150	15	4.9	80:20	**1,525**	3,765	**1,525**	3,765	**1,525**	3,765	**1,525**	3,765
150	15	5.1	80:20	**1,421**	3,670	**1,421**	3,670	**1,421**	3,670	**1,421**	3,670
150	15	4.9	70:30	**2,332**	6,562	**2,332**	6,562	**2,332**	6,562	**2,332**	6,562
150	15	5.1	70:30	**2,114**	5,543	**2,114**	5,543	**2,114**	5,543	**2,114**	5,543
150	25	4.9	70:30	**296**	670	**296**	670	**296**	670	**296**	670
150	25	5.1	70:30	**294**	653	**294**	653	**294**	653	**294**	653
200	20	4.9	80:20	**1,415**	4,140	**1,415**	4,140	**1,415**	4,140	**1,415**	4,140
200	20	5.1	80:20	**1,717**	3,808	**1,717**	3,808	**1,717**	3,808	**1,717**	3,808
200	20	4.9	70:30	**2,512**	6,988	**2,512**	6,988	**2,512**	6,988	**2,512**	6,988
200	20	5.1	70:30	**2,253**	6,735	**2,253**	6,735	**2,253**	6,735	**2,253**	6,735
200	25	4.9	80:20	**673**	1,657	**673**	1,657	**673**	1,657	**673**	1,657
200	25	5.1	80:20	**644**	1,537	**644**	1,537	**644**	1,537	**644**	1,537
200	25	4.9	70:30	**895**	2,246	**895**	2,246	**895**	2,246	**895**	2,246
200	25	5.1	70:30	**875**	2,179	**875**	2,179	**875**	2,179	**875**	2,179

Table B.5: Measuring the effectiveness of Forward Checking on SEBJ for unsolvable graph colouring problems where one or more agents has no local solution.

B.3 Meeting Scheduling Problems

B.3.1 Solvable Problems

Median results for solvable meeting scheduling problems appear in Table B.7. The problems had 50-80 meetings, 5 departments, a timeframe of 6 or 7 time units and a constraint density of 0.18. Two departments with common meetings have a random distance between 1 and 3 time units. The percentage of intra-agent constraints varied between 70% and 90% with the remainder being inter-agent constraints.

