Syracuse University
SURFACE at Syracuse University

Dissertations - ALL
SURFACE at Syracuse University

2-4-2015

# Coalition Formation For Distributed Constraint Optimization Problems

Nathaniel Gemelli
*Syracuse University*

Follow this and additional works at: https://surface.syr.edu/etd

Part of the Computer Sciences Commons

## Recommended Citation

Gemelli, Nathaniel, "Coalition Formation For Distributed Constraint Optimization Problems" (2015). *Dissertations - ALL*. 1801.
https://surface.syr.edu/etd/1801

This Dissertation is brought to you for free and open access by the SURFACE at Syracuse University at SURFACE at Syracuse University. It has been accepted for inclusion in Dissertations - ALL by an authorized administrator of SURFACE at Syracuse University. For more information, please contact surface@syr.edu.

# ABSTRACT

This dissertation presents our research on coalition formation for Distributed Constraint Optimization Problems (DCOP). In a DCOP, a problem is broken up into many disjoint sub-problems, each controlled by an autonomous agent and together the system of agents have a joint goal of maximizing a global utility function. In particular, we study the use of coalitions for solving distributed $k$-coloring problems using iterative approximate algorithms, which do not guarantee optimal results, but provide fast and economic solutions in resource constrained environments. The challenge in forming coalitions using iterative approximate algorithms is in identifying constraint dependencies between agents that allow for effective coalitions to form. We first present the Virtual Structure Reduction (VSR) Algorithm and its integration with a modified version of an iterative approximate solver. The VSR algorithm is the first distributed approach for finding structural relationships, called *strict frozen pairs*, between agents that allows for effective coalition formation. Using coalition structures allows for both more efficient search and higher overall utility in the solutions. Secondly, we relax the assumption of strict frozen pairs and allow coalitions to form under a probabilistic relationship. We identify probabilistic frozen pairs by calculating the *propensity* between two agents, or the joint probability of two agents in a $k$-coloring problem having the same value in all satisfiable instances. Using propensity, we form coalitions in sparse graphs where strict frozen pairs may not exist, but there is still benefit to forming coalitions. Lastly, we present a cooperative game theoretic approach where agents search for Nash stable coalitions under the conditions of additively separable and symmetric value functions.

# COALITION FORMATION FOR DISTRIBUTED CONSTRAINT OPTIMIZATION PROBLEMS

By

Nathaniel Gemelli
B.S. State University of New York at Oswego, 2001
M.S. Syracuse University, 2006

Dissertation

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Syracuse University
December 2014

Copyright ©2014 Nathaniel Gemelli

All rights reserved

# ACKNOWLEDGMENTS

First, I would like to thank my advisor, mentor, and friend, Dr. Jae C. Oh. Over the years his advice and direction has helped guide me to achieve what, at times, I thought was unattainable. I would also like to thank my committee members, Dr. Mehrotra, Dr. Mohan, Dr. Thorson, Dr. Phoha, and Dr. Bhatia for their insightful discussions and comments. I would like to extend my special thanks to all the members of the DMA lab at Syracuse who both helped me along the way while entertaining me with their wide array of personalities and humor! I would like to thank the Air Force Research Laboratory's Information Directorate for their continued support of my academic and professional development. In addition, I'd like to extend a thank you to all my colleagues at the Air Force Research Laboratory. The next best thing to having an adivsor by your side is having a talented and experience research group to bounce ideas and questions off of. Lastly, I'd like to express my love and appreciation to my entire family, especially my wife Adriane, for their unwavering support throughout this entire process. Without it, I would have never succeeded. To my children, Gabriella, Xavier, and Elijah, daddy loves you and you've helped me more than you will ever realize!

# TABLE OF CONTENTS

**List of Tables** viii

**List of Figures** x

**1 Introduction** 1

**2 Background and Related Work** 9

2.1 Distributed Constraint Optimization Problems . . . . . . . . . . . . . . . . . 10

2.1.1 Constraint Graphs . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.1.2 Constraint Density . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.1.3 Motivation for using DCOPs . . . . . . . . . . . . . . . . . . . . . 13

2.1.4 Distributed $k$-coloring and its Applications . . . . . . . . . . . . . 15

2.1.5 Algorithms for Solving DCOPs . . . . . . . . . . . . . . . . . . . . 17

2.2 Coalition Formation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

2.2.1 General Framework . . . . . . . . . . . . . . . . . . . . . . . . . . 24

2.2.2 Coalition Formation Related Work . . . . . . . . . . . . . . . . . . 25

**3 Virtual Structure Reduction on Constraint Graphs** 30

3.1 Frozen Pairs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

3.2 Definition of Frozen Pairs . . . . . . . . . . . . . . . . . . . . . . . . . . 33

3.3 VSR-DSA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

3.3.1 Example . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

3.3.2 Algorithmic Description . . . . . . . . . . . . . . . . . . . . . . . . 39

3.3.3 Algorithmic Details . . . . . . . . . . . . . . . . . . . . . . . . . . 39
3.3.4 Targeted Local Structure Sharing . . . . . . . . . . . . . . . . . . 40
3.3.5 Frozen Pair Discovery . . . . . . . . . . . . . . . . . . . . . . . . 41
3.3.6 Collapse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
3.3.7 Assignment phase . . . . . . . . . . . . . . . . . . . . . . . . . . 43
3.4 Properties of Frozen Pair Discovery . . . . . . . . . . . . . . . . . . . . 45
3.5 Empirical Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
3.5.1 Phase Transitions in Constraint Satisfaction . . . . . . . . . . . . 47
3.5.2 Coalition Formation Results . . . . . . . . . . . . . . . . . . . . . 49
3.5.3 Solver Performance . . . . . . . . . . . . . . . . . . . . . . . . . 52

**4 Propensity-based Coalitions** **67**
4.1 Markov Properties of $k$-Coloring . . . . . . . . . . . . . . . . . . . . . 69
4.1.1 $k$-Coloring as a Markov Chain . . . . . . . . . . . . . . . . . . . 70
4.1.2 $k$-Coloring as a Markov Network . . . . . . . . . . . . . . . . . . 79
4.1.3 Strict Frozen Pairs and Propensity . . . . . . . . . . . . . . . . . . 83
4.2 $\alpha$VSR-DSA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84
4.2.1 Internal Conflict . . . . . . . . . . . . . . . . . . . . . . . . . . . 86
4.3 Empirical Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
4.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89

**5 The Stability of Coalitions for $k$-Coloring** **92**
5.1 Hedonic Coalition Formation Game . . . . . . . . . . . . . . . . . . . . 93
5.1.1 Preferences and Value Functions for Hedonic Games . . . . . . . . 94
5.1.2 Nash Stability . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
5.1.3 Frozen Pairs Graph . . . . . . . . . . . . . . . . . . . . . . . . . 96
5.2 Additively Separable Value Function for Coalition Formation . . . . . . . 98
5.2.1 Strict frozen pairs . . . . . . . . . . . . . . . . . . . . . . . . . . 99

5.2.2 Probabilistic frozen pairs . . . . . . . . . . . . . . . . . . . . . . . . 101

5.3 Hedonic Game DSA Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 103

5.3.1 Empirical Results for HG-DSA . . . . . . . . . . . . . . . . . . . . . 105

5.4 Effects of Coalition Partners with Low Propensity . . . . . . . . . . . . . . 109

5.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111

**6 Conclusions and Future Work 114**

**References 118**

# LIST OF TABLES

3.1 The resulting average **density** for 30, 60, and 90 variable problems after performing a virtual structural collapse. . . . . . . . . . . . . . . . . . . . . . 51

3.2 The resulting number of **active** variables remaining after VSR is used for 30, 60, and 90 variable problems. . . . . . . . . . . . . . . . . . . . . . . . 52

3.3 The average number of experiments that resulted in an “at most” percentage of violated constraints after 500 cycles for the VSR-DSA and DSA-B algorithms. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60

3.4 Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 2.5. . . . . . . . . . . 62

3.5 Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 2.75. . . . . . . . . . . 62

3.6 Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 3.0. . . . . . . . . . . 63

3.7 Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 3.5. . . . . . . . . . . 64

3.8 Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 4.0. . . . . . . . . . . 64

4.1 A factor function for $k = 3$ coloring. . . . . . . . . . . . . . . . . . . . . . 80

4.2 A factor function for $k = 3$ coloring with the added “equals” constraint being valued for two variables adjacent and having the value “blue”. . . . . 83

5.1 Additive separable values for each agent in the coalition game from Figure 5.3. . . . . . 100

5.2 Additive separable values for each agent in the coalition game from Figure 5.4. . . . . . 102

# LIST OF FIGURES

1.1 The traditional process for solving a DCOP problem. . . . . . . . . . . . . 2

1.2 The coalition formation integrated process for solving a DCOP problem. . . 2

1.3 A Kilobot. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.4 A sensor coverage problem framed as a DCOP. Different task roles are indicated by the colors of the coverage areas surrounding each Kilobot. . . 4

1.5 Example of reducing the original problem into a set of coalitions that represent the original problem in a more compact way. . . . . . . . . . . . . 5

2.1 An example of a DCOP represented as a constraint graph. . . . . . . . . . . 11

2.2 The phase transition for k-coloring DCOPs from easy to hard to easy problem instances based on constraint density. . . . . . . . . . . . . . . . . . 12

2.3 A taxonomy of DCOP algorithms. In the diagram, DSA = Distributed Stochastic Algorithm, DBO = Distributed Breakout Algorithm, MGM = Maximum Gain Messaging Algorithm, FP = Fictitious Play, JSFP = Joint-Strategy Fictitious Play, RM = Regret Matching, ADOPT = Asynchronous Distributed Optimization Algorithm, OptAPO = Optimal Asynchronous Partial Overlay Algorithm, DPOP = Dynamic Programming Optimization Algorithm. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.1 Simple example of a frozen pair between $x_1$ and $x_2$. Every satisfiable value assignment for this problem results in these two variables having the same color. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

3.2 Examples of frozen pair structures in graph form, with the core in grey, for domains of size 2, 3, and 4. Frozen pairs are variables that must have the same assigned value in a solution when they share the same neighbors of a core. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.3 Illustration of VSR on k-coloring problem, k=3. (a) Constraint graph in original form. (b) Identification of frozen pairs indicated by dashed lines. (c) Grey nodes indicate surrogate agents. (d) Reduced constraint graph. . . 38
(a) Initial constraint graph. . . . . . . . . . . . . . . . . . . . . . . . . 38
(b) First identification of strict frozen pairs. . . . . . . . . . . . . . . . . 38
(c) Second identification of strict frozen pairs. . . . . . . . . . . . . . . 38
(d) Resulting constraint graph. . . . . . . . . . . . . . . . . . . . . . . . 38
3.4 Symmetry property of frozen pairs. Agents $x_1$ and $x_2$ will independently discover each other. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
3.5 Transitive property of frozen pairs. Agents $x_2$ and $x_3$ do not initially recognize each other, but will end up in the same coalition with $x_1$. . . . . . . 46
3.6 Induction property of frozen pairs. Upon the collapse of $x_2$ and $x_3$, $x_1$ and $x_4$ will discover each other in a frozen pairs relationship allowing them to form a coalition. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
3.7 The phase transition for k-coloring DCOPs from easy to hard to easy problem instances based on constraint density. The rightmost shaded region is the original density region where the “hard” problems lie. The leftmost shaded region is the density range we attempt to achieve through the coalition formation process. . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
3.8 Resulting active density values from virtual structure collapsing. . . . . . . 51
3.9 Resulting active structure from virtual structure collapsing. . . . . . . . . . 52

3.10 Average number of messages required to solve various instances of 3-coloring. Here, the shaded area under the curve is displaying the number of messages required for only the coalition formation phase (VSR alone). . 54
3.11 Average solution quality, $n = 40$. . . . . . . . . . . . . . . . . . . . . . . . 55
3.12 Average number of cycles used to return a result or exceed maximum cycles allotted. Here, the shaded area under the curve is displaying the number of cycles required for only the coalition formation phase (VSR alone). . . . . 57
3.13 Average solution quality at a density of 2.25, just before entering the phase transition of a $k$-coloring, 100 variable problem. . . . . . . . . . . . . . . 57
3.14 Average solution quality at a density of 2.7, at the beginning of the phase transition of a $k$-coloring, 100 variable problem. . . . . . . . . . . . . . . 58
3.15 Average solution quality at a density of 3.5, well into the phase transition of a $k$-coloring, 100 variable problem. . . . . . . . . . . . . . . . . . . . . 59
3.16 Average number of messages passed at a density of 2.5 as number of variables increase. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
3.17 Messages passed at density = 2.5. . . . . . . . . . . . . . . . . . . . . . . 61
3.18 Cycles used at density = 2.5. . . . . . . . . . . . . . . . . . . . . . . . . 61
3.19 Messages passed at density = 2.75. . . . . . . . . . . . . . . . . . . . . . 62
3.20 Cycles used at density = 2.75. . . . . . . . . . . . . . . . . . . . . . . . 62
3.21 Messages passed at density = 3.0. . . . . . . . . . . . . . . . . . . . . . . 63
3.22 Cycles used at density = 3.0. . . . . . . . . . . . . . . . . . . . . . . . . 63
3.23 Messages passed at density = 3.5. . . . . . . . . . . . . . . . . . . . . . . 64
3.24 Cycles used at density = 3.5. . . . . . . . . . . . . . . . . . . . . . . . . 64
3.25 Messages passed at density = 4.0. . . . . . . . . . . . . . . . . . . . . . . 65
3.26 Cycles used at density = 4.0. . . . . . . . . . . . . . . . . . . . . . . . . 65
3.27 Average number of msec required to solve various instances of k-coloring DCOPs. Performed on a 2.4Ghz Intel Core i7. . . . . . . . . . . . . . . . 66

4.1 Full enumeration of all possible proper colorings for $k = 3$, $n = 3$. . . . . . 71

4.2 A simple chain example of the 3-coloring problem. Dashed lines indicate a propensity measure. . . . . . . . . . . . . . . . . . . . . . . . . . . . 72

4.3 A chain example of the 3-coloring problem with associated probabilities of variables having the same joint assignment. Dashed lines indicate this probability. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72

4.4 A chain example of the 4-coloring problem with associated probabilities of variables having the same joint assignment. Dashed lines indicate this probability. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73

4.5 Probabilities of equivalent joint assignments based on graph distance between two nodes in a chain, random = $\frac{1}{k}$. . . . . . . . . . . . . . . . . . 74

4.6 Propensity measures (dashed lines) between agents with growing cycle structure. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75

4.7 As $n$ grows sufficiently large in a cyclic structure, propensity is observed to converge to that found in the Equation 4.1. . . . . . . . . . . . . . . . . 76

4.8 Increasing shared neighborhoods between two variables leads to rapidly increasing propensity values. . . . . . . . . . . . . . . . . . . . . . . . 77

4.9 Propensity for increasing sized shared neighborhoods. In the figure, $b$ represents the size of the shared neighborhood between two nodes in the graph. 79

4.10 A strict frozen pair for $k$=3 coloring. Dashed line indicates propensity. . . . 84

4.11 a. True propensity, b. Local propensity estimation, c. Resulting coalition with internal conflict between agents 4 and 6 if this coalition is formed. . . 87

4.12 Solution quality for $n = 50$ at a density of 2.25. . . . . . . . . . . . . . . . 88

4.13 Solution quality for $n = 75$ at a density of 2.25. . . . . . . . . . . . . . . . 89

4.14 Solution quality for $n = 100$ at a density of 2.25. . . . . . . . . . . . . . . 90

5.1 A simple constraint graph (left) with $\alpha = 0.67$. Resulting frozen pairs graph (right) with two edges. . . . . . . . . . . . . . . . . . . . . . . . 97

5.2 A simple constraint graph (left) with $\alpha = 1.0$. Resulting frozen pairs graph (right) with no edges as there are no strict frozen pairs. . . . . . . . . . . . 97

5.3 (a) A simple constraint graph with two frozen pairs. (b) A simple frozen pair graph. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100

5.4 a. True propensity, b. Local propensity estimation, c. Resulting coalition with internal conflict between agents 4 and 6 if this coalition is formed. . . 102

5.5 The frozen pairs graph for the constraint graph in Figure 5.4. . . . . . . . . 102

5.6 Average number of cycles used to find solutions using HG-DSA. . . . . . . 105

5.7 Average number of messages sent within the system using HG-DSA. . . . . 106

5.8 Solution quality for $n = 50$. . . . . . . . . . . . . . . . . . . . . . . . . . 107

5.9 Solution quality for $n = 75$. . . . . . . . . . . . . . . . . . . . . . . . . . 108

5.10 Solution quality for $n = 100$. . . . . . . . . . . . . . . . . . . . . . . . . 108

5.11 (a) Original constraint graph with $\alpha = 0.67$. (b) Resulting coalition from using $\alpha = 0.67$. (c) Original constraint graph with $\alpha = 0.5$. (d) Resulting coalitions from using $\alpha = 0.5$. . . . . . . . . . . . . . . . . . . . . . . . . 109

5.12 HG-DSA at $\alpha = 0.5$ with $n = 75$ using a surrogate selection that looks for highest propensity first (High), lowest propensity first (Low), and exact propensity first (Exact). . . . . . . . . . . . . . . . . . . . . . . . . . . 111

6.1 Solution quality using degree-based surrogate selection for $n = 100$ using VSR-DSA. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117

# CHAPTER 1

# INTRODUCTION

Coalition formation is of fundamental importance in a wide range of research disciplines, including social sciences, economics, and computer science. Ultimately, the process of forming a coalition comes down to choice. An agent must choose which coalition to join and what role that agent will play within the coalition. Unfortunately, in environments with dynamic tasks, time constraints, and unknown team sizes, a priori planning and/or centralized approaches to designating agents to coalitions becomes infeasible. It becomes necessary to enable the agents themselves to reason about the proper organization of the coalitions that should form in order to effectively solve a problem. In order to represent the decision problem of coordinating agent teams, researchers in multi-agent systems introduced distributed constraint satisfaction and optimization problems, or DCSP and DCOP respectively. Both DCSP and DCOP are an extension from successful work in the area of constraint programming and a powerful way to represent local reasoning and communication interactions. In this work, we assume that agents are solving a constraint problem in the form of DCOP as it is a popular method for representing both non-cooperative and cooperative agent systems that maps well to a variety of domains.