							Median mssgs				
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
150	15	4.9	80:20	**144**	**144**	**250**	262	**2,300**	2,877	**484**	685
150	15	5.1	80:20	187	**186**	**311**	339	**1,927**	2,539	676	**519**
150	15	4.9	70:30	**388**	**388**	**518**	**518**	**2,346**	2,504	**387**	561
150	15	5.1	70:30	**208**	**208**	364	**363**	2,718	**2,526**	367	**357**
150	25	4.9	80:20	**48**	**48**	261	**255**	**4,131**	4,682	**446**	488
150	25	5.1	80:20	**27**	**27**	**246**	249	**3,879**	3,884	**361**	390
150	25	4.9	70:30	61	**60**	328	**327**	**3,046**	3,769	**309**	356
150	25	5.1	70:30	48	**47**	**333**	**333**	**4806**	5554	**331**	636
200	20	4.9	80:20	266	**265**	414	**413**	3,677	**3,539**	**452**	488
200	20	5.1	80:20	**176**	**176**	342	**341**	3,895	**3,703**	**431**	463
200	20	4.9	70:30	**1,324**	**1,324**	**1,528**	**1,528**	3,650	**3,447**	**429**	440
200	20	5.1	70:30	744	**574**	952	**796**	4280	**2,443**	**317**	333
200	25	4.9	80:20	186	**185**	**376**	377	4,740	**3,909**	**292**	304
200	25	5.1	80:20	**116**	**116**	**313**	331	4,913	**4,664**	**279**	282
200	25	4.9	70:30	354	**353**	627	**602**	5,899	**4,066**	**361**	404
200	25	5.1	70:30	**204**	**204**	**498**	**498**	4,752	**3,823**	**333**	360
							Median NCCCs				
Num Nodes	**Num Agents**	**Deg**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
150	15	4.9	80:20	**2,184**	3,216	**2,275**	3,321	**3,316**	4,245	**4,431**	6,026
150	15	5.1	80:20	**2,166**	3,522	**2,355**	3,654	**2,719**	3,884	6,644	**5,000**
150	15	4.9	70:30	**7,566**	8,851	**7,566**	8,559	**5,524**	8,965	**5,332**	8,559
150	15	5.1	70:30	**4,250**	5,855	**4,250**	5,855	**5,838**	6,422	**5,457**	5,840
150	25	4.9	80:20	**439**	704	**830**	1,040	**3,017**	3,258	**3,613**	4,075
150	25	5.1	80:20	**394**	614	**814**	1,007	**2,694**	2,801	**3,236**	3,477
150	25	4.9	70:30	**558**	919	**1,339**	1,645	**3,738**	4,293	**3,596**	4,371
150	25	5.1	70:30	**514**	837	**1,155**	1,479	**5,266**	7,005	**4,112**	9,474
200	20	4.9	80:20	**3,263**	4,633	**3,263**	4,633	4,175	**3,716**	**4,036**	4,173
200	20	5.1	80:20	**2,375**	3,795	**2,666**	4,176	**3,605**	3,901	**4,100**	4,406
200	20	4.9	70:30	**10,130**	11,390	**10,130**	11,390	8,473	**7,489**	**6,320**	6,586
200	20	5.1	70:30	**7,502**	11,330	**7,502**	11,330	8,037	**4,666**	**4,669**	5,182
200	25	4.9	80:20	**1,607**	1,999	**1,718**	2,158	3,847	**3,207**	**2,017**	2,220
200	25	5.1	80:20	**1,126**	1,652	**1,399**	1,847	3,649	**3,118**	**1,967**	2,077
200	25	4.9	70:30	**3,528**	4,423	**3,532**	4,512	7,140	**4,261**	**4,000**	4,872
200	25	5.1	70:30	**1,968**	3,382	**2,736**	3,945	6,233	**4,321**	**4,224**	4,935

Table B.6: Measuring the effectiveness of Forward Checking on SEBJ for unsolvable graph colouring problems where all agents have local solutions but there is no global solution.