The research presented in this dissertation can be viewed as a marriage between DCOP and coalition formation. In Figure 1.1 we show a representation of how the process of finding DCOP solutions occurs in a traditional sense. A problem instance is created from a set of variable constraints, a DCOP solver is called, and a solution is output. While some approaches exist, which we cover later in this chapter, that pre-process the DCOP instance before sending the instance to a solver, they are performed offline and with no solver interaction. Our work is unique for two reasons: (i) we utilize a coalition formation process that operates directly inline with a solver to reduce problem difficulty, and (ii) we integrate the coalition formation process to work with iterative approximate solvers such that it requires minor modification to the solver itself. In Figure 1.2 we show the insertion of the coalition formation process into the traditional solution process. As this dissertation will cover, the formation of coalitions simplifies DCOPs such that problem complexity is reduced as well as problem size. By doing so, DCOP solvers find solutions faster and more efficiently as they are performing search on smaller representations of the same problem.

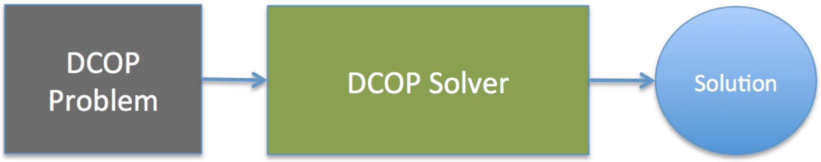


Fig. 1.1: The traditional process for solving a DCOP problem.

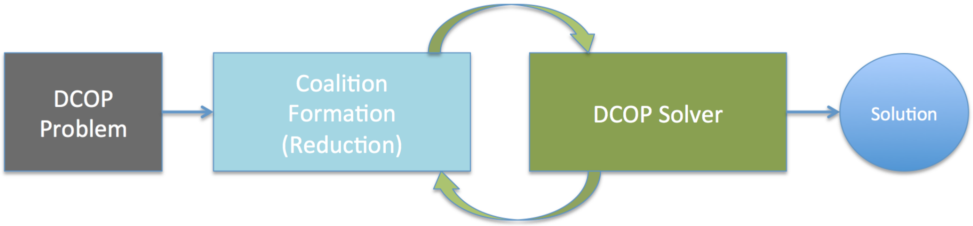


Fig. 1.2: The coalition formation integrated process for solving a DCOP problem.

To motivate the use of DCOPs for the real-world, we present an example. Consider the simple scenario of a team of agents self-organizing and collaborating to achieve the selection of roles in a robotic monitoring application using very simple hardware with limited capabilities. The Kilobot platform (Figure 1.3), a recent development from the

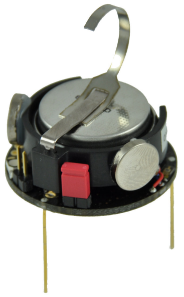


Fig. 1.3: A Kilobot.

Harvard Self-organizing Systems Research Group[1], is a low-power, quarter-sized robot that relies on fast and economic solutions to carry out complex behaviors.

Let us assume that a large number of these Kilobots have been disseminated into an environment via airdrop from a plane with an unknown distribution (and uncertain relative placement), Figure 1.4. The Kilobots are expected to carry out certain tasks such as monitoring temperature, observing siezmic activity, or providing some other sensor coverage. Assume a desired global property of the group of Kilobots is to avoid creating dense pockets of the same task role in any given region as this helps to provide maximal sensor coverage of all types across the region. Kilobots form restrictions, or constraints, with other agents that are within a certain distance. Constraints between any two Kilobots dictate that they do not take on the same task role. As the robots were placed without prior knowledge of where they would be spatially located before deployment, prior assignment of roles or a priori planning will very likely fail in this scenario. A natural way of representing this problem is as a DCOP. The Kilobots negotiate their role in the larger group by using the constraints they have formed with other agents within their vicinity. On small-scale robotics with limited capabilities, computing optimal solutions could be very computationally burdensome, probably would not be possible, and most likely are not even necessary. Simpler approaches, such as approximate algorithms, to negotiating the task roles would be sufficient.

A major hurdle with using simpler approaches in distributed applications is that as prob-

---

[1] http://www.eecs.harvard.edu/ssr/

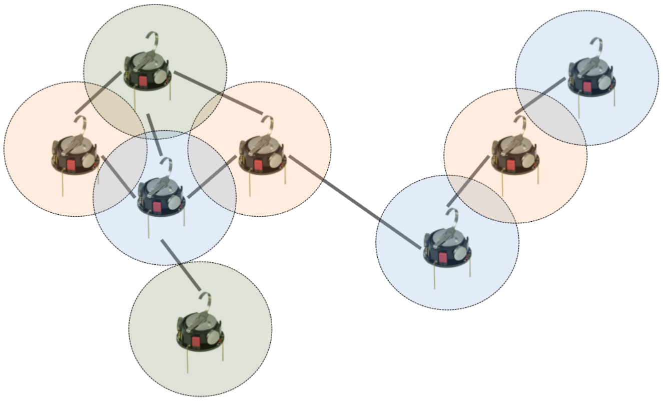


Fig. 1.4: A sensor coverage problem framed as a DCOP. Different task roles are indicated by the colors of the coverage areas surrounding each Kilobot.

lems grow in terms of both size and complexity, performance tends to degrade in terms of cost and quality. As in the scenario above, we cannot just turn back to more complex algorithms, so the question becomes "how can we make simple solutions better?" In this work we will show that using *coalitions* of agents within a larger group can enable simpler solution approaches to perform at higher levels, even as problem size and complexity grows. The principal question addressed in this thesis is the following:

> *How can we enable a set of autonomous agents to distributedly self-organize into coalitions of agents that are more efficient and effective at solving Distributed Constraint Optimization Problems using simple iterative approximate solvers?*

What we wish to do is to provide a mechanism for agents to identify coalitional partners and form coalitions that represents the same problem, but in a reduced way. For example, in Figure 1.5 we show a constraint graph on the left that represents the original problem in full form. By using our coalition formation technique, we reduce the complexity of the graphs in terms of size and difficulty, as shown on the right of Figure 1.5. The coalitions that form represente the original problem in an abstract way, where agents within a coalition are represented by one individual that negotiates for all members. When only one agent negotiates for an entire group, search for a solution can progress in a more streamlined fashion. The work we present in this dissertation is on how the process of coalition formation can be achieved in DCOPs and what the effects of that coalition formation process are in terms of problem complexity and solver performance.

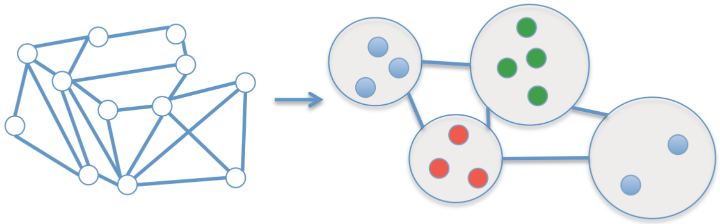


Fig. 1.5: Example of reducing the original problem into a set of coalitions that represent the original problem in a more compact way.

### *Layout of Dissertation*

The formation of coalitions for solving DCOPs has received limited investigation in the literature. Of the work that has been done, the research has been focused on partitioning a DCOP into (possibly overlapping) partitions and then running complete optimal solvers within those partitions. We detail the shortcomings of this methodology in Chapter 2. We distinguish our contributions in this dissertation as we propose a methodology for forming coalitions to solving DCOPs using iterative approximate methods, which results in very good approximate solutions at a very low cost. In particular, we show that the use of coalition formation directly impacts the efficiency and effectiveness of low-cost iterative approximate solvers in the distributed $k$-coloring problem [2], a canonical problem used in testing DCOP solvers from both the complete optimal and iterative approximate approaches. The $k$-coloring problem can be used to represent many problem types, which we cover in Chapter 2, including the task role allocation problem presented above. We compare our coalition formation approach on $k$-coloring problems of varying degree of difficulty and report positive results in empirical testing against other iterative approximate solvers introduced in Chapter 2. We propose three state-of-the-art contributions.

---

[2] We will also simply refer to this as the $k$-coloring problem.

First, we have developed a novel distributed approach to efficiently forming coalitions of agents in DCOPs with binary constraints and homogeneous domains, called the Virtual Structure Reduction (VSR) algorithm. As we will discuss in Chapter 3, the VSR algorithm works by finding relationships between agents in a DCOP that must have the same variable assignment. We call these relationships *strict frozen pairs*, or simply *frozen pairs*. Frozen pairs were first discussed in foundational work carried out by Cheeseman and Kanefsky [13], and continued by Culberson and Gent [15], in analyzing the "easy-hard-easy" phase transition that exists where constraint satisfaction problems go from satisfiable to non-satisfiable as a function of problem density. Once frozen pairs are identified, it allows for the delegation of authority from one agent to another over their joint variable assignment. Agents that take on this authority to negotiate variable assignment for others are termed *surrogate* agents and result in surrogate-led coalitions. To test the impact that coalition formation has on the search for a DCOP solution, we combine our VSR approach with a new version of the iterative approximate solvers, the Distributed Stochastic Algorithm (DSA). The original DSA algorithm is detailed in Chapter 2 and the new version, DSA-S, that works with our coalition formation process is presented in Chapter 3. By using the VSR algorithm online with the DSA-S algorithm, we find both efficiency gains and solution quality gains within the DCOP phase transition area. Our empirical results [20, 21] tested against an optimized version of the DSA, DSA-B, and the Maximum Gain Messaging (MGM) algorithm show that surrogate-led coalitions are a powerful approach for increasing efficiency in the search for a DCOP solution; by directly reducing message passing and computation overhead as well as indirectly reducing problem difficulty.

Our second contribution, detailed in Chapter 4, is in the introduction of weaker version of the frozen pair concept. As we will show, the strict frozen pair relies on very particular relationship to exist between agents in a DCOP instance. The strict frozen pairs relationship is prevalent in particular regions of problem density. In regions where strict frozen pairs are not as plentiful, we would like to be able to find pairwise relationships that can still enable agents to find suitable coalitional partners that lead to positive results. We introduce the idea of a *probabilistic frozen pair* and show the effectiveness of using them as the basis for coalition formation in sparse regions of the phase transition where problem density is lower. Probabilistic frozen pairs lack the property of being guaranteed to have the same joint variable setting, but are highly correlated with each other so that they could still benefit from forming a coalition and being solved via surrogate-led negotiation. We introduce a measure of *propensity*, or a probability of variables having the same value assignment in a satisfying solution of a $k$-coloring problem. We show that by using different levels of propensity as a criteria for agents to form coalitions, we can obtain increased solution quality where the strict frozen pairs coalition formation may have suffered. We introduce our $\alpha$VSR-DSA [19] algorithm, which introduces coalition formation among probabilistically frozen agent pairs but solves $k$-coloring problems using the same iterative approximate search processes as with the VSR coalition formation process. The $\alpha$VSR-DSA algorithm weakens the assumption of strict frozen pairs for a more flexible solution while still mitigating the degradation of solution quality seen in large, complex problem instances. The $\alpha$VSR-DSA algorithm provides an alternative approach for dealing with problem instances where the occurrence of strict frozen pairs is not prevalent.

Our third contribution in this work, Chapter 5, is the game theoretic analysis of the coalition structures generated by the VSR-DSA and $\alpha$VSR-DSA algorithms. We frame the coalition formation process as a Hedonic game, or game of preference, and show that the coalitions which form under our formation process are Nash stable. We present the Hedonic Game algorithm (HG-DSA) and show that by allowing agents to selectively consider different coalitions based on the members within that coalition, we can improve solution quality further in low density regions of the $k$-coloring problem. We provide some counter-intuitive results of the coalition formation process under the Hedonic game formulation, and discuss why selecting the highest propensity coalitions might not always be the best choice.

We conclude in Chapter 6 with remarks about our work and some future research directions regarding the coalition formation process in $k$-coloring and general DCOPs.

# CHAPTER 2

# BACKGROUND AND RELATED WORK

This research focuses on the process of forming coalitions for solving DCOPs using an iterative approximate solver. The use of an iterative approximate solver offers us a solution type which is economic and efficient in terms of both computing power and time, respectively. To demonstrate the effectiveness of our solution technique, we focus on a canonical benchmark problem which many researchers use for testing the efficiency and effectiveness of their algorithms; the distributed $k$-coloring problem. As we will point out, the distributed $k$-coloring problem represents many application areas of practical use.

In the first section of this chapter, we discuss the general Distributed Constraint Optimization Problem (DCOP) formalization and the techniques used to solve Multi-Agent System (MAS) decision problems framed in such a context. We present the mathematical formulation of a DCOP and its constraint graph representation. We present the distributed $k$-coloring problem and frame it as a DCOP, then provide examples of a mapping from $k$-coloring to application areas. We then break DCOP solution techniques into 2 distinct types: (1) optimal complete algorithms and (2) approximate local iterative algorithms. We give an overview of the approach types with presentation of the most influential and successful algorithms from their respective areas.

The second section of this chapter is focused on coalition formation games, a branch of cooperative game theory. We introduce coalition formation games and its related work from game theory, distinguish between cooperative and non-cooperative games, and introduce a particular type of coalition formation game called Hedonic coalition formation games. We provide an overview of approaches from the DCOP community that are closely related to coalition formation games for solving DCOPs, but point out that they do not utilize iterative approximate solvers.

## 2.1 Distributed Constraint Optimization Problems

Distributed Constraint Optimization Problems (DCOP) arose from the relaxation of Distributed Constraint Satisfaction Problems (DCSP) [74]. Since it is not always possible to find a fully satisfying solution to a DCSP, optimizing the solution as much as possible is desirable in many contexts. A DCOP is defined as a 3-tuple $\langle X, D, F \rangle$ such that:

- $X = \{x_1, \ldots, x_n\}$ is a set of variables
- $D = \{d_1, \ldots, d_n\}$ is a set of finite domains for each variable, where each $d_k$ represents a finite set of possible variable values
- $F = \{f_1, \ldots f_m\}$ is a set of cost functions, or constraints between the variables in $X$

Each $f_i$ is defined over a scope of variables $(x_{i_1}, \ldots, x_{i_z})$ that returns a real-valued number, $r \in \mathcal{R}$ based on the combination of variables involved in the constraint. A *binary* constraint problem is one where every $f_i$ is defined over the combination of two variables, such that $f_i(x_j, x_k) = r$. Given a set of cost functions over a set of variables, the goal is to find a value assignment for each variable, $A = \{x_1 = d_1, \ldots, x_n = d_n\}$, such that a global cost function is either minimized or maximized,

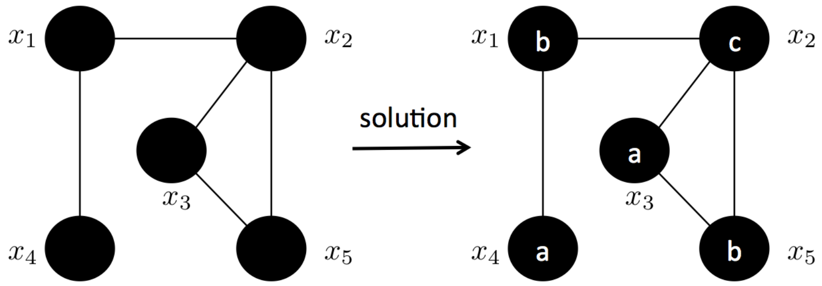
$$G(A) = \sum_{f_i \in F, (x_y, x_z) \in A} f_i(x_y, x_z), \tag{2.1}$$

where $x_y$ and $x_z$ have been assigned values according to $A$. For clarity, but without loss of generality, most researchers focus on binary constraint problems.

### 2.1.1 Constraint Graphs

A binary constraint problem can be represented as an undirected *constraint graph* (see Figure 2.1) of the form $G = \langle V, E \rangle$ where each variable, $x_i \in X$, is represented by a vertex, $v_i \in V$, and each binary constraint, $f_i$, is represented by a weighted edge, $e_i \in E$, where the edge weight, denoted as $w(e_i)$, is equivalent to the cost function $f_i$. We assume that all cost functions are binary relations although relations of higher order are possible. In addition, we assume that each agent is given responsibility over a single variable at the beginning of a problem instance as is typically assumed, however multi-variate extensions have been developed for many of the single-variate approaches [52].

$$D = \{d_1, d_2, d_3, d_4, d_5\} = \{a, b, c\}$$



$$F = \{f_1(x_1, x_4), f_2(x_1, x_2), f_3(x_2, x_3), f_4(x_2, x_5), f_5(x_3, x_5)\}$$

Fig. 2.1: An example of a DCOP represented as a constraint graph.

In Figure 2.1, we provide a simple, five variable example of a constraint graph, along with its solution. Each variable is represented by a node in the graph, and for each constraint in the problem, there is an edge relating the appropriate variables. Here the domains, $\{d_1, d_2, d_3, d_4, d_5\}$, for each variable, $\{x_1, x_2, x_3, x_4, x_5\}$, are the same, specifically $\{a, b, c\}$. The set of binary constraints, $F$, for this problem are the "not-equals" constraint, where no two adjacent nodes (variables) may take on the same value assignment. Pictured on the right is one possible solution with zero constraint violations. As a DCOP, each variable would be assigned to an agent that is responsible for negotiating an assignment for its variable with the other agents in the problem.