			Median mssgs							
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
50	7	90:10	**20**	**20**	**54**	60	81	**70**	**50**	55
50	7	80:20	**139**	190	**75**	100	**71**	130	**45**	55
50	7	70:30	**460**	961	**328**	471	**221**	278	**73**	193
50	6	90:10	**10**	14	**45**	54	**65**	**65**	**50**	**50**
50	6	80:20	**20**	**20**	**60**	**60**	73	**70**	**35**	45
50	6	70:30	**184**	325	**102**	140	**70**	104	**42**	55
60	7	90:10	**20**	**20**	**60**	**60**	81	**70**	**50**	55
60	7	80:20	80	**50**	**60**	**60**	**70**	**70**	**45**	55
60	7	70:30	**412**	592	**173**	311	**140**	234	**49**	66
60	6	90:10	**10**	20	**45**	60	66	**65**	50	**45**
60	6	80:20	**10**	20	**45**	60	**65**	70	**35**	45
60	6	70:30	**42**	100	**60**	**60**	173	**72**	**35**	55
70	7	90:10	**20**	**20**	**60**	**60**	**68**	70	**50**	55
70	7	80:20	**20**	**20**	**60**	**60**	**70**	**70**	**42**	55
70	7	70:30	**228**	325	**90**	100	**74**	130	**38**	55
70	6	90:10	**20**	**20**	**45**	60	**66**	70	**40**	45
70	6	80:20	**20**	**20**	**60**	**60**	**65**	70	**35**	45
70	6	70:30	**40**	60	**60**	60	**70**	73	**35**	45
80	7	90:10	**20**	**20**	**60**	**60**	**70**	**70**	**50**	55
80	7	80:20	**20**	**20**	**60**	**60**	**70**	75	**37**	55
80	7	70:30	**151**	184	74	**61**	**130**	**130**	**36**	46
80	6	90:10	**20**	**20**	**45**	60	**65**	70	**40**	45
80	6	80:20	**20**	**20**	**60**	**60**	**68**	70	**35**	45
80	6	70:30	**20**	**20**	**60**	**60**	**70**	**70**	**35**	45
			Median NCCCs							
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
50	7	90:10	**7,162**	10,133	**8,623**	11,233	**7,571**	11,476	**7,571**	11,331
50	7	80:20	**10,852**	16,898	**13,139**	20,324	**13,147**	19,869	**13,460**	20,788
50	7	70:30	**20,684**	39,886	**25,451**	44,404	**19,763**	32,431	**19,847**	33,323
50	6	90:10	**2,933**	4,945	**3,956**	5,152	**3,503**	5,181	**3,592**	5,181
50	6	80:20	**4,803**	7,538	**5,259**	7,723	**5,881**	8,042	**5,146**	8,217
50	6	70:30	**7,451**	7,738	**9,632**	15,436	**7,757**	12,900	**7,738**	12,621
60	7	90:10	**10,777**	16,256	**12,076**	15,739	**15,114**	18,853	**12,833**	18,661
60	7	80:20	**16,251**	23,404	**16,367**	25,760	**18,113**	26,628	**18,050**	26,497
60	7	70:30	**37,138**	63,124	**36,649**	71,647	**33,811**	55,387	**33,999**	55,856
60	6	90:10	**5,095**	8,360	**5,700**	8,472	**6,634**	9,138	**5,948**	9,066
60	6	80:20	**6,163**	9,597	**6,346**	9,385	**6,353**	10,289	**6,428**	10,080
60	6	70:30	**11,334**	16,568	**11,654**	17,454	**16,639**	17,338	**12,236**	17,529
70	7	90:10	**15,377**	19,859	**17,757**	18,776	**18,496**	23,921	**18,255**	23,511
70	7	80:20	**20,174**	26,525	**21,380**	26,525	**23,920**	33,704	**24,287**	33,205
70	7	70:30	**38,453**	64,757	**45,164**	61,639	**34,708**	57,505	**35,181**	57,330
70	6	90:10	**6,586**	9,901	**7,573**	9,200	**8,194**	10,506	**7,585**	10,845
70	6	80:20	**9,523**	14,276	**9,632**	12,454	**9,627**	15,882	**9,827**	15,404
70	6	70:30	**14,375**	18,818	**12,949**	19,872	**14,191**	20,317	**14,768**	19,850
80	7	90:10	**17,434**	20,459	**17,651**	23,665	**20,834**	24,962	**20,432**	24,246
80	7	80:20	**27,460**	38,254	**26,809**	38,553	**30,384**	43,599	**30,172**	43,390
80	7	70:30	**50,844**	77,650	**50,219**	83,087	**47,197**	74,693	**49,563**	76,379
80	6	90:10	**8,863**	9,808	**8,461**	10,661	**8,587**	11,019	**8,407**	10,842
80	6	80:20	**10,967**	14,219	**11,202**	13,549	**13,515**	16,051	**11,258**	17,364
80	6	70:30	**14,073**	19,554	**15,645**	19,013	**15,926**	22,457	**16,750**	22,268

Table B.7: Measuring the effectiveness of Forward Checking on SEBJ for solvable meeting scheduling problems.

Whilst occasionally forward checking can slightly reduce messages, the large increase in NCCCs means that forward checking is not beneficial for any of the algorithms for meeting scheduling problems.

B.3.2 Unsolvable Problems

The problems had 50-80 meetings, 5 departments, a timeframe of 6 or 7 time units and a constraint density of 0.18. Two departments with common meetings have a random distance between 1 and 3 time units. The percentage of intra-agent constraints varied between 70% and 80% with the remainder being inter-agent constraints. Median results for unsolvable meeting scheduling problems where one or more agents had no solution to their local problem appear in Table B.8. Forward checking is beneficial for all problems except those with 7 time units and 50 meetings. It would appear that forward checking can, in general, take advantage of the structured nature of the problem to detect unsolvability with fewer NCCCs. Median results for unsolvable meeting scheduling problems where all agents had solution to their local problem but there was no global solution appear in Table B.9.

Multi-HDCS-DB is the only algorithm to benefit from forward checking in terms of number of messages. This would appear to arise from the quick ordering provided by *InterDisBO-wd* and the more focused approach of *SEBJ* with forward checking. However, the cost of forward checking results in an increase in the NCCCs. For all other algorithms, forward checking is not beneficial in terms of number of messages. For NCCCs, it is only for problems with 80 variables where *Multi-Hyb-Pen*, *Multi-Hyb-DB* and *Multi-HDCS-DB* can occasionally perform better with forward checking. This would appear to be because the intra-agent problems now have more variables per agent and therefore the benefit of the pruning outweighs the additional constraint checks required.

			Median mssgs							
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
50	7	80:20	13	13	13	13	13	13	13	13
50	7	70:30	14	14	14	14	14	14	14	14
50	6	80:20	12	12	12	12	12	12	12	12
50	6	70:30	14	14	14	14	14	14	14	14
60	7	80:20	12	12	12	12	12	12	12	12
60	7	70:30	14	14	14	14	14	14	14	14
60	6	80:20	11	11	11	11	11	11	11	11
60	6	70:30	12	12	12	12	12	12	12	12
70	7	80:20	10	10	10	10	10	10	10	10
70	7	70:30	12	12	12	12	12	12	12	12
70	6	80:20	12	12	12	12	12	12	12	12
70	6	70:30	12	12	12	12	12	12	12	12
80	7	80:20	10	10	10	10	10	10	10	10
80	7	70:30	10	10	10	10	10	10	10	10
80	6	80:20	10	10	10	10	10	10	10	10
80	6	70:30	12	12	12	12	12	12	12	12
			Median NCCCs							
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
50	7	80:20	**3,051**	3,981	**3,051**	3,981	**3,051**	3,981	**3,051**	3,981
50	7	70:30	**3,174**	3,526	**3,174**	3,526	**3,174**	3,526	**3,174**	3,526
50	6	80:20	2,315	**1,461**	2,315	**1,461**	2,315	**1,461**	2,315	**1,461**
50	6	70:30	1,916	**1,472**	1,916	**1,472**	1,916	**1,472**	1,916	**1,472**
60	7	80:20	3,055	**2,790**	3,055	**2,790**	3,055	**2,790**	3,055	**2,790**
60	7	70:30	3,476	**2,894**	3,476	**2,894**	3,476	**2,894**	3,476	**2,894**
60	6	80:20	2,211	**1,502**	2,211	**1,502**	2,211	**1,502**	2,211	**1,502**
60	6	70:30	1,980	**1,422**	1,980	**1,422**	1,980	**1,422**	1,980	**1,422**
70	7	80:20	3,395	**2,944**	3,395	**2,944**	3,395	**2,944**	3,395	**2,944**
70	7	70:30	4,343	**2,946**	4,343	**2,946**	4,343	**2,946**	4,343	**2,946**
70	6	80:20	2,275	**1,373**	2,275	**1,373**	2,275	**1,373**	2,275	**1,373**
70	6	70:30	2,576	**1,602**	2,576	**1,602**	2,576	**1,602**	2,576	**1,602**
80	7	80:20	4,637	**2,288**	4,637	**2,288**	4,637	**2,288**	4,637	**2,288**
80	7	70:30	3,941	**2,462**	3,941	**2,462**	3,941	**2,462**	3,941	**2,462**
80	6	80:20	2,210	**1,529**	2,210	**1,529**	2,210	**1,529**	2,210	**1,529**
80	6	70:30	2,890	**1,674**	2,890	**1,674**	2,890	**1,674**	2,890	**1,674**

Table B.8: Measuring the effectiveness of Forward Checking on SEBJ for unsolvable meeting scheduling problems where one or more agents had no solutions to their local problem.