### 2.1.2 Constraint Density

Of course, not all DCOPs are created equal. *Constraint density* is the ratio of the number of constraints to the number of variables, $\mathcal{D} = \frac{|F|}{|X|}$, and is considered to be directly correlated with problem difficulty [2]. As constraint density grows, agents must reason over a larger set of constraints, making for more complex problem instances. Typical constraint densities studied in the literature vary, but for small domain sizes, typical values range from 2.0 (underconstrained) to 3.75 (highly constrained). The constraint density in our simple example problem, Figure 2.1, is only 1.0.

The importance of the constraint density, and the role it plays in the hardness of finding solutions for DCOPs should not be understated. Similar to phase transitions in physical and natural processes such as water freezing to ice, the density of a constraint optimization problem is considered the *order parameter* of the system. In Figure 2.2 we illustrate the effect of density on the $k$-coloring problem for both problem difficulty and number of problem instances that are satisfiable.

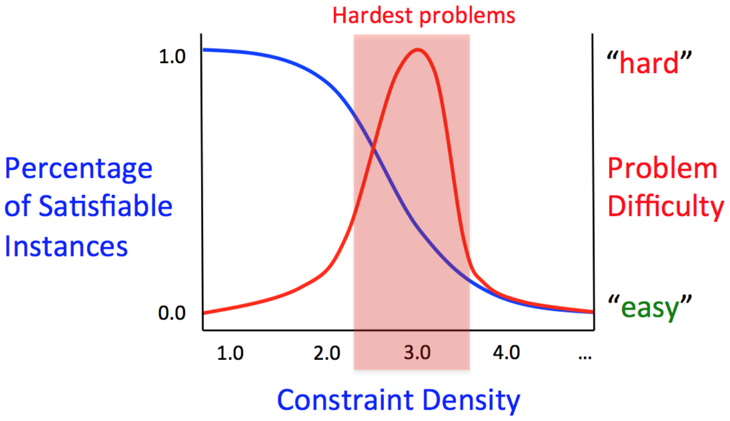


Fig. 2.2: The phase transition for k-coloring DCOPs from easy to hard to easy problem instances based on constraint density.

A rapid transition from easy-to-solve problems to hard-to-solve problems and back to easy-to-solve problems occurs through the space of DCOP instances as the constraint density of the problem goes from low values, such as 1.0, to high values of 5.0 and greater. In addition, the set of satisfiable instances becomes less and less as density continues to rise. We cover the phase transition in more detail in Chapter 3, Section 3.5.1.

### 2.1.3 Motivation for using DCOPs

Constraints have been recognized as an useful and powerful way of representing reasoning problems in AI [61]. Constraint Satisfaction Problems (CSP) [76] are a general way of representing various problems in artificial intelligence where the goal is to find a combination of variable settings that are consistent, or do not violate any constraints that exist between the variables value settings. *Distributed* Constraint Satisfaction Problems (DCSP) arose as an extension to CSPs where a set of autonomous agents are each given the responsibility of solving for only the variable(s) which they are assigned. In some environments, the discovery of a consistent solution for a DCSP is not always desirable or possible. Some environments could be over-constrained, and there simply exists no consistent solution. In other environments, the complexity of the problem and limited amount of search time may dictate that waiting for a consistent solution is not possible. And even more so, there are instances of problems where different consistent solutions are more valuable than other consistent solutions. Due to the reality of complex problems and varying solution qualities, researchers turned to a relaxed version of DCSP where solution quality mattered and consistency was not always necessary.

The motivation for designing a system of loosely-coupled agents in such a fashion of a DCOP is to allow for collaborative autonomous agents to work together to accomplish a global goal. Formulating problems in a distributed manner can provide benefit in terms of scalability, redundancy, and flexibility of a system, all important aspects of problems where an environment has unknowns and the number of agents in the system is expected to be large. Explicitly, for scalability, we are interested in increasing the efficiency of a system, regardless of how large that system could become. Redundancy allows us to avoid a single point of failure. Flexibility allows for the online addition or subtraction of agents to the system. As each agent only has partial knowledge of the overall problem, effectively coordinating behavior in this domain can be challenging for a number of reasons, including:

- *Naturally distributed environments*. Environments such as sensor networks or robotic teaming where the set of operating agents are physically decoupled from one another removes the possibility of a centralized controller. This is especially true as the scale of the problem grows.
- *Unknown team composition*. While designers may know prior to release the types of agents they will be deploying into an environment, it may be very likely that they will not know the reality of composition after deployment is completed. Many of the sensors could be damaged during an airdrop deployment, or a certain subset of robots being used have power or actuator failures. Being the case, allocating certain agents to one task/role or another a priori is infeasible due to unknown failures or deployment methods.
- *Limited Computational Power*. As introduced in Chapter 1, when we are attempting to coordinate the behavior of many low-powered, low-computationally outfitted devices, lightweight solutions become necessary. Coordination must be efficient and economical.
- *Limited Time*. In environments where time is of the essence, the challenge of finding quick and sufficient solutions is paramount. In domains where the dynamics of the system can change the problem quickly, such as a target tracking domain with a moving target, fast solvers with sub-optimal behavior are acceptable.

### 2.1.4 Distributed $k$-coloring and its Applications

Many real-world applications exist that can be modelled using the DCOP framework, and in particular, the distributed $k$-coloring problem. These range from wireless sensor networks [3] to distributed planning and scheduling [8]. Below, we introduce the distributed $k$-coloring problem and provide two examples of how it can be framed to address real-world problems.

Given that solving for optimal DCOPs is known to be NP-Complete [29], empirical evaluation of solution techniques is necessary in order to evaluate performance. As certain practical problems such as target tracking and traffic channel allocation have been studied, there are a number of NP problems frequently used by researchers as benchmarks for the study of DCOP approaches, including boolean satisfiablity and distributed $k$-coloring, or graph coloring.

To begin, we formulate the $k$-coloring problem as a constraint satisfaction problem. Given a graph, $G$, of size $n$ nodes and $k$ possible colors, the problem is to decide whether the graph can be colored with no more than $k$ colors so that no two adjacent nodes are the same. In this problem, nodes are the variables, constraints are the edges, and colors are the variables domain. It is assumed that constraints are binary and of the type "not-equal". A satisfying assignment is a mapping of variables to value assignments that do not violate any of the constraints. A pair of adjacent nodes that are colored the same is a constraint violation. When no constraint violations exist, a solution is said to be satisfied.

To make the $k$-coloring problem that of an optimization problem, we recognize that not all problem instances may be able to be satisfiably colored in full either due to being over-constrained instances or due to a lack of sufficient time. Instead, we look to minimize the number of violated constraints, therefore maximizing global utility. This is referred to as *MaxSAT $k$-colorability*, or more simply, $k$-coloring. In the distributed case for $k$-coloring, it is assumed that each variable in the system is assigned to at most one agent. The agent is responsible for negotiating the color for its variable based on the constraints assigned to its variable.

Constraints can be defined such that they carry a certain amount of weight when violated. For example, if two adjacent nodes are blue, that could result in a weight of 10, whereas two adjacent red nodes results only in a weight of 1. In this way, we can represent many different types of preferences for assignment and generalize graph coloring to have a wider applicability.

### *Target Tracking*

Target Tracking is a vital problem for surveillance and monitoring applications that involves a set of sensors which are responsible for tracking or observing a set of targets in a given area. The goal of target tracking is to provide information about targets to a user about such things as location, velocity, etc. Each sensor in a target tracking application can have different modes which effect the value of the information about the target. Collaboration between sensors is crucial to provide the highest level of performance of the system.

To formulate the problem of target tracking, we can assume that there is an agent assigned to each sensor in the network. Sensors have different modes, potentially different capabilities, and perhaps different responsibilities. Constraints are typically defined between agents/sensors with overlapping sensing ranges. Constraints relate to the specific targets or areas of responsibility and could dictate such things as the number of sensors required for a target, the types or modes of sensing, and amount of time required to follow the target. The global function could be to maximize the number of targets tracked, maximize the accuracy of the tracked targets, or minimize the power consumed by the MAS as a whole. Successful examples of target tracking using DCOPs can be found in [78], [9], and [38], and as an explicit $k$-coloring problem in [32] where targets are colors and agents negotiate for the colors based on their neighbors assignments, capabilities, and current workload.

### *Role Allocation*

Role allocation can be viewed as a distributed $k$-coloring problem where agents in a team are required to periodically negotiate what role they will take on to carry out tasks or achieve a goal. Similar to our Chapter 1 example, constraints between individuals could be generated via geometric distance from one another, helping to distribute roles evenly among the set of all agents. In some cases, the joint-capability of two individuals taking on similar or contrasting roles in the team may need to be addressed, requiring the creation of a constraint between the two. In peer-to-peer wireless networks or Mobile Adhoc Networks (MANET) [28], different nodes may need to take on different roles, represented as a coloring, depending on the goals of the system. Determining which agents are allowed to enter sleep mode to reserve energy versus which agents are required to stay awake to maintain network connectivity could be negotiated via constraint reasoning. Role allocation as a distributed constraint optimization problem has also been used in Predator/Prey pursuit games [1].

## 2.1.5 Algorithms for Solving DCOPs

In this section we present a taxonomy for DCOP solution techniques and representative approaches from each which represent the state of the art. As stated, all these techniques apply to distributed $k$-coloring as it is a problem within DCOP. Given the breadth of types of applications that DCOPs can be formulated for, there are a number of types of approaches developed for solving them. In this work we would like to adopt a similar taxonomy to that found in [10]. We break the area of DCOP solutions into 2 main classes: (i) complete solutions, (ii) iterative approximate solutions, shown in Figure 2.3. Even within the class of complete solutions, we can still find a further dichotomy within these algorithms, classifying a complete solution as either synchronous or asynchronous during execution. Within the class of incomplete solvers, we find that there may be a further breakdown of solution types to that of local search approaches and learning approaches. We detail many current approaches in this section, however this is not a complete list. Our focus is to introduce many of the popular approaches, focusing on the approaches in local search, as that is where our work belongs.

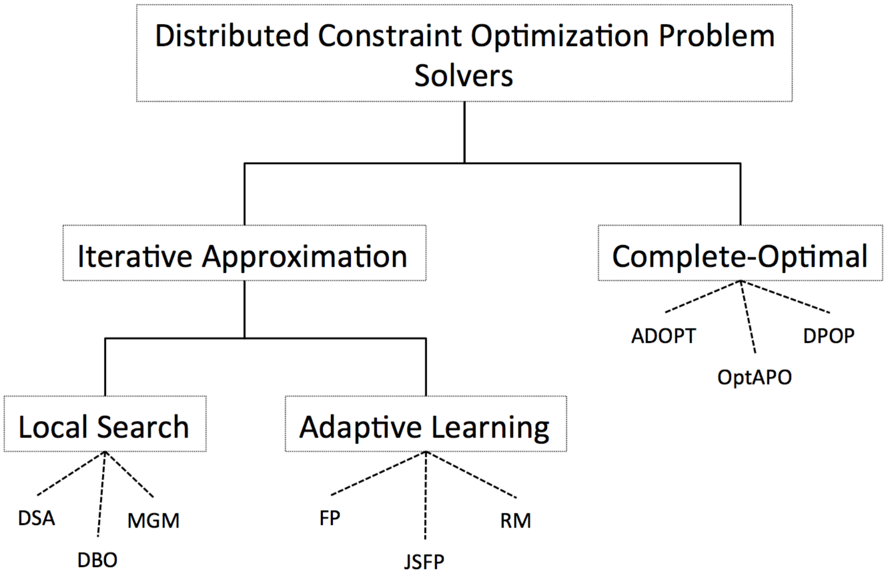


Fig. 2.3: A taxonomy of DCOP algorithms. In the diagram, DSA = Distributed Stochastic Algorithm, DBO = Distributed Breakout Algorithm, MGM = Maximum Gain Messaging Algorithm, FP = Fictitious Play, JSFP = Joint-Strategy Fictitious Play, RM = Regret Matching, ADOPT = Asynchronous Distributed Optimization Algorithm, OptAPO = Optimal Asynchronous Partial Overlay Algorithm, DPOP = Dynamic Programming Optimization Algorithm.

### *Complete Solutions*

A complete solution to a DCOP involves always searching for the optimal solution to the problem at hand. Finding the optimal solution for a DCOP is an NP-Hard problem, which can be seen by reducing the problem to the 3-colorability decision problem on a graph; known to be NP-Complete [14]. While these techniques are theoretically important and compelling, from a practical point of view, the exponential increases in computation and communication quickly emerges as the size of the problem increases. That said, they are still an important area of research when enough time and resources exist to dedicate to finding a complete solution.

Many approaches exist in the literature with very nice theoretical guarantees for providing complete, optimal solutions. Dynamic Programming Optimization (DPOP) [49], Asynchronous Distributed Optimization (ADOPT) algorithm [38], No-Commitment Branch and Bound (NCBB) [12], and Asynchronous Forward Bounding (AFB) [22] all provide optimality guarantees and allows for agents to operate asynchronously. What this means is that agents do not need to wait for messages from other agents in order to continue executing.

ADOPT performs distributed backtracking search using a best-fit strategy. Each agent assigns the best value it can find given local information. In order for ADOPT to run, the constraint graph must be re-organized into a depth first search (DFS) tree. Polynomial time algorithms exist for computing this structure [48] and many complete search algorithms use them. The key ideas behind ADOPT are (i) keeping a lower bound estimate of solution quality, (ii) backtracking thresholds, and (iii) upper bound estimate of solution quality. Each agent maintains lower and upper bound estimates of their local solution quality. Once these values are equal, the search ceases. Backtracking thresholds can be used to introduce error bounds into solution quality, allowing for sub-optimal results to be returned by the algorithms. Several advances to ADOPT have been developed, including BnB-ADOPT [73] and ADOPT-ng [65].

The DPOP algorithm is a dynamic programming based approach that applies equally well to constraint networks as well as more general graphical models such as Bayes nets and Markov random fields. The DPOP algorithm also utilizes a DFS tree and works in three phases: (i) the variable arrangement into a DFS tree, (ii) propagation of a utility message from leaf nodes to the root node, and (iii) propagation of a value message from root nodes to leaves. In phase (i), the current utility of the agents is propagated upwards, that includes both the current utility of the local assignments and the desired value assignment of each agent. Once all messages are propagated up, phase (ii) involves the root node computing a more desirable solution, in an attempt to minimize the violated constraints taking into account the information obtained from the children nodes. This calculation is downward propagated to the children in phase (iii), the value assignment phase. This process continues until an optimal assignment is determined. Like ADOPT, there have been many extensions developed from DPOP, including Open DPOP [47] and MDPOP [51], a DPOP algorithm that takes into consideration social choice theory and Vickrey auction mechanisms.

### *Iterative Approximate Solutions*

The second class of algorithms, iterative best response type approaches. Finding optimal solutions for constraint networks is an NP-Hard problem. Therefore, in many cases, the complexity of using a complete solution solver is not practical for real-world applications. Approximate algorithms are often preferred methods which generally scale well to large distributed applications. We term these approximate approaches as iterative best response after the framework developed in [10] which attempts to unify the DCOP literature with game theoretic literature.

The trade off in using best response approaches is in sacrificing optimality for a reduction in communication and computation overhead. Although typical results of many iterative best response algorithms perform quite well there exist pathological instance of DCOPs where either we cannot put guarantees on algorithmic behavior or the algorithms just perform plain bad. Regardless, these approaches are an important area of study, especially in time and resource constrained environments.

Most every approach in this class starts with a random assignment for all variables and performs a series of local moves in an attempt to optimize the global objective function. In algorithms that are considered greedy search, the process involves agents changing a small set of variables in a local neighborhood with a new assignment that is closer to the min/max of the objective function. This is typically referred to as *gain*. The process stops once there is no local move that provides a positive gain. These approaches require very little memory or computation power and work in a variety of settings. Approaches such as Stochastic Local Search, walkSAT [64], and simulated annealing are considered greedy search approaches. Most approaches in this area utilize a random restart of randomized steps to avoid local min/max.

The main issue with greedy approaches is in the fact that these type of algorithms assume that they are making changes to the problem while the local state remains the same. If all agents have the same belief, then greedy algorithms can lead to very chaotic behavior and negative gain can be the result. This is referred to as the incoherence problem. To address this problem, algorithms have been developed that introduce a stochastic element to the decision process. The Distributed Stochastic Algorithm (DSA) [18] is one such algorithm that has been extensively studied in the literature and experienced great success as a popular benchmarking technique. DSA initializes with a random state DCOP and proceeds into an infinite loop. At each execution step, each agent calculates the best gain it can achieve. Each DSA agent holds an *activation threshold*, $\alpha \in \Re$, that is compared against a randomly generated number at each step, $r \in \Re$. If $\alpha < r$ then the agent will make the change to its variable. Otherwise, the agent will due nothing. This simple addition helps to address the negative gain problem by reducing the number of changes that occur in the environment. As said, empirical results using DSA typically show monotonically increasing solution quality during the execution time, but there are no theoretical guarantees on DSA's performance as an anytime algorithm, or one that can be stopped at anytime and provide the best-to-date solution.

An alternative to DSA, which in addition to solving the coherence problem also provides theoretical guarantees on anytime behavior, is the Maximum-Gain Messaging (MGM) algorithm [46]. MGM works by having agents in a local neighborhood agree on which of them should change their value at each step. MGM is an extension of the Distributed Breakout Method (DBO) [74]. DBO works using weighted edge constraint graphs. If a violation occurs between two variables, that constraint edge is incremented by 1 and agents compute maximum gain based on the sum of the weighted edges. The problem with DBO that MGM addresses is outdated knowledge. Constraint edge weights accumulate over time in DBO and lead to inaccurate estimations on gain. MGM does not consider edge weight, resulting in more coherent behavior. Unlike DSA, MGM is considered anytime as it acts more as a hill climbing approach than stochastic search. MGM does not rely upon a parameter setting of $\alpha$, but does require much more overhead in message passing due to the exchange of potential gain of each agent shared with its neighborhood. Empirical results show comparable performance between DSA and MGM [44].