Num Meetings	Num Times	intra: inter	Multi -Hyb -Pen	Multi -Hyb -Pen +FC	Multi -Hyb -DB	Multi -Hyb -DB +FC	Multi -HDCS -Pen	Multi -HDCS -Pen +FC	Multi -HDCS -DB	Multi -HDCS -DB +FC
			Median mssgs							
50	7	80:20	**344**	720	**150**	351	**1,029**	1,191	2,613	**275**
50	7	70:30	**624**	1,270	**517**	1,330	**686**	1,613	730	**171**
50	6	80:20	**222**	652	**91**	165	661	**622**	2,764	**104**
50	6	70:30	**204**	922	**119**	463	**575**	683	959	**83**
60	7	80:20	**320**	1,294	**125**	515	**765**	837	989	**116**
60	7	70:30	**284**	1,448	**210**	983	**547**	1,027	332	**77**
60	6	80:20	**16**	306	**45**	83	**522**	655	1,154	**74**
60	6	70:30	**190**	476	**60**	136	**450**	638	487	**53**
70	7	80:20	**248**	536	**89**	139	**554**	714	492	**63**
70	7	70:30	**242**	588	**91**	238	**517**	936	180	**48**
70	6	80:20	**146**	192	**45**	63	**446**	603	537	**53**
70	6	70:30	**94**	256	**45**	256	**326**	565	211	**45**
80	7	80:20	**196**	502	**83**	502	**577**	691	239	**62**
80	7	70:30	**162**	426	**71**	211	**430**	659	97	**43**
80	6	80:20	**118**	126	**43**	44	**358**	540	273	**42**
80	6	70:30	**86**	164	**45**	60	**318**	466	123	**31**
			Median NCCCs							
Num Meetings	**Num Times**	**intra: inter**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
50	7	80:20	**14,345**	25,983	**18,004**	29,440	**24,749**	29,330	**14,709**	25,035
50	7	70:30	**33,084**	64,581	**38,739**	66,933	**36,899**	69,112	**24,253**	46,122
50	6	80:20	**5,318**	8,212	**7,223**	12,576	**12,884**	13,240	**6,185**	8,488
50	6	70:30	**9,480**	15,242	**10,630**	18,093	**19,681**	23,902	**7,615**	11,970
60	7	80:20	**17,819**	29,143	**19,668**	37,562	**23,816**	30,813	**18,745**	28,338
60	7	70:30	**33,445**	49,026	**37,229**	54,549	**36,205**	60,048	**25,679**	38,605
60	6	80:20	**5,860**	7,387	**6,891**	10,418	**13,498**	16,343	**6,194**	7,654
60	6	70:30	**7,599**	9,900	**9,114**	12,265	**17,772**	26,276	**7,441**	9,660
70	7	80:20	**17,692**	20,660	**20,279**	24,259	**24,275**	27,437	**18,907**	20,643
70	7	70:30	**27,350**	31,753	**29,213**	37,165	**33,241**	56,323	**25,162**	30,350
70	6	80:20	**7,089**	9,176	**8,841**	10,907	**14,621**	20,855	**7,032**	9,152
70	6	70:30	**8,791**	10,335	**9,461**	13,252	**18,238**	30,472	**8,801**	10,187
80	7	80:20	23,671	**23,561**	26,352	**25,211**	**31,380**	35,278	24,388	**23,455**
80	7	70:30	**33,516**	34,936	36,516	**36,108**	**38,205**	52,061	**31,133**	33,253
80	6	80:20	**8,602**	9,222	**9,664**	10,843	**15,678**	23,582	**8,667**	9,522
80	6	70:30	**10,999**	11,473	12,072	**11,473**	**20,603**	32,660	**10,898**	11,139

Table B.9: Measuring the effectiveness of Forward Checking on SEBJ for unsolvable meeting scheduling problems where all agents had solutions to their local problem but there was no global solution.

B.4 Sensor Network Problems

B.4.1 Solvable Problems

Table B.10 shows median results for solvable sensor networks with 5 to 8 targets, 25-64 sensors (grids of 5, 6, 7 and 8), k-visibility of 2, k-compatibility of 1, probability of visibility of 0.9 and probability of compatibility of 0.6. The ratio of intra-agent constraints to inter-agent constraints is 15% to 85%.