### *Metrics for Evaluating DCOP algorithms*

Algorithms for solving DCOPs are evaluated with three primary metrics: *solution quality*, *messages passed*, *cycles used*. The global cost in Equation 2.1 of a potential solution is the solution quality metric. We note this as it is the way most ofter referring to the minimization or maximization of Equation 2.1, and is the primary metric for evaluating a DCOP solvers effectiveness.

To obtain solutions for a DCOP, agents must interact with one another through message passing. Every time an agent changes its variable assignment, it must communicate that information to all its neighboring agents. We adopt the accepted definition of a cycle for DCOPs, as introduced by Yokoo [75], such that during each cycle an agent:

1. Processes newly received messages
2. Performs local computation
3. Sends messages to other agents if necessary

Almost every analysis of a DCOP algorithm includes an evaluation of the performance of that algorithm in terms of number of messages passed in the system or per agent as well as the number of cycles taken to find a solution of given quality. Minimizing messaging and cycle counts directly correlates with algorithmic efficiency, but does not necessarily equate to good solution quality. Other metrics can be used to evaluate DCOP solver performance, including memory use and computation time in the local computation portion of the cycle above.

## 2.2 Coalition Formation

In this section we present a review of coalition formation and its game theoretic origins as well as its relevance to multi-agent systems (MAS). Coalition formation has been studied extensively in the context of economics, social interactions, political party formations, organized crime, etc. While the literature on the topic of coalition formation is vast, we will narrow our view to those pieces which we believe pertain to our goal of groups solving DCOPs. Coalition Formation can be generalized as a Complete Set Partitioning Problem (CSPP) [72] where agents attempt to discover disjoint coalition allocations that maximizes the reward of all coalitions. Many traditional approaches have been developed to deal with finding exact and approximate solutions to the NP-hard problem of CSPP, but rely on techniques where the complexity is exponential in the number of agents and the algorithms are designed in a centralized manner. The problem of solving a DCOP is naturally distributed and typically large scale; both of which present problems for CSPP solution techniques. As such, we consider a more flexible and localized approach to forming coalitions using game theory.

There are two distinct branches of game theory; cooperative and non-cooperative. The main distinction between the two branches is in the level of analysis. In cooperative game theory, the focus is on a set of agents who have come together to form a group while its non-cooperative counterpart focuses on the analysis of the individual and the full set of actions available to them. Non-cooperative game theory is the study of strategic interactions between two or more agents. Non-cooperative games, or simply games, are analyzed by using solution concepts. Solution concepts, such as the Nash equilibrium, provide a way to describe what strategies will be used by the agents in a game. When used to study the coalition formation process, the focus shifts from equilibrium to stability, although they are essentially the same concept. Stability is used in settings where individuals will be negotiating for admittance into a group and the analysis is based on the coalitions that could form and deviations between them. Analysis in cooperative game theory is focused on the group, not the individual.

In the next section we provide a general framework for coalition formation followed by a discussion on the types of coalition formation techniques discussed in the literature with a focus on work relating to coalition formation with DCOP research.

### 2.2.1 General Framework

There have been many proposed frameworks for modelling the coalition formation process. We present a model closely following that in [58] Typically, the main components of the model include the following:

- A finite set of *players* or agents, $N$, which is typically referred to as the *grand coalition*
- A compact set of *states*, $X$
- A (possibly) infinite set of time, $t = 0, 1, \ldots$
- Initial state, $x_0 \in X$
- A payoff function, $u_i$ for each $i \in N$ defined over each $x \in X$
- For each pair of states $x$ and $y$, a collection of coalitions $E(x, y)$ that are *effective* at moving from $x$ to $y$
- A set of existing coalitions, $S$
- A set of potential partners, $P \subseteq N \setminus S$

Existing coalitions may propose, or be proposed to, by the potential partners in the system. These partners are usually referred to as *free* or *non-committed* agents. In addition to the above components, there are other formal properties that are useful for various models of the coalition formation process. A protocol, $p$, represents a probability distribution over the systems "choice" of active coalitions at each time step $t$. As our work will employ active negotiating agents, we do not consider a system model of $p$. A response order for every set of $Q \subseteq P$, where $Q$ is the set of agents presented with a proposal, is used to determine whether or not the system *moves* from state $x_t$ to state $x_{t+1}$. It is normally assumed that if just one agent in $Q$ rejects the proposal, the system does not move during that time step. Movement in a coalition formation process is referred to as $m_t = (x_t, S_t, Q_t)$. And lastly, we can keep track of a *history*, $h_t$, of moves that have taken place to date. Histories maintain the sequence of state spaces that the coalition formation process has moved through and are similar to that of a Markov Chain. State space information can be as simple or complex as needed, but should minimally contain the coalitions that exist at each time step, and probably the relationships that exist between those coalitions (if any).

We can represent the *end state* of a system as $E(x, y) = \emptyset$, meaning that there is no transition to another state possible in the system. Some coalition formation processes have no end state due to the possibility of renegotiation, coalition reformation, and expiration of agreements. The process of forming coalitions then is the process of moving from one state to another either ad infinitum or until an end state is discovered.

### 2.2.2 Coalition Formation Related Work

There are many approaches to coalition formation. In the DCOP literature, this is referred to as partial centralization. In game theory, it is referred to as coalition formation games or cooperative games. There has been some work in using non-cooperative game theory for forming coalitions as well, and we give some discussion on that as well.

The Asynchronous Partial Overlay (APO) [35] and Optimal APO (OptAPO) [34] are two algorithms that work to solve DCOPs by partially centralizing pieces of the problem via the use of cooperative mediation. The process of centralizing the problem into subsets of the original problem allows for increases in efficiency to finding the solution. The use of *mediators* as the subsets problem solver is used to represent the group with a shared utility function and negotiate with other mediators. Agents are allowed to change groups and be represented by different mediators over time. This process can continue until much of the problem is centralized, and utilizes either a complete solver (in APO), or modified version of such (in OptAPO), to solve the subproblem. The solution technique and coalition generation are tightly coupled, as both rely upon each other. Our work focuses on using existing solutions and forming coalitions without the need to tailor existing solution techniques, creating a strict de-coupling of solution and coalition formation. The work of [67] discusses a similar process and cites its work as very similar in nature to cooperative mediation.

Non-cooperative game theory has been used as a negotiating basis for developing approaches to coalition formation. As non-cooperative games are the study of strategic interactions between individuals, they have been most popularly studied as a bargaining game [11]. The process of bargaining can be seen as the proposing and acceptance/rejection of agents to join a group. The use of bargaining has been studied extensively in economics and political science in works such as [41] and [23].

The work of Chapman in [10] reformulates the the problem of a DCOP as that of a non-cooperative potential game on hypergraphs. Chapman attempts to bring to light the fact that solving a DCOP and solving for a potential function in game theory equate to the same process. He discusses the lack of a common vocabulary between the economics and computer science research communities and highlights how much of the work in non-cooperative game theory and DCOP is the same with different terminology. Chapman develops a framework for analyzing approaches to solving potential games and DCOPs for single, peer-to-peer, strategic interactions.

In cooperative game theory research on coalition formation games, the use of a standard value function, or characteristic function, is typically employed. The value function assigns a real-valued payoff to each each agent for each possible coalition that can form in the environment. Works such as [57], [55], and [62] employ this technique. This type of work assumes that the value function can be derived ahead of time and that the different combinations of individuals working together can be quantified.

There has been some scattered pieces of work, aside from Chapman [10], that have looked at the relationship between DCOP and coalition formation games. There has been quite a bit of research regarding k-optimality [45][43], t-optimality [30], and c-optimality [69]; a family of optimality measures where the joint action space of agents is explored in terms of guaranteeing solution quality within some bounds $\delta$ in terms of size ($k$), time ($t$), and space ($c$). This family of optimality measures give guarantees that a solution is locally optimal if no group of agents can change their actions unilaterally to increase the local utility of that group. From these optimality measures came the C-DALO algorithm [69] where groups of neighboring agents are formed and a selected leader agent attempts to solve for the subproblem within the bounds of $\delta$. The C-DALO algorithm is more computational complex than iterative best response type algorithms, using sophisticated locking techniques and data structures to allow for agents to exist in multiple groups at the same time.

There have been approaches developed for performing coalition formation based on the results from solving DCOPs. In both [68] and [42] the works utilize the optimal solutions found by running various DCOP solution solvers to define the optimal coalition structures for team environments. While related, this work is using DCOPs to generate coalitions as opposed to using coalitions to solve DCOPs.

Rahwan et al [56] propose the introduction of constraints between agents to bound the search for suitable coalitions. As is accepted in traditional coalition formation and the NP-Hard problem of SPP, every possible subset of agents needs to be considered in order to find a solution of coalitions that will form. By introducing constraints to the problem, they reduce the number of possible coalitions that need to be analyzed. Introduced is a new model for studying coalition formation called Constrained Coalition Formation (CCF) where restrictions are placed upon certain agents from working together, similar to the model of constraints placed between DCOPs. It is possible that we can utilize the CCF model in our studies.

### *Other Work Related to Coalition Formation*

Partial centralization and coalition formation have been shown to be a powerful tools for solving complex instances of DCOPs. DPOP [49] is a partial centralization algorithm that works by first reformulating the constraint graph as a depth-first search tree in which adjacent nodes in the original graph fall into the same branch of the tree. DPOP suffers from over-centralizing the problem and creating messages that are far larger than necessary. To remedy this, the authors developed PC-DPOP [50] which bounded the centralization and increased performance. Once cluster nodes have formed, both DPOP and PC-DPOP use centralized algorithms to solve their problem partitions. OptAPO [34] is a partial centralization approach that utilizes a process called cooperative mediation. Partitions of cooperating agents are represented by a mediator agent that uses a centralized technique to solve their problem partition. Since partitions are not unique, additional work is done due to possible partition overlap. Another partial centralization algorithm is the VSR-DSA algorithm [20] in which the authors investigate using stochastic methods, but rely too heavily on partially centralizing based on one specific structure that is identified via frozen pairs discussed previously. This results in poor performance on sparse graphs. Related to partial centralization is the coalition formation research for DCOPs. There has been much work in $k$-optimality [45], t-optimality [30], and c-optimality [69]; a family of optimality measures where the joint action space of agents is explored in terms of guaranteeing solution quality within some bounds $\delta$ in terms of size ($k$), time ($t$), and space ($c$). This family of optimality measures give guarantees that a solution is locally optimal if no group of agents can change their actions unilaterally to increase the local utility of that group. From these optimality measures came the C-DALO algorithm [69] where groups of neighboring agents are formed and a selected leader agent attempts to solve for the subproblem within the bounds of $\delta$. The C-DALO algorithm is more computational complex than stochastic search algorithms, using sophisticated locking techniques and data structures to allow for agents to exist in multiple groups at the same time. With the exception of [20], we differentiate from all these pieces of work in that we are using an algorithm which is both distributed and stochastic.

# CHAPTER 3

# VIRTUAL STRUCTURE REDUCTION ON CONSTRAINT GRAPHS

In this chapter, we introduce our Virtual Structure Reduction algorithm (VSR). The VSR algorithm is the first distributed algorithm that allows for the search and discovery of *frozen pairs* of agents, or agents which are guaranteed to have the same variable assignment in a solved instance of a DCOP. Agents independently discover their frozen pair relationships and are given the opportunity to form a coalition with one another. The coalitions that are formed represent a single bargaining unit within the DCOP, and are represented by a single agent, termed a *surrogate* agent. Surrogate agents are given the responsibility by the coalition to negotiate the variable settings for all agents within that coalition. As coalitions form, the structure of the problem is changed to reflect the new negotiation channels between surrogate agents and agents which the coalition has constraints with.

We integrate the VSR algorithm with a modified version of the Distributed Stochastic Algorithm (DSA) and show that by using coalitions of agents we can improve efficiency in terms of message passing, cycle consumption, solution quality, and time to solution on randomized instances of the $k$-coloring problem against two successful iterative approximate algorithms, DSA and MGM (see Chapter 2.1.5 for a review). Our positive empirical results are a direct result of the coalition formation process, and how it affects the constraint graph structure. *Constraint density*, or the ratio of the number of constraints to the number of variables, has a significant effect on the difficulty of distributed problem solving. As the density of problem instances increases, a phase transition occurs from under-constrainedness to over-constrainedness, resulting in an easy-hard-easy behavior of search for solutions. In the middle of this phase transition, problems tend to be most difficult to solve. Reducing density equates to reducing problem difficulty and we report results pertaining to how the formation of coalitions leads to a reduction in active constraint graph density and the overall reduction in problem size.

The remainder of this chapter is organized as follows: First, we discuss the concept of frozen variables, and provide definitions we will be using, highlighting some important properties of frozen pairs. We then detail the VSR algorithm and VSR-DSA integration used to communicate structure and negotiate control between variables. Empirical results with regard to the structural changes of the problem are presented showing reductions on constraint graph active density and active variables. Further in our evaluation, we provide results demonstrating the performance increase using VSR-DSA compared with DSA-B and MGM algorithms.

## 3.1 Frozen Pairs

The intuition behind frozen pairs is straightforward; two variables that must have the same value in all satisfying assignments of a given problem instance. For example, if one was to find every set of satisfying assignments for the 3-coloring problem shown in Figure 3.1, they would find that no matter what, variables $x_1$ and $x_2$ would always have to be the same value.

The concept of frozen pairs, first introduced by Culberson and Gent [15], provides us with a new perspective on strategies for solving DCOPs with variables that share the

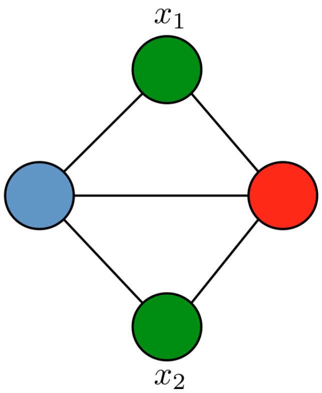


Fig. 3.1: Simple example of a frozen pair between $x_1$ and $x_2$. Every satisfiable value assignment for this problem results in these two variables having the same color.

same domain. Culberson and Gent were studying the phase transition region of $k$-coloring problems and its relationship to finding hard to solve instances within that region. This easy-hard-easy phase transition has attracted much attention and has been documented by many researchers [39, 54, 66]. Culberson and Gent identified frozen pairs through a centralized process of adding constraints and detecting inconsistencies (unsolvable problem instances) that occurred as a result of the newly added constraint. If the violation could be resolved only by making the pair of variables the same, those two variables were deemed frozen.

In addition to Culberson and Gent pointing out that there exists an intimate relationship between certain variables in $k$-coloring problems, Cheeseman et al [13] pointed out a similar relationship in his work. Studying the varying degree of difficulty in $k$-coloring and $k$SAT problems, Cheeseman suggested that reducing a graph based on identifying nodes that shared a $k-1$ clique would be beneficial for finding solutions while not affecting the original problem given the proper bookkeeping. He also pointed out that the sharp phase transition from fully satsifiable instances to unsatisfiable instance directly correlated with problem difficulty.

Both Culberson's and Cheeseman's work were foundational in the area of $k$-coloring for centralized constraint satisfaction. No known distributed algorithm for finding frozen pairs exists which could be utilized for solving DCOPs online. Our work is novel in the respect that we are able to identify and effectively utilize the property of frozen pairs in a distributed fashion. Our work is also the first to utilize an iterative approximate algorithm for finding solutions for coalitions of agents, sometimes referred to as partial centralization.

When a pair of variables must have the same assignment, either agent in that pair can negotiate individually with the members of their shared neighborhood and provide a satisfying assignment to the other frozen variable. Having one agent negotiate on behalf of two or more agents (i.e. a coalition) eliminates a significant number of messages and cycles that would otherwise be wasted "competing" between variables for satisfying assignments within their shared neighborhood. As a result, efficiency gains can be immediately realized. But another byproduct of forming coalitions between frozen pairs is that the resulting reduced graphs have a lower *constraint density*.

As introduced in Section 2.1, constraint density is the ratio of the number of constraints to the number of variables in a problem, $\mathcal{D} = \frac{|F|}{|X|}$. Low constraint density problems are more likely to yield a satisfiable, or closer to satisfiable, solution from an initial random assignment as opposed to high constraint densities. Utilizing the concept of frozen pairs can allow us to lower the number of active negotiating agents in the problem, which we have empirically found to also reduce the effective constraint density of the problem. For both Distributed Constraint Satisfaction Problems (DisCSP) and Optimization (DCOP), lower densities typically lead to problem instances that are easier to solve.

## 3.2 Definition of Frozen Pairs

A binary constraint problem $\langle X, D, F\rangle$ can be represented as a *constraint graph* of the form $G = \langle V, E\rangle$, where each vertice $v_i \in V$ represents a variable, $x_i \in X$ with domain $d_i \in D$, and each edge $e_j \in E$ represents a binary constraint $f_j \in F$. For more detailed information, we refer the reader to Section 2.1.