		Median n. Messages							
Num Targets	**Num Sensors**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
5	25	**69**	396	**63**	118	**145**	271	**50**	51
5	36	**50**	336	**49**	85	145	**102**	**40**	46
5	49	**25**	105	**42**	60	**85**	88	**40**	50
5	64	**14**	31	**34**	42	85	**68**	**40**	47
6	25	1,649	**475**	765	**376**	**595**	1,070	121	**120**
6	36	1,383	**336**	242	**224**	**210**	398	**54**	60
6	49	338	**103**	116	**86**	**210**	**210**	**54**	**54**
6	64	510	**179**	310	**247**	**120**	126	**54**	57
7	25	3,814	**1,674**	2,300	**619**	**15,737**	26,943	**1,568**	1,650
7	36	3,868	**341**	1,051	**206**	**502**	1,453	100	**95**
7	49	1,092	**139**	210	**114**	**161**	539	**63**	64
7	64	482	**325**	**196**	791	**120**	126	**54**	57
8	25	16,471	**12,171**	**3,644**	5,946	**15,253**	90,248	**12,171**	14,918
8	36	**5,522**	5,749	**3,847**	5,507	**1,083**	1,319	318	**307**
8	49	2,753	**192**	1,100	**176**	**379**	720	**76**	77
8	64	1,175	**712**	**411**	986	**208**	209	**74**	78
		Median n. NCCCs							
Num Targets	**Num Sensors**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
5	25	**4,072**	4,862	**6,599**	8,952	**2,727**	3,399	4,716	**4,248**
5	36	**2,936**	3,885	5,353	**2,625**	2,337	**2,308**	**2,512**	3,433
5	49	**2,708**	3,601	3,431	**2,755**	2,254	**2,229**	**2,374**	3,129
5	64	**2,541**	2,714	2,759	**2,562**	2,371	**2,320**	**2,373**	3,256
6	25	13,164	**11,785**	49,144	**19,578**	**13,266**	**13,266**	17,087	**13,270**
6	36	7,819	**5,138**	**2,306**	13,028	**2,782**	3,527	5,436	**4,473**
6	49	5,706	**3,902**	**2,112**	5,820	**2,406**	3,321	**2,651**	3,362
6	64	18,774	**4,144**	2,497	**2,362**	**2,512**	3,455	**2,594**	3,463
7	25	120,789	**103,725**	133,882	**57,935**	263,885	**253,030**	**253,031**	253,309
7	36	7,819	**5,138**	**2,306**	13,028	**2,782**	3,527	5,436	**4,473**
7	49	21,124	**4,327**	**2,288**	8,329	**2,570**	3,516	4,662	**3,725**
7	64	16,961	**5,488**	2,229	19,532	**2,689**	3,587	2,812	**3,619**
8	25	1,395,619	**1,162,405**	**595,777**	803,731	**2,477,556**	**2,477,556**	2,678,899	**2,477,556**
8	36	21,999	**16,584**	**133,809**	168,663	**36,801**	**36,801**	**39,546**	44,856
8	49	7,420	**6,026**	36,938	**13,040**	**3,045**	3,816	6,737	**4,279**
8	64	19,316	**5,155**	**2,417**	20,647	**2,854**	3,660	**3,713**	3,859

Table B.10: Measuring the effectiveness of Forward Checking on SEBJ for solvable sensor network problems.

Forward checking improves the performance of *Multi-Hyb-Pen* in terms of both number of messages and NCCCs for 6 targets or more. For *Multi-Hyb-DB*, *Multi-HDCS-Pen* and *Multi-HDCS-DB*, it depends very much on the problem parameters as forward checking can often be useful whilst on other problems, it incurs additional costs.

B.4.2 Unsolvable Problems

Table B.11 shows median results for unsolvable sensor networks with 5 to 8 targets, 25-64 sensors (grids of 5, 6, 7 and 8), k-visibility of 2, k-compatibility of 1, probability of visibility of 0.9 and probability of compatibility of 0.6. The ratio of intra-agent constraints to inter-agent constraints is 15% to 85%.

In general, forward checking was an improvement on a small number of targets with a small number of sensors or a high number of targets with a high number of sensors for most algorithms. In particular, forward checking was an improvement on *Multi-HDCS-DB* for almost all problems for both number of messages and NCCCs.