Let $\mathcal{N}$ be a neighbor function, such that $\mathcal{N}(x_i) = \{x_p | \exists f(d_i, d_p) \in F\}$ and $\mathcal{S}$ is the shared neighbor function, such that $\mathcal{S}(x_i, x_j) = \mathcal{N}(x_i) \cap \mathcal{N}(x_j)$. We define $x_i$ to be *frozen* with $x_j$ iff $\mathcal{S}(x_i, x_j)$ contains a maximal (k-1)-clique. For domains of size $k$, $x_i$ is frozen with $x_j$ iff $\mathcal{S}(x_i, x_j)$ contains a (k-1)-clique and $\mathcal{S}(x_i, x_j)$ does not contain a $k$-clique. In the event that a $k$-clique is detected, the problem can be deemed unsatisfiable as we will show below. Note that we use the terms "node", "agent", and "variable" interchangeably unless it is necessary to differentiate between them.

**Definition 1.** *Given a constraint graph G, two nodes, $x_i \in G$ and $x_j \in G$ are a* ***frozen pair****, $FP(x_i, x_j)$, iff there exists a maximal $(k-1)$-clique within the shared neighborhood $\mathcal{S}(x_i, x_j)$ where $\forall x_p \in \mathcal{S}(x_i, x_j), [f_i(d_i, d_p) \equiv f_j(d_j, d_p)]$ and $d_i \cap d_j \neq \emptyset$.*

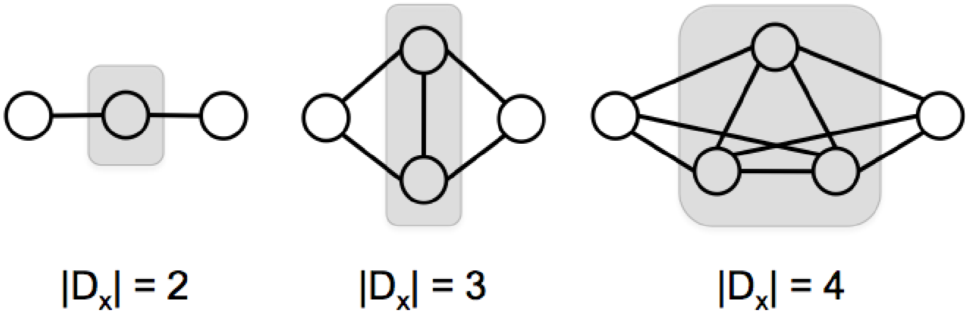


Fig. 3.2: Examples of frozen pair structures in graph form, with the core in grey, for domains of size 2, 3, and 4. Frozen pairs are variables that must have the same assigned value in a solution when they share the same neighbors of a core.

If $x_i$ is frozen with $x_j$, then we refer to the ($k$-1)-clique in $\mathcal{S}(x_i, x_j)$ as the *structural core* of that frozen pair. In Figure 3.2, we highlight the structural cores that enable frozen pair detection. A single variable may be in multiple cores or be frozen with another variable. When frozen, $x_i$ and $x_j$ will have, at most, one possible value after the members of the core have been given their assignments. This property ensures that any agent who is frozen can negotiate a local assignment with the members of the core and share that assignment with the other frozen agent. A variable that is frozen to another variable has a value that can be induced by the assignment of the core.

**Proposition 1.** *When two nodes are a frozen pair, they must have the same variable assignment.*

*Proof.* Assume there exists two nodes, $x_i$ and $x_j$, such that $FP(x_i, x_j)$ and $|D_i| = |D_j|$. Let $k = |D_i|$. Then a $k$-clique is the largest fully connected subgraph of $G$ such that there exists a variable assignment that is a satisfiable solution. By definition, the frozen nodes $x_i$ and $x_j$ share a $(k-1)$-clique. Let the $(k-1)$-clique shared by $x_i$ and $x_j$ be labelled $P$. Assume that each variable in $P$ has a value assigned such that $P$ is locally satisfied. Now consider the set of variables $P'$, where $P' = P \cup \{x_i\}$. There must exist only a single value for $x_i$ in $P'$ such that $P'$ is a local satisfiable solution. This is induced by $|P'| = k$ and that the other $k-1$ values had already been assigned in $P$. Now consider the set of variables $P''$, where $P'' = P \cup \{x_j\}$. Again, there must exist only a single value for $x_j$ in $P''$ such that $P''$ is also a local satisfiable solution. As $x_i$ and $x_j$ both share $P$, the single value assigned to $x_i$ and $x_j$ must be the same. □

If $x_i$ is frozen with $x_j$, $x_i$ and $x_j$ both have constraints with all of the members of their shared core. We can allow $x_j$ to negotiate with the core as a *surrogate* for $x_i$ (or vice versa). In other words, once the core has found an assignment that is consistent with $x_j$, $x_i$ will have the same exact variable assignment. By allowing $x_j$ to complete the negotiation process on $x_i$'s behalf, we can reduce both the number of messages and cycles necessary to find a solution. The reduction of messaging is a direct result of there being a lower number of negotiating agents in the search process, while the reduction of cycles is due to there being a smaller search space. Because there are a lower number of negotiating agents, there are less occurrences of “thrashing” that occur, or variable settings that are not consistent. Consider the case of three agents, $x_1$, $x_2$, and $x_3$, such that a constraint exists between $x_1$ and $x_2$, and a constraint exists between $x_2$ and $x_3$. In a $k = 2$ coloring problem, $x_1$ and $x_3$ would be frozen. It is more likely that conflicts occur during the search for a solution to the problem with both agents $x_1$ and $x_3$ negotiating with $x_2$ than if only one of the two agents were to negotiate with $x_2$. We define the set of all of an agent $x_i$'s frozen variables as its *frozen list*, $F_i$. Once an agent has become a surrogate for another agent, or set of agents from $F_i$, it stores those agents' IDs in a list called its *coalitionList*.

In addition to providing a detection of frozen pairs, our search process can allow agents to detect and report the occurence of unsatisfiable instances if it is so desired by the designer of the system.

**Proposition 2.** *If an agent detects a $k$-clique during the search for frozen pairs, the problem instance is unsatisfiable.*

*Proof.* The proof follows from recognizing that in a satisfiable instance, a $k$-clique has exhausted each possible domain value for the agents involved. If an agent searching for a frozen pair discovers a $k$-clique to which it has a connection to each member of that $k$-clique, there exists a clique of size $k + 1$, and there is no possible way to satisfy the instance. $\square$

Being able to recognize an unsatisfiable instance of a $k$-coloring problem instance is desirable in certain cases SAT problems with hard constraints (i.e. no violations allowed). Standard methods of iterative approximate algorithms do not have a utility for recognizing unsatisfiable instances. In our integration of the VSR with DSA, we can now allow that to be done. In addition to simply identifying unsatisfiable instances themselves, we also allow for the detection of unsatisfiable cores [63], or the regions of a problem that are causing the instance to be unsatisfiable. Recognizing not only that an instance is unsatisifable but where that instance is unsatisfiable is important to researchers in model checking, debugging logic, generating certificates of unsatisfiability.

## 3.3 VSR-DSA

The Virtual Structure Reduction DSA (VSR-DSA) algorithm is a two phase algorithm consisting of *coalition formation* and *assignment*. It is a 3-state cyclic process for identifying surrogates and forming coalitions which terminates when no further frozen pairs are found. An entire series of the 3-state process of coalition formation requires 3 cycles (i.e. each state receives messages, processes and computes, then sends messages) and a varying number of messages.

The coalition formation phase consists of the following three states that are executed, in order, by each active agent:

1. Targeted Local Structure Sharing
2. Frozen Pair Discovery
3. Collapse

These states are executed locally by agents that are currently *active*, with all agents being initialized with *active* = *true*. When an agent is not active, it is because that agent has joined a coalition and has relinquished control of its variable assignment to the surrogate agent leading the coalition. All inactive agents are maintained in the surrogate agent's *coalitionList* which is updated in Algorithm 2 when a 'collapse' message is received.

### 3.3.1 Example

In Figure 3.3 we illustrate the process of the coalition formation phase on a constraint graph. Frozen pairs are indicated by dashed connections. Active surrogates are indicated by the darker grey coloring. We can see that once an agent has relinquished control of its variable assignment, the resulting graph can actually be reduced, or folded, because the connections of all agents in a surrogates coalition list have become its own. Since there are shared constraints between the frozen pairs, edges and nodes are "virtually collapsed" on top of each other resulting in smaller, and typically less dense constraint graphs. The illustration provided is solely for the purpose of demonstrating what is logically occurring. While there is a manipulation of the constraints and variables going on through the coalition formation process, it does not change the evaluation of the original problem. Once a solution is found, the original constraints and variable relationships are used to determine the quality of the solution.

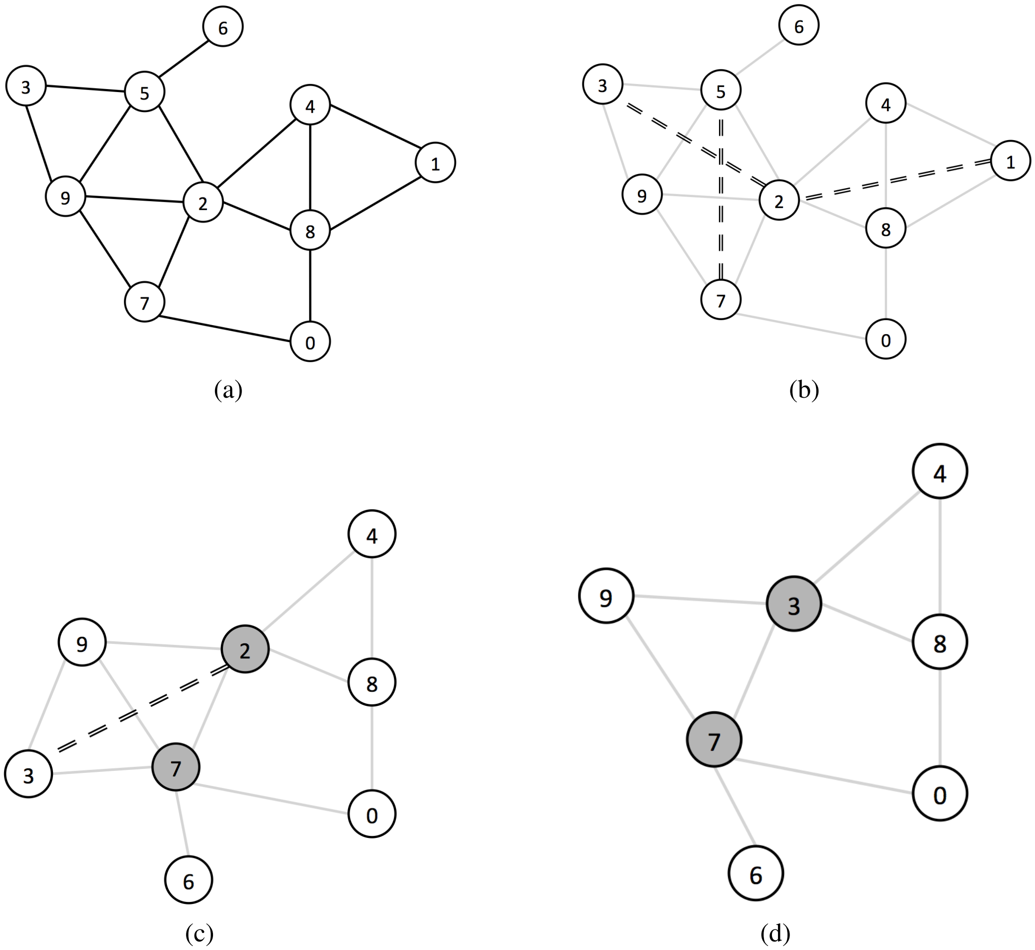


Fig. 3.3: Illustration of VSR on k-coloring problem, k=3. (a) Constraint graph in original form. (b) Identification of frozen pairs indicated by dashed lines. (c) Grey nodes indicate surrogate agents. (d) Reduced constraint graph.

In Figure 3.3a we show a 10-variable binary constraint graph in original form with a density of 1.4. In Figure 3.3b, 3 frozen pairs have been identified. We see the resulting graph from the first full series of the coalition formation phase in Figure 3.3c. Agent$_2$ has become the surrogate for agent$_1$ and agent$_7$ is the surrogate for agent$_5$. Figure 3.3d shows the second full series of the coalition formation phase where agent$_3$ has become the surrogate for agent$_2$. At this point, there are no more cores and coalition formation is complete. Note that agent$_3$ was moved to agent$_2$'s position in the illustration, but the positioning is inconsequential to the resulting graph structure. The resulting constraint density of the graph in Figure 3.3d is approximately 1.29.

### 3.3.2 Algorithmic Description

The VSR coalition formation technique is integrated with the DSA solver and presented as the VSR-DSA in Algorithm 1. Following the coalition formation phase, illustrated above, the *assignment phase* takes place where active agents negotiate a value for their variables and can be done similar to other DCOP solvers, but leveraging the surrogate assignments provided by the coalition formation phase.

It should be noted that the use of the DSA algorithm with our VSR coalition formation technique is only one possible candidate for integration. Other iterative approximate algorithms, such as the MGM algorithm, could be considered as candidates for integration with the VSR coalition formation process as there is a clear delineation with the assignment phase. While the prospect of integration with other solvers is high, it will always require some modification to the solver itself, but not at the risk of changing the foundation of the chosen algorithm. Results we present in the following section suggest that any existing algorithm could benefit from utilizing VSR either as a full integration or a pre-processing method. If used as a pre-processing method, Algorithm 1 could simply be abridged by ending at line 24 and having no assignment phase.

### 3.3.3 Algorithmic Details

At the beginning of each cycle, agents process all new messages sent to them from the previous time step via Algorithm 2. It is in this algorithm where local changes are collected, including changes to an agent's neighborhood, an agent's neighbors neighborhood, the agent's coalition list, and the local variable assignments. The types of messages sent between agents are:

- *local structure* - informing a receiving agent about the sending agent's neighbors and constraint types
- *collapsing* - informing a receiving agent that the sending agent will be delegating authority to another agent (surrogate), thereby allowing the receiving agent to update its internal data structures
- *collapse* - informs a receiving agent that it has been given authority over the sending agent's variable, including the sending agent's local information (neighborhood)
- *assignment* - informs a receiving agent what the sending agent has selected as its new variable assignment

We now provide more detail on the coalition formation and assignment phases.

### 3.3.4 Targeted Local Structure Sharing

Starting on line 5 of Algorithm 1, all agents will send a 'local structure' message to their neighbors containing their local view. An agent's local view consists of their variable(s) domain and a list of the agents with whom their variable(s) have a constraint with. Sharing this information provides an extended view of the structure of the local problem to each neighboring agent. The state is switched to Frozen Pair Discovery for the beginning of the next cycle. Agents will only send local information to others if it is determined to be a necessary update. If an agent's local neighborhood has not changed, then there is no need to send a 'local structure' message to its neighbors. Additionally, if agents detect that a local state change also affects certain agents within its neighborhood (i.e. via a shared neighbor between the two sending an update), then the agent will only send 'local structure' updates to those agents which it deems could not have been part of another update it has received. This is necessary to update neighboring agents with information regarding new constraints or second neighbors.

**Algorithm 1** Sketch of VSR-DSA executed by agent$_i$

```
 1: {Given: set of neighbors, $\mathcal{N}_i$}
 2: call ProcessMessages() {Process all new messages}
 3: if active = true then
 4:     {Coalition Formation phase}
 5:     if state = Targeted Local Structure Sharing then
 6:         for each neighbor $j \in \mathcal{N}_i$ do
 7:             if local structure update necessary for $j$ then
 8:                 send ‘local structure’ message to $j$
 9:         state ← Frozen Pair Discovery
10:     if state = Frozen Pair Discovery then
11:         for each neighbor $j \in \mathcal{N}_i$ do
12:             Compute $\mathcal{S}_{i,j}$
13:         for each neighbor $j' \in \mathcal{N}_j$ do
14:             Compute $F_{i,j'}$
15:         if $|F_i| > 0$ then
16:             surrogate ← SurrogateSelection($F_i$)
17:         if canCollapse = true then
18:             for each neighbor $j \in \mathcal{N}_i$ do
19:                 send ‘collapsing’ message to $j$
20:         state ← Collapse
21:     if state = Collapse then
22:         if canCollapse = true then
23:             send ‘collapse’ message to surrogate
24:             active ← false
25:         state ← Local Structure Sharing
26:     {Assignment phase}
27:     if localChange = false then
28:         call DSA-S()
```

### 3.3.5 Frozen Pair Discovery

In this state, each agent $i$ processes the local views provided by neighbors to determine if a structural core exists. This process begins on line 9 of Algorithm 1 and proceeds by building the structural cores and storing them in $\mathcal{S}$. Recall that the structural core is the $(k-1)$ clique. Using the set of cores stored in $\mathcal{S}$, the agent determines the frozen list, $F$, by inspecting agents that are the neighbors of his neighbors. If any agent $j' \in N_j$ is fully connected to the same structural core as agent $i$, $j'$ is placed in $F$. In the event an agent discovers it is frozen with another agent(s) it must also determine whether or not it

**Algorithm 2** Sketch of ProcessMessages executed by agent$_i$

```
for each message in mailbox from agent $j$ do
    if message type is 'local structure' then
        localChange $\leftarrow$ true
        Update $\mathcal{N}(agent_j)$ and related domains
    if message type is 'collapsing' then
        localChange $\leftarrow$ true
        $\mathcal{N}(agent_i) \leftarrow N(agent_i) - agent_j$
    if message type is 'collapse' then
        localChange $\leftarrow$ true
        $\mathcal{N}(agent_i) \leftarrow N(agent_i) \cup N(agent_j)$
        $coalitionList_i \leftarrow coalitionList_i + agent_j$
        $coalitionList_i \leftarrow$ MERGE($coalitionList_i$, $coalitionList_j$)
    if message type is 'assignment' then
        Record local variable assignment change
```

will potentially become the *surrogate* agent responsible for another. This is decided via SurrogateSelection, Algorithm 3. Our first implementation of the VSR algorithm uses a lexicographical ordering based on agent ID to determine a surrogate selection between two agents. This is an accepted methodology in multi-agent systems and DCOP for differentiating agents. Looking forward, we will show in Chapter 4 that there are ramifications for making such naive selections of a surrogate, and in Chapter 5 we will show that there are improvements that can be made by performing smarter surrogate selection. When set to false, the *canCollapse* flag indicates that there is no possibility of becoming a surrogate of another lower ranking (via lexicographical identifier) agent.

**Algorithm 3** Sketch of SurrogateSelection executed by agent$_i$

```
{Given: set of frozen agents, $F_i$}
canCollapse $\leftarrow$ true
for each agent $j \in F_i$ do
    if ID$_j$ < ID$_i$ then
        canCollapse $\leftarrow$ false
if canCollapse = true then
    surrogate $\leftarrow$ arbitrary agent from $F_i$
```

When an active agent reaches line 18 in Algorithm 1 it checks if the *canCollapse* flag is set to true. If *canCollapse* is true, the agent sends a message consisting of the header string ‘collapsing’ to indicate the message type and the agent ID that will become its surrogate agent. This step is necessary to provide consistency between agents that neighbor each other and could both be possibly collapsing. If two neighboring agents collapse to two different surrogates, they will both need to update their local neighbors list before sending it forward to the surrogate agent. This update is performed in Algorithm 2, line 7. as agents receive new structure information and maintains consistency in the local structural knowledge of all agents.

### 3.3.6 Collapse

Active agents that enter the *Collapse* state will again check to see if the *canCollapse* flag is set to true. At this point, all neighbors of the agent know of its intent to collapse to a surrogate agent and have updated their local knowledge accordingly. All that is left is to send a ‘collapse’ message to the surrogate agent identified in Algorithm 3. Agents with the *canCollapse* flag equal to true send a message consisting of the header ‘collapse’ to indicate the message type, the list of its current neighbors, and its coalition list. Sending the coalition list is necessary as it is possible that the collapsing agent is acting as a surrogate for other agents and those agents will need to be added to the new surrogate’s coalition list. This agent now sets its *active* flag to false, and simply waits for a ’color’ message from its surrogate once a viable solution has been found. In short, after an agent sends a ’collapse’ message, it is no longer an active participant in the problem solving process.

### 3.3.7 Assignment phase

In the current implementation of the VSR-DSA algorithm, variable assignment is attempted in every cycle when no local structure changes have occurred by calling our minimally modified version of DSA, DSA-S, shown in Algorithm 4. A local structure change is detected by either a ‘local structure’, ‘collapsing’, or ‘collapse’ message. Any general DCOP solver could be used for this phase with modifications with respect to the assignment of surrogates. However, because this phase is local, a solver would also need to allow for surrogate assignment to take place after the local solving process has started. Otherwise, some additional form of a signal would be required to start the solving process after the coalition formation process is complete and all surrogates have been assigned.

**Algorithm 4** Sketch of DSA-S

```
{Given: domain $D$, activation probability $\alpha = 0.33$}
if there is a conflict of local variable(s) then
    Compute best possible conflict reduction, $\delta$
    $p \leftarrow$ random value from (0.0, 1.0]
    if $p < \alpha$ then
        Change variable assignment to $\delta$
    if new assignment != old assignment then
        Send neighbors new 'assignment' message
    if time elapsed since last local structure change $\geq \Delta$ then
        Send all agents in the coalitionList the new variable 'assignment' message
```

One could simply run the VSR algorithm as a pre-processing step and then run the DCOP solver of choice. While straightforward, this would lose some of the inherent and fundamental aspects of the distributed problem solving process and may not always be possible. In fact, our preliminary work in integrating VSR with the DSA solver shows that there is some benefit to integration over a strict delineation between coalition formation and assignment (see Section 3.5.3. Although the VSR algorithm could be used as a stand-alone pre-processing function for any DCOP solver, integrating it with more sophisticated complete optimal algorithms for a full online solution could be a challenge, if not impossible without major modification. Many complete optimal solvers assume that the problem is static throughout algorithm execution as they rely on backtracking and tree search. Inherent in the VSR approach is the possibility of a dynamic graph topology. Because an agent's view of the problem structure is changing, modified algorithms will need to take into consideration the possible lack of consistent structure from time step $i$ to time step $i + 1$. The DSA algorithm was a simple integration due to its probabilistic nature and non-assumption of continuity in the problem structure.

## 3.4 Properties of Frozen Pair Discovery

Here we discuss 3 basic properties of frozen pairs: Symmetry, Transitivity, and Induction.

**Definition 2.** ***Symmetry***. *Given constraint graph G and two nodes,* $x_i \in G$ *and* $x_j \in G$*, if* $x_i$ *is frozen with* $x_j$*, then* $x_j$ *is frozen with* $x_i$.

As we will discuss in Chapter 5, symmetry is an important property of frozen pairs as it allows us to prove stability of the coalitions that form under the frozen pairs relationship. As shown in Figure 3.4, $x_i$ would discover $x_j$ and vice versa in the VSR algorithm Frozen Pair Discovery state.

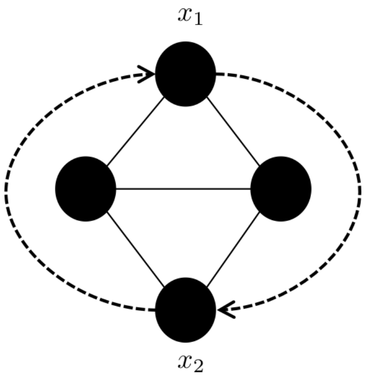


Fig. 3.4: Symmetry property of frozen pairs. Agents $x_1$ and $x_2$ will independently discover each other.

**Definition 3.** ***Transitivity***. *Given constraint graph G and two frozen pairs,* $FP(x_i, x_j)$ *and* $FP(x_j, x_k) \rightarrow FP(x_i, x_k)$.

As agents may be frozen with more than one agent at a time, pairs of agents that would not initially discover each other will do so via a shared frozen pair relationship. This property, shown in Figure 3.5, allows coalitions of agents to form that spans larger portions of the constraint graph. Agents will find transitive frozen pair relationships once a collapse occurs in the Collapse state of the VSR algorithm, and another search for frozen pairs begins.

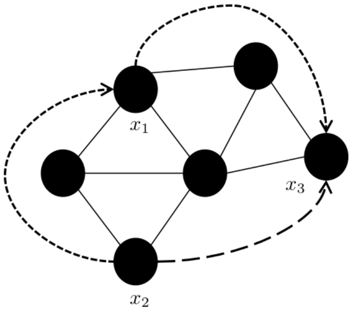


Fig. 3.5: Transitive property of frozen pairs. Agents $x_2$ and $x_3$ do not initially recognize each other, but will end up in the same coalition with $x_1$.

**Definition 4.** ***Induction***. *Given constraint graph $G$ and a frozen pair, $FP(x_i, x_j)$, a new frozen pair will form between two nodes, $x_w$ and $x_u$, given they share a constraint with the frozen pair, $FP(x_i, x_j)$, and share a constraint with another node, $x_z$, such that $x_z$ has a constraint with either $x_i$ or $x_j$.*

We give an example of the induced frozen pairs in Figure 3.6. Agents $x_1$ and $x_4$ are not directly frozen with one another, and would not initially detect each other. However, because $x_2$ and $x_3$ are frozen, they will collapse, creating a new frozen pair due to $x_1$ and $x_2$'s shared constraint with $x_5$. Induced frozen pairs happen as a result of a coalition formation between two nodes and are discoverable only after the first full iteration through the VSR algorithm has been completed. As the formation of a coalition combines constraints of two agents into a single surrogate, that surrogate may enable the detection of further pairs of agents that will be frozen given the new topology of the constraint graph. Merging edges and nodes in a repeated fashion results in nested induced frozen pairs and allows for coalition formation to occur between agents in the network that at first seem unrelated to one another.

## 3.5 Empirical Evaluation

In this section we present empirical results pertaining to the structural changes in the constraint graphs as well as the algorithmic performance for solving $k$-coloring problems. All

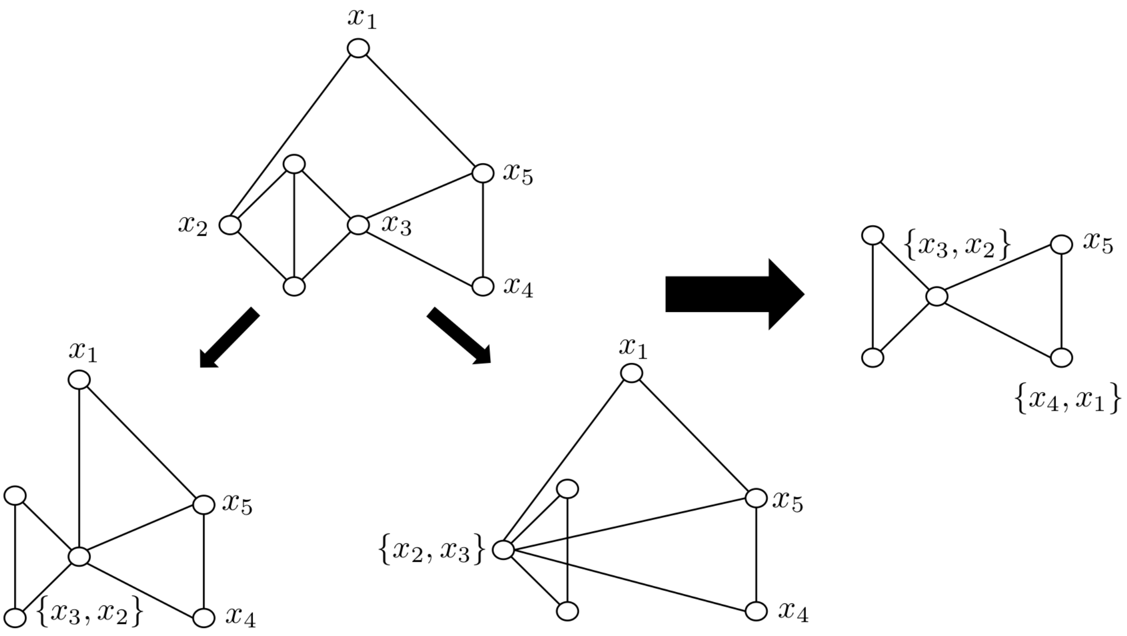


Fig. 3.6: Induction property of frozen pairs. Upon the collapse of $x_2$ and $x_3$, $x_1$ and $x_4$ will discover each other in a frozen pairs relationship allowing them to form a coalition.

of our simulations were run in the MASON multi-agent simulation toolkit [1], developed out of George Mason University. MASON is a discrete-event based multi-agent simulation library designed to allow researchers to custom build large-scale, lightweight simulations. Before we begin discussing the coalition formation results, we will discuss the phase transition in constraint satisfaction problems and how it relates to DCOPs and $k$-coloring. Our discussion on the phase transition is necessary to justify the parameter settings used in the empirical evaluations.

### 3.5.1 Phase Transitions in Constraint Satisfaction

Constraint satisfaction problems (CSP) are one of the fundamental problems in computer science and statistical physics. A typical CSP will involve a large number of variables with a small domain, usually something such as $\{0, 1\}$ or {true, false}. Given a set of constraints relating a set of variables, the question of how many variables can be simultaneously satis-

---

[1] http://cs.gmu.edu/ eclab/projects/mason/

fied has been at the heart of discussion in many works. As we have pointed out, and as has been discussed in the literature, CSPs tend to go through a phase transition of easy-hard-easy sets of problems. The regions of this phase transition have been studied by various researchers since the 1991 seminal work of Cheeseman, Kanefsky, and Taylor [13] and the conclusions as to where the boundaries of easy and hard problems reside has been mostly agreed upon for various types of CSPs. While there is some minor variations as the where the region of hard problems begins and ends, it is most certainly agreed upon that there are more hard problem instances in the middle of the phase transition than at the tails. This region can typically be found via the order parameter of they problem.

For $k$-coloring problems, the order parameter used to define the phase transition is usually the average degree of the graph. Average degree is twice as large as constraint density since average degree $= \frac{2\times|E|}{|V|}$ and constraint density = $\frac{|E|}{|V|}$ (a simple ratio). We unify the theoretical results we report on below to be given in terms of constraint density, where necessary.

Given this, we can control the difficulty of the problems by controlling the constraint density of the generated random instances. Many works provide a thorough investigation in an attempt at locating the bounds of the phase transition for $k$-coloring in terms of average degree, but a strict consensus has yet to arise. Cheeseman et al [13] empirically showed that the probability of finding a solvable problem instance sharply transitioned from 1.0 to 0.0 at approximately an average degree $= 5.0$, which is a constraint density of $= 2.5$. They also showed that there was a correlation at this point with the computational difficulty of finding a solution. Culberson and Gent [15] also performed many empirical evaluations on the phase transition in $k$-coloring problems, finding that the phase transition starts as early as constraint density of 2.3 running to about a max of $3.5$. In 2005, Achlioptas, Naor, and Peres [2] provided a proof that claims the bounds on $d$ could be as wide as $\frac{2(k-1)ln(k-1)}{2} < d < \frac{(2k)ln(k)}{2}$, which for $k = 3$ coloring problems is $\approx 1.4 < d <\approx 3.3$. Their work was strictly on Erdos-Reyni graphs, some with many disconnected components; a property criticized in [40]. We are using random networks, generated similarly as Minton [36] did, which has been adopted in other works such as [38] [35], and [37] where the network is more connected in terms of maximum graph distance. As a result, most researchers investigating random networks recognize a phase transition from $\approx 2.4 < d <\approx 3.5$. Therefore, our test and evaluation will be on graph instances that fall within this range.

As we will show, outside of the phase transition areas, we do not see as much benefit, if any at all, from utilizing the VSR algorithm as we do from using it within the phase transition region. This is due to the under and over constraindness of the problems outside the “hard region" of the phase transition. It has been shown that iterative approximate algorithms, such as DSA, provide very good results in underconstrained problem sets [79]. In the case of overconstrained problems, iterative approximate algorithms are going to perform poorly but, theoretically, there is very little room for them to improve upon the bad results they find anyhow. When problems are underconstrained, there is a lack of sufficient numbers of frozen pairs to allow coalitions to form, leading to wasted cycles and messaging trying to detect such pairs. In overconstrained problem sets, there is as well a lack of frozen pairs to lead to the coalition formation of agents using VSR. Agents detect structural cores but quickly detect that a constraint exists between themselves and the agent opposite the structural core, again leading to wasted cycles.

### 3.5.2 Coalition Formation Results

One of the main goals of performing coalition formation in DCOPs is to reduce the difficulty of finding solutions. In terms of the phase transition, this can be seen as taking a problem instance that lies at some point of density within the transition, and attempting to “push it back” in the transition to an earlier part of the curve. We reintroduce the figure presented in Chapter 2, Figure 3.7, which highlights the area of the transition where we are attempting to reduce our problem instance densities to.

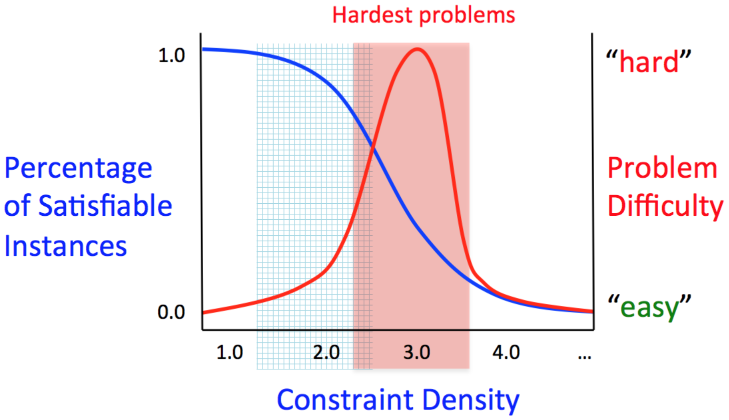


Fig. 3.7: The phase transition for k-coloring DCOPs from easy to hard to easy problem instances based on constraint density. The rightmost shaded region is the original density region where the "hard" problems lie. The leftmost shaded region is the density range we attempt to achieve through the coalition formation process.