B.5 Summary

In this appendix, we have shown that forward checking can be beneficial for some problems. However, we often found that the use of maximum degree heuristic to order the agents according to connectivity decreased the effectiveness of the forward checking technique since the maximum degree heuristic already minimised backjumping requirements. We therefore leave forward checking as an option which can be used with our algorithms but is not used in the experimental evaluations.

		Median n. Messages							
Num Targets	**Num Sensors**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
5	25	1,293	**1,098**	730	**391**	**1,733**	2,727	262	**247**
5	36	875	**633**	560	**358**	**2,505**	2,836	331	**235**
5	49	1,006	**798**	531	**445**	**2,349**	2,534	300	**232**
5	64	**554**	608	**320**	585	2,052	**704**	**265**	305
6	25	**2,771**	4,328	**1,723**	1,759	**13,910**	26,394	5,641	**1,768**
6	36	14,643	**11,596**	7,069	**6,405**	3,550	**2,579**	5,882	**597**
6	49	**176**	2,801	**136**	238	1,345	**1,037**	3,518	**116**
6	64	**1,156**	3,190	**815**	3,304	2,930	**744**	6,056	**584**
7	25	**7,235**	7,956	4,047	**3,485**	**32,035**	64,571	5,767	**3,157**
7	36	**5,962**	7,516	2,775	**1,951**	**2,386**	2,800	3,486	**370**
7	49	**721**	5,991	**574**	659	**484**	719	3,098	**126**
7	64	**2,041**	3,032	**1,501**	3,443	**1,495**	3,520	7,045	**370**
8	25	**20,488**	90,436	**13,809**	14,077	**52,480**	90,545	**12,580**	12,874
8	36	**8,333**	10,522	**5,098**	6,147	**5,177**	10,377	4,786	**919**
8	49	**1,011**	11,754	**641**	4,563	**1,361**	1,755	3,131	**175**
8	64	**6,539**	13,570	**5,295**	7,879	3,966	**3,204**	7,041	**313**
		Median n. NCCCs							
Num Targets	**Num Sensors**	**Multi -Hyb -Pen**	**Multi -Hyb -Pen +FC**	**Multi -Hyb -DB**	**Multi -Hyb -DB +FC**	**Multi -HDCS -Pen**	**Multi -HDCS -Pen +FC**	**Multi -HDCS -DB**	**Multi -HDCS -DB +FC**
5	25	**22,275**	39,270	29,873	**18,072**	**13,536**	22,952	21,870	**21,708**
5	36	**15,229**	16,909	20,391	**19,819**	**11,340**	11,929	12,587	**9,603**
5	49	22,827	**20,301**	24,551	**19,867**	**10,917**	11,847	**8,393**	8,815
5	64	9,225	**8,304**	**9,787**	12,309	10,170	**6,696**	**4,887**	7,078
6	25	**110,032**	247,259	**131,431**	166,185	**205,965**	248,766	421,110	**205,965**
6	36	821,636	**535,863**	821,633	**523,617**	17,568	**13,266**	194,603	**44,734**
6	49	**3,037**	31,587	**3,364**	8,310	8,123	**4,329**	**2,712**	4,645
6	64	37,684	**19,860**	**38,626**	50,084	17,136	**7,689**	126,000	**18,144**
7	25	**331,460**	616,476	431,012	**419,851**	**381,672**	389,200	624,456	**356,026**
7	36	**65,204**	262,381	55,508	**44,049**	14,283	**14,139**	50,121	**31,932**
7	49	**9,608**	133,907	**11,516**	12,343	**2,813**	6,543	**2,787**	7,146
7	64	**30,609**	35,499	39,313	**38,876**	**15,219**	19,523	189,248	**18,693**
8	25	**1,556,926**	8,916,910	2,071,355	**1,934,516**	**1,782,819**	**1,782,819**	**1,814,238**	1,898,028
8	36	**153,330**	438,717	226,993	**172,452**	**62,316**	69,330	232,983	**65,934**
8	49	**19,284**	365,161	**20,455**	29,468	**7,794**	8,559	**2,921**	8,892
8	64	337,895	**167,629**	379,813	**157,864**	**13,968**	15,462	50,949	**19,170**

Table B.11: Measuring the effectiveness of Forward Checking on SEBJ for unsolvable sensor network problems.