Our first evaluation is with respect to the reduction in density of the structure resulting from the coalition formation process. We perform coalition formation only on random instances for $k = 3$ constraint graphs. These graphs were generated via the routine reported in [36] for generating satisfiable instances. We then take an existing edge and replace it with an edge between two randomly selected pairs in the constraint graph. This way of generating random graph instances is similar in fashion to the Model B, Erdos-Renyi function and provides us a way to generate graphs with Erdos-Renyi like properties while guaranteeing full connectedness of the graph. We present the resulting density measures after running VSR on constraint graphs of different sizes in Figure 3.8, and provide more detail for certain densities of interest in Table 3.1. Our analysis shows that using VSR on problems with original densities that are relatively low (i.e. 2.0) we see very little gain. This is due to the lack of core structures that create frozen pairs in the problem. Without a sufficient number of frozen pairs, significant coalition formation can not be found. In Figure 3.8 we see that as density rises more density reduction occurs due to coalition formation. Interestingly, once density rises above a value around 2.7, we see rapid reduction in the resulting density values. This is the result of the occurrence of many structural cores that allow for frozen pairs to form, and corresponds to the phase transition bounds.. In

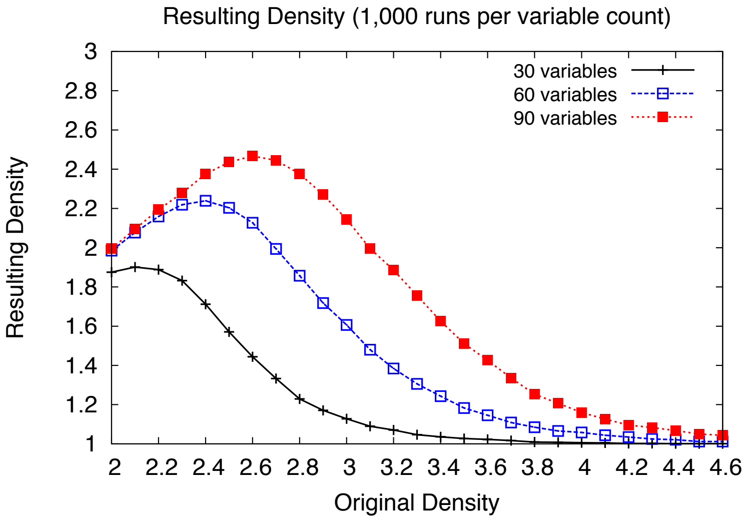


Fig. 3.8: Resulting active density values from virtual structure collapsing.

| **Original Density** | **30 var** | **60 var** | **90 var** |
|---|---|---|---|
| | resulting ± std | resulting ± std | resulting ± std |
| **2.0** | 1.875 ± 0.168 | 1.985 ± 0.037 | 1.996 ± 0.012 |
| **2.3** | 1.832 ± 0.414 | 2.219 ± 0.193 | 2.279 ± 0.059 |
| **2.7** | 1.333 ± 0.453 | 1.994 ± 0.662 | 2.445 ± 0.511 |

Table 3.1: The resulting average **density** for 30, 60, and 90 variable problems after performing a virtual structural collapse.

Table 3.2, we show how the actual active problem size decreases. The active problem size is simply the number of variables that have not collapsed. Note again how the problem size decreases quickly as the coalition formation process continues to occur in higher and higher densities. This should be expected, as the number of frozen pairs is increasing and allowing for more surrogate relationships to form, also supported by the results in Figure 3.8. Also noted is the reduction in overall problem structure resulting from the use of the VSR. As shown in Figure 3.9, we see that problem structure is reducing to approximately 20% of the original problem structure. There has been much discussion in the literature of satisfiablity about backdoor variables [71, 70] and the key variables [60] revolving around what makes the hard problems hard. This could help shed some light on to that problem.

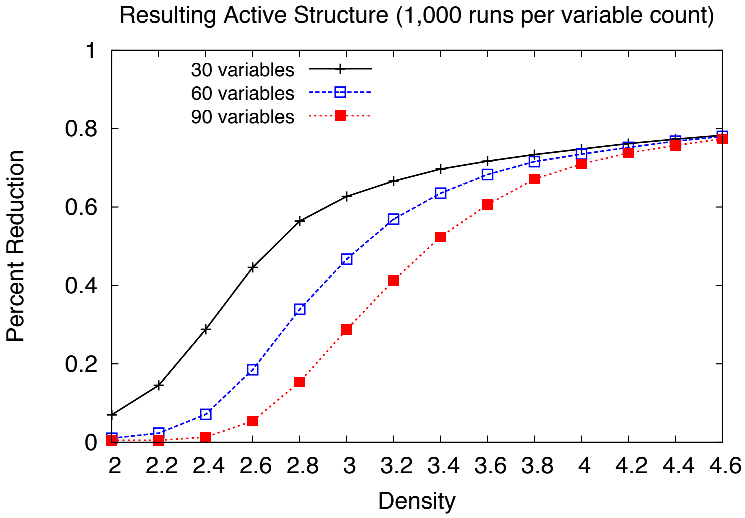


Fig. 3.9: Resulting active structure from virtual structure collapsing.

| **Original** | **30 var** | **60 var** | **90 var** |
|---|---|---|---|
| **Density** | active ± std | active ± std | active ± std |
| **2.0** | 25.73 ± 4.15 | 56.84 ± 2.85 | 87.52 ± 1.96 |
| **2.3** | 18.83 ± 7.13 | 51.43 ± 7.93 | 85.12 ± 5.90 |
| **2.7** | 8.59 ± 5.24 | 30.64 ± 18.13 | 66.05 ± 24.17 |

Table 3.2: The resulting number of **active** variables remaining after VSR is used for 30, 60, and 90 variable problems.

### 3.5.3 Solver Performance

To test performance, we evaluate the VSR-DSA algorithm on a standard benchmark, *MaxSAT 3-colorability* DCOP, or as we have referred to it, the distributed $k$-coloring problem, using randomized graph instances. Given a set of cost functions $f = \{f_1, \dots f_m\}$ where each $f_i(d_i, d_j)$ returns 0 iff $d_i \neq d_j$, 1 otherwise, the goal is to find a set of assignments $A = \{d_1, \dots, d_n\}$ such that global cost $G(A) = \sum_{i=1}^{m} f_i(A)$ is minimized. A $k$-coloring problem is an instance of graph coloring where each variable has the same finite domain of $k$ colors to choose from. Finding an assignment of colors for a $k$-coloring problem, for any fixed integer $k$ greater than 2, is known to be NP-complete [14]. We compare the VSR-DSA algorithm against DSA-B [78], a variation of the Fixed Probability algorithm outlined in [18] and the Maximum Gain Message (MGM) algorithm, a modification of the

Distributed Breakout Method [77], and detailed in [33]. DSA-B has been shown to have the best performance on the $k$-coloring problem of any of the DSA variations that have been developed. DSA-B works by choosing a new assignment with probability $p$ (the activation probability) when the agent can move to another state of equal or better value. DSA-B was chosen as a benchmark to show the relative improvement found by using VSR for coalition formation while the MGM algorithm was chosen as it employs a form of locally consistent behavior through unilateral variable assignment. Both algorithms are covered in more detail in Chapter 2.1.

The four main comparisons we carry out in this section are regarding the coalition formation processes overhead, overall solution quality, message passing, and cycles to completion. We also present a minor result for the real-time performance of the algorithms. In all experiments, we determine that a solution is completed when the solution quality is either maximized (i.e. no constraints violated) or when a maximum number of cycles is exhausted. In all our experiment cases, 500 maximum cycles is used to end testing if no maximal solution is found. We have run tests with maximal cycles of 1000 and 5000 with minimal gains in solution quality.

### *VSR Overhead*

We first present results that report on the amount of overhead associated with using the VSR algorithm in conjunction with the DSA solver. We have developed two different integrations with VSR. VSR-DSA is our full integration attempting to find an optimal solution while collapsing the constraint graph. The VSR-DSA algorithm is an interactive algorithm, meaning that the coalition formation process is being performed inline with the DSA attempting to find a solution. Also presented is the VSR+DSA algorithm, which is a modified version of VSR-DSA which allows for the VSR to be used as a pre-processing routine. The VSR+DSA algorithm is considered a batch pre-processing approach. When using VSR+DSA, coalition formation is run to completion, and then the DSA-B algorithm

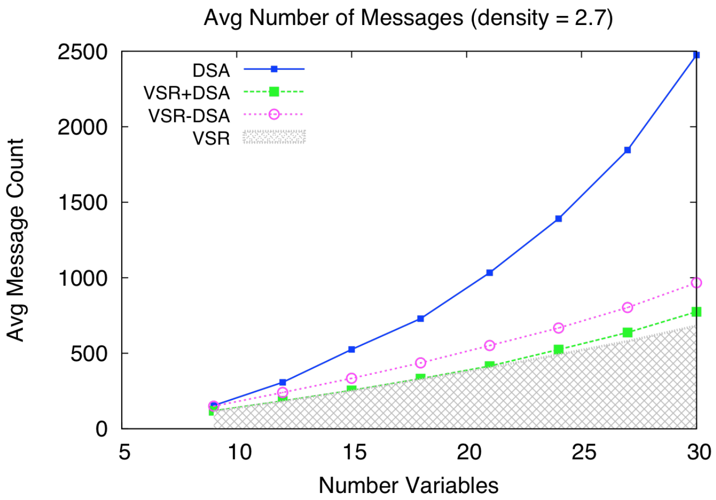


Fig. 3.10: Average number of messages required to solve various instances of $3$-coloring. Here, the shaded area under the curve is displaying the number of messages required for only the coalition formation phase (VSR alone).

is called on the already reduced graphs. As we discussed in Section 3.3, we can abridge the VSR-DSA algorithm to use it as a pure graph processing utility. In order to determine when to stop the coalition formation process, we run VSR until there have been no local structural changes for $\delta$ time steps. The experiments shown reporting on the VSR overhead use a $\delta = 5$.

Choosing to use either the VSR-DSA or VSR+DSA algorithms is a matter of the type of environment the algorithm will be run in. Although we will show that there is a benefit to using VSR-DSA over VSR+DSA, we point out that using VSR as a pre-processing method with solvers other than DSA may be useful. This is especially true when integrating the VSR coalition formation process with a complete optimal solver where problem consistency is required. If we tried to use an inline (or interactive) version of VSR with a complete optimal solver, such as ADOPT [37], the graph topology would change and ruined the assumptions of the algorithm. Again, this is why we are using an iterative approximate solver, as there is no requirement for problem consistency to exist as there are no backups in the search process.

In Figure 3.10, we have a $k$-coloring problem with a density of 2.7. The shaded area under the curve represents the amount of overhead in terms of messages used to find coalitions in the constraint graphs of different variable numbers. The VSR+DSA line itself represents the combined amount of messages used to find a solution once the pre-processed graph has been passed to DSA. As we can see, a significant amount of messaging is used to find the coalition structures that change the graph topology. Very little work is actually required by the DSA solver itself as the graphs reduce down to a small enough number of active variables and a low enough constraint density, as we reported on above. The VSR-DSA full integration requires slightly more messages to find a solution to the problem. This is due to the fact that during the coalition formation process itself, the graph topology is changing and taking instances of problem solutions that are close to a solution and making them non-optimal for the new topology. This requires new negotiation over variable value assignments. Most of this additional message passing due to topology changes happens early in the solution process. If we take a snapshot of the average solution quality for VSR-DSA on a 30 variable problem instance (Figure 3.11) at density of 2.7, we observe solution quality suffering early in search. Topology changes affect already “agreed upon” local solutions. This triggers more negotiation for variable assignment. Once the structures start to settle, the solution quality quickly begins to converge

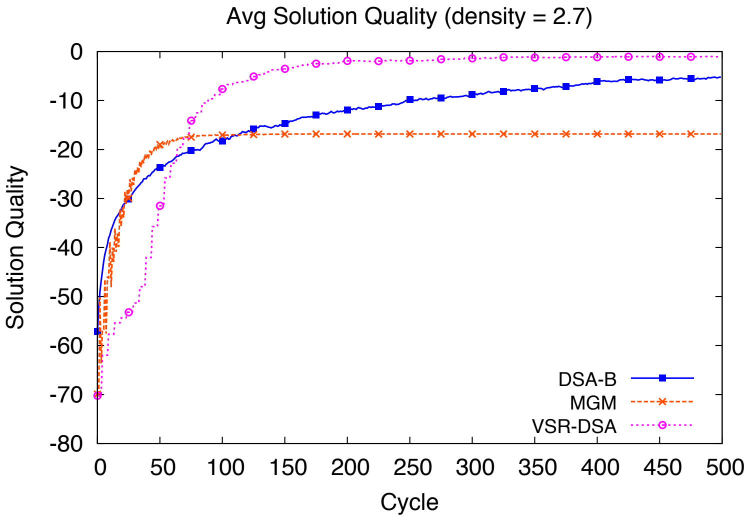


Fig. 3.11: Average solution quality, $n = 40$.

Then the question might be, why don't we just run the VSR algorithm as a pre-processing routine as opposed to the full integration? The answer turns out to be related to the stochastic nature of iterative approximate algorithms themselves. Figure 3.12 shows the number of cycles used on the same problem sets to find an optimal solution. Again, we report the amount of VSR overhead associated with forming the coalitions in terms of number of cycles as the grey shaded area under the VSR+DSA curve. What we find is that the number of cycles used to find a solution using the VSR algorithm as a pre-processing routine and then running DSA requires more work on average than if we used the full integration, VSR-DSA. This is a due to the fact that *sometimes you just get lucky*. More specifically, because the DSA algorithm moves from one solution to another in a stochastic manner, there are opportunities to find a solution before the graph topology settles. In addition to having an opportunity at finding a solution more quickly, in terms of cycles, is that it is difficult to quantify the exact value of $\delta$ for larger numbers of variables. To blame for this is the existence of the Induced Frozen Pairs. When problem instances are small, induced frozen pairs exist in either a very small number or not at all. As the variable size grows, the number of induced frozen pairs grows. Quantifying the number of induced frozen pairs could help us identify the proper value of $\delta$, however the randomization of the problem graph topologies makes this a difficult task. We will discuss this further in Chapter 6.

### *Solution Quality*

Here we will study the effects of density on solution quality, which is the maximum number of constraint violations that have been satisfied. To show the effects that density has on both the problem of finding a solution as well as the opportunity for the VSR algorithm to build coalitions of agents, we investigate instances of the problem before entering the phase transition, within the phase transition region itself, and in the late stages of the phase transition.

Figure 3.13 shows the behavior of the solution quality found using both the DSA, VSR-

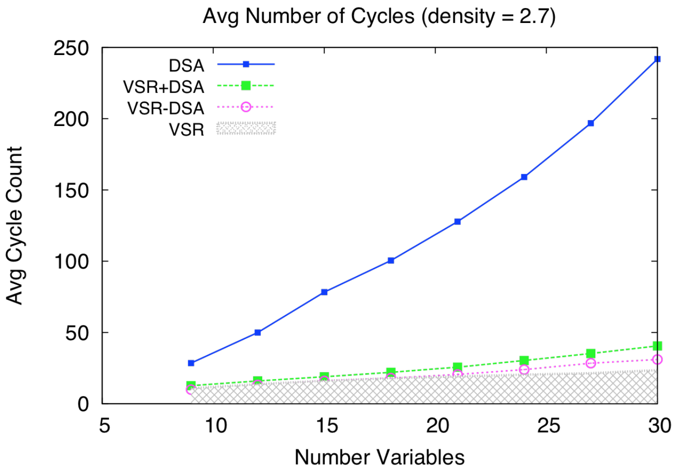


Fig. 3.12: Average number of cycles used to return a result or exceed maximum cycles allotted. Here, the shaded area under the curve is displaying the number of cycles required for only the coalition formation phase (VSR alone).

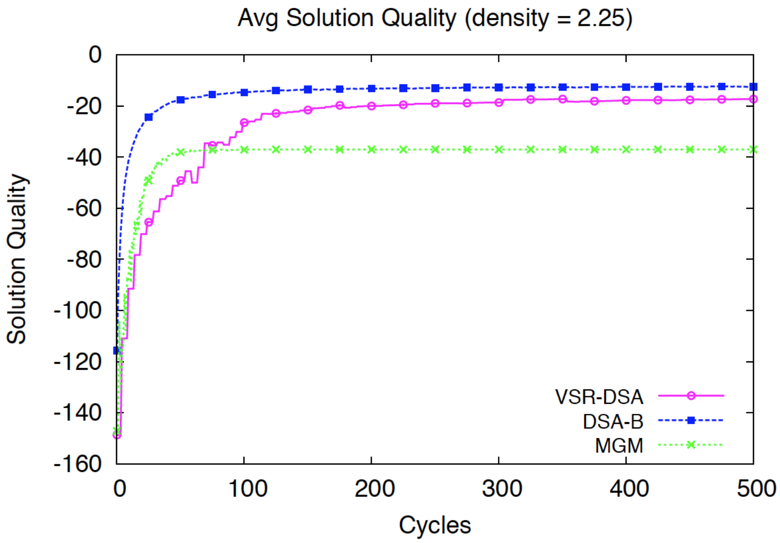


Fig. 3.13: Average solution quality at a density of 2.25, just before entering the phase transition of a $k$-coloring, 100 variable problem.

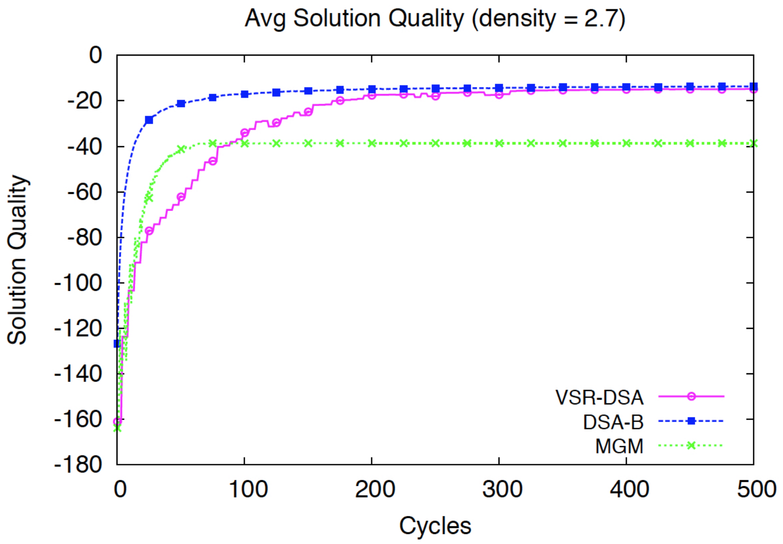


Fig. 3.14: Average solution quality at a density of 2.7, at the beginning of the phase transition of a $k$-coloring, 100 variable problem.

DSA, and MGM algorithms just before entering the phase transition area. As we can see from the results, there is not much benefit to using a coalition structure at this point. As the density of the problem is only 2.25, there is a lack of frozen pair structures to be utilized by the coalition formation process. In addition, most of the graph structures generated with such a low average degree tend to be chains. As the VSR-DSA algorithm is attempting to do some frozen pair discovery where there is a significant lack of them, it pushes the convergence point out farther due to wasted cycles that were not used on search for a solution. Although we do not find benefit in the use of the VSR-DSA vs using DSA alone, we do find that the VSR-DSA converges to similar solution quality. In very sparse graphs, DSA is a better choice for finding a faster convergence on solution quality.

As we enter the beginning of the phase transition at a density of 2.7, we begin to see the VSR-DSA and DSA algorithms converging to approximately the same quality. Frozen pair structures are becoming more prevalent with the higher density allowing VSR-DSA to form coalitions and find solutions equivalent in quality to simply using DSA alone.

At a density of 3.5 and well within the phase transition, we find that the difficulty of

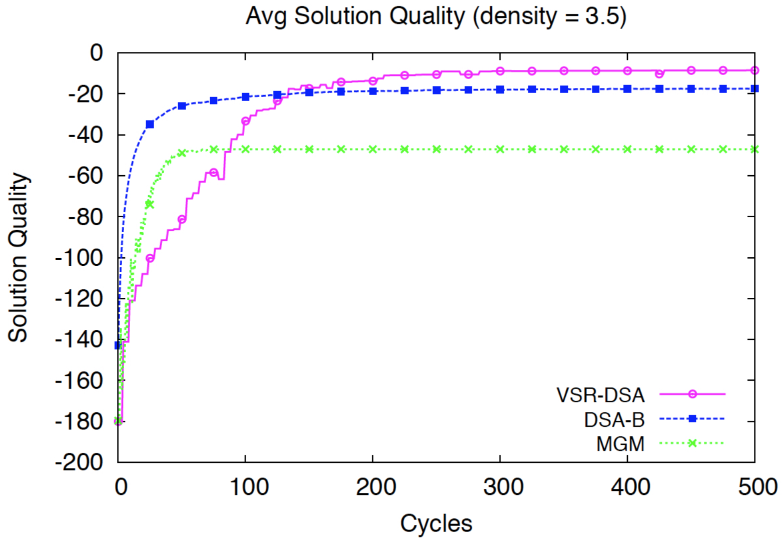


Fig. 3.15: Average solution quality at a density of 3.5, well into the phase transition of a $k$-coloring, 100 variable problem.

the problems located here begin to pose a problem for the stochastic search mechanism, DSA. As the VSR-DSA algorithm forms coalitions and the graph topology reduces, the problem instances themselves begin to alleviate some of the difficult search structures that make cause DSA to get stuck in local maximums.

As is typical with the VSR-DSA, convergence tends to lag behind due to the graph topology changes. Interestingly, it appears that the graph topology begins to settle just as the solution quality begins to overtake the DSA solution quality results. This is most likely due to the fact that the frozen pairs themselves are what is making the problem more difficult, and therefore limiting DSA's ability to breakout of the local maximum it finds itself in. By reducing the structures that harbor the frozen pairs as we do with VSR, we are able to increase the performance of the search and obtain better results.

In Table 3.3 we present the distribution of solution quality for three particular density regions of the problems. The VSR-DSA algorithm finds higher numbers of optimal solutions (no constraint violations) as density increases. When density is at a value of 3.0, 92% of the time the optimal solution is found, however, there is still a small chance that we find a very bad solution (0.7% less than 50% of constraints satisfied). Unfortunately, this is the curse of using iterative approximate algorithms which utilize a stochastic search methodology.

| | | Mean Number of Instances per 100 | |
|---|---|---|---|
| Density | Percent Constraint Violation | VSR-DSA | DSA-B |
| 2.25 | 0% (optimal) | 0.5 | 1.2 |
| | 1% | 4.1 | 6.2 |
| | 5% | 51.3 | 66.9 |
| | 10% | 39.3 | 25.6 |
| | 20% | 4.7 | 0.0 |
| | 50% | 0.1 | 0.1 |
| 2.7 | 0% (optimal) | 52.4 | 4.0 |
| | 1% | 4.9 | 0.3 |
| | 5% | 5.3 | 24.3 |
| | 10% | 27.2 | 67.9 |
| | 20% | 5.8 | 3.4 |
| | 50% | 4.4 | 0.1 |
| 3.0 | 0% (optimal) | 92.0 | 6.1 |
| | 1% | 0.0 | 7.4 |
| | 5% | 2.7 | 15.8 |
| | 10% | 1.5 | 57.5 |
| | 20% | 3.1 | 12.4 |
| | 50% | 0.7 | 0.8 |

Table 3.3: The average number of experiments that resulted in an "at most" percentage of violated constraints after 500 cycles for the VSR-DSA and DSA-B algorithms.

### *Message Passing and Cycle Usage*

Here we discuss the efficiency gains in terms of message passing and cycle usage from the VSR coalition formation process. Given some of the results wide variances reported below, we provide both mean results with their associated variances as well as the Welch's two sample t-test to provide the probability of the samples having equal means[2]. We begin again by investigating solver performance just as we enter the phase transition region. At a density of 2.5, Figure 3.16, the number of messages exchanged between agents using either the VSR-DSA or DSA algorithms begins to vary slightly. The number of frozen pairs begins to appear as approximately $5\%$ of the active nodes are in a frozen relationship. The MGM algorithm offers no benefit in terms of message passing at this density, or in any of our experimental results. Each agent employing the MGM algorithm first sends a

---

[2] Welch two sample tests were run in R (http://www.r-project.org/)

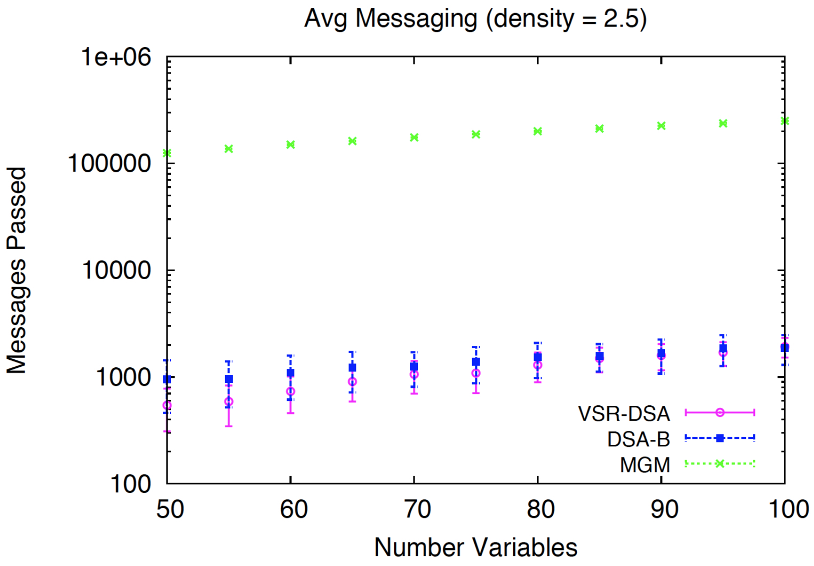


Fig. 3.16: Average number of messages passed at a density of 2.5 as number of variables increase.

coordination message to each of its neighbors so that agents can calculate whether or not they would be the maximal gaining agent in the event of a value change. Then, each agent sends a second message indicating it's new value. The overhead associated with the MGM negotiation and search provides little benefit for the $k$-coloring problem and there for we omit it for the rest of the message passing analysis. Seen in Figure 3.17 (associated Welch t-test in Table 3.4) is a better scaled view of the messaging results for a density of 2.5.

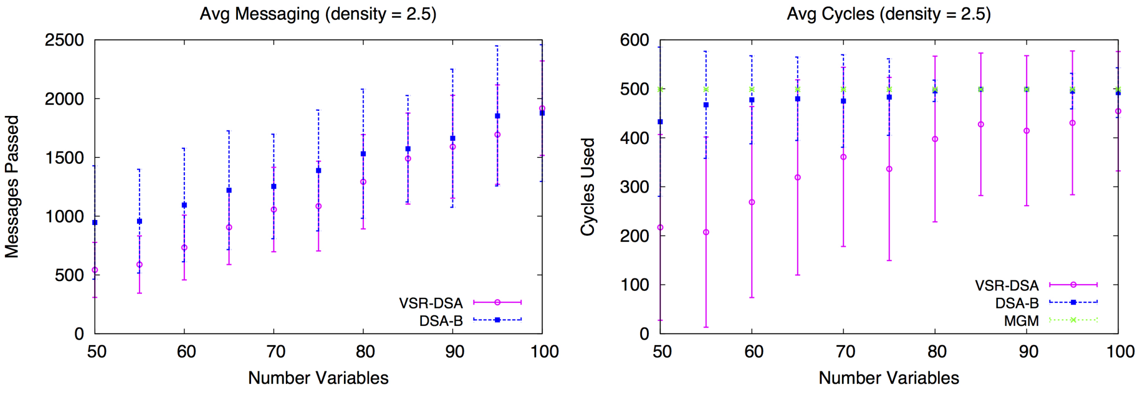


Fig. 3.17: Messages passed at density = 2.5.

Fig. 3.18: Cycles used at density = 2.5.

| | Messages | | | Cycles | | |
|---|---|---|---|---|---|---|
| Number Variables | VSR-DSA Mean | DSA Mean | $p$(DSA $\leq$ VSR-DSA) | VSR-DSA Mean | DSA Mean | $p$(DSA $\leq$ VSR-DSA) |
| 50 | 517.3 | 838.6 | 1.29e$^{-10}$ | 219.7 | 475.3 | 2.20e$^{-16}$ |
| 75 | 1154.4 | 1369.0 | 1.32e$^{-3}$ | 349.1 | 495.7 | 7.17e$^{-12}$ |
| 100 | 1894.5 | 1851.0 | 5.31e$^{-1}$ | 454.5 | 499.0 | 4.34e$^{-4}$ |

Table 3.4: Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 2.5.

As we proceed farther into the phase transition region, at a density of 2.75 in Figure 3.19 (associated Welch t-test in Table 3.5), we begin to see a clear advantage to using frozen pairs to form coalitions. At this density, the percentage of nodes in a frozen pairs relationship is still only about $12\%$, but the average gain in message reduction is upwards of $60\%$ as the size of the problem grows.

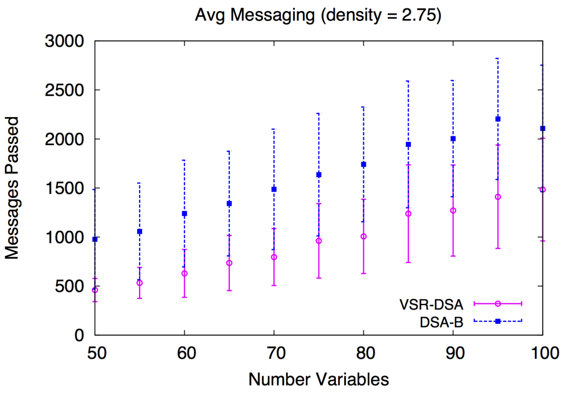


Fig. 3.19: Messages passed at density = 2.75.

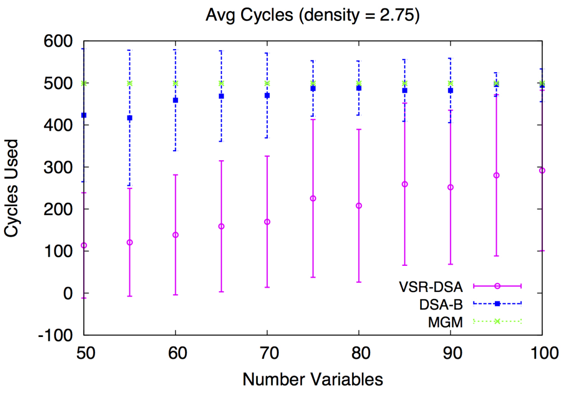


Fig. 3.20: Cycles used at density = 2.75.

| | Messages | | | Cycles | | |
|---|---|---|---|---|---|---|
| Number Variables | VSR-DSA Mean | DSA Mean | $p$(DSA $\leq$ VSR-DSA) | VSR-DSA Mean | DSA Mean | $p$(DSA $\leq$ VSR-DSA) |
| 50 | 435.8 | 952.5 | 2.20e$^{-16}$ | 93.9 | 448.2 | 2.20e$^{-16}$ |
| 75 | 903.6 | 1526.1 | 5.99e$^{-16}$ | 213.5 | 468.1 | 2.20e$^{-16}$ |
| 100 | 2335.7 | 1629.9 | 2.53e$^{-14}$ | 324.5 | 495.0 | 1.12e$^{-13}$ |

Table 3.5: Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 2.75.

Proceeding even farther into the phase transition, at densities of 3.0 and 3.5 (Figures

3.21 and 3.23, respectively and associated Welch t-tests in Tables 3.6 and 3.7, respectively) we find further benefit in terms of the number of messages passed and cycles used. We also observe a tightening in the variance of our results. While the variance around the number of cycles is a product of the random graph generation and certain problem instances being more difficult to solve than others, we can see that the messaging becomes quite efficient as there are more frozen pairs and a higher degree of coalition formation occurring.

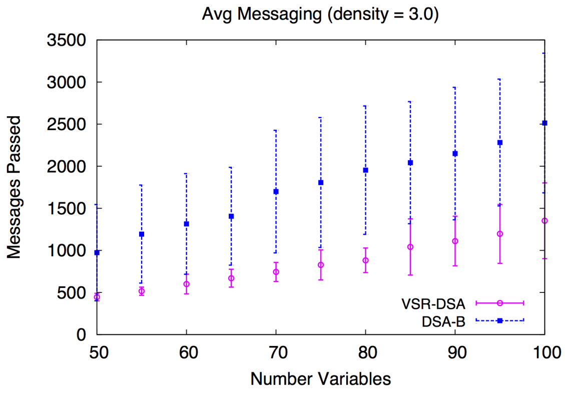


Fig. 3.21: Messages passed at density = 3.0.

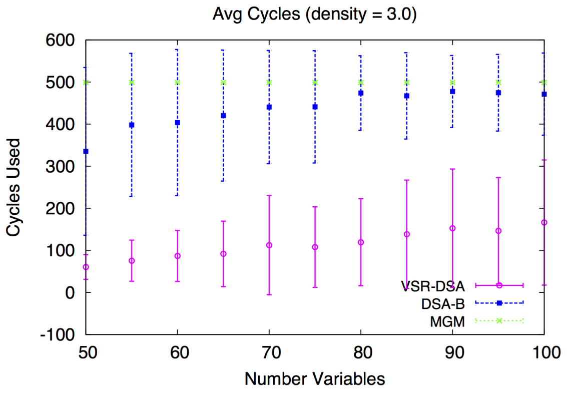


Fig. 3.22: Cycles used at density = 3.0.

| | Messages | | | Cycles | | |
|---|---|---|---|---|---|---|
| Number Variables | VSR-DSA Mean | DSA Mean | $p$(DSA $\leq$ VSR-DSA) | VSR-DSA Mean | DSA Mean | $p$(DSA $\leq$ VSR-DSA) |
| 50 | 446.8 | 1048 | $2.20e^{-16}$ | 71.6 | 341.2 | $2.20e^{-16}$ |
| 75 | 805.1 | 1540.2 | $2.20e^{-16}$ | 100.1 | 435.9 | $2.20e^{-16}$ |
| 100 | 1224.9 | 2474.2 | $2.20e^{-16}$ | 151.2 | 488.3 | $2.20e^{-16}$ |

Table 3.6: Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 3.0.

At these densities, roughly $50\%$ of the graph is inactive (nodes and edges) and we report the percentage of variables that are frozen at these densities to fall between $18\%$ to $25\%$. The percent increase of message passing efficiency is upwards of $45\%$ at $d = 3.0$ and $50\%$ at $d = 3.5$. The percent of increase for cycle useage efficiency is upwards of $67\%$ at $d = 3.0$ and $72\%$ at $d = 3.5$ on average.

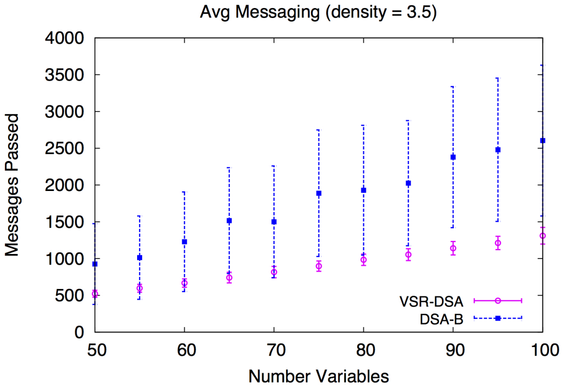


Fig. 3.23: Messages passed at density = 3.5.

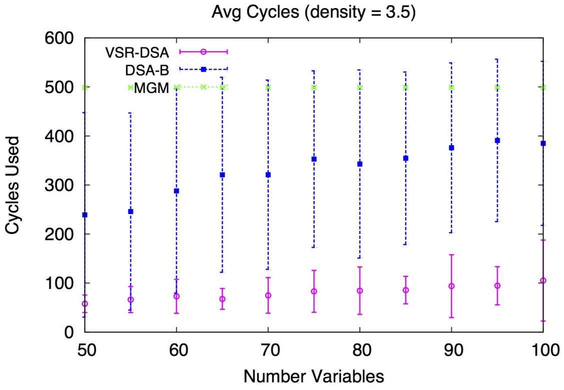


Fig. 3.24: Cycles used at density = 3.5.

| | Messages | | | Cycles | | |
|---|---|---|---|---|---|---|
| Number Variables | VSR-DSA Mean | DSA Mean | $p(\text{DSA} \leq \text{VSR-DSA})$ | VSR-DSA Mean | DSA Mean | $p(\text{DSA} \leq \text{VSR-DSA})$ |
| 50 | 526.8 | 916.9 | $1.81e^{-9}$ | 61.0 | 210.5 | $3.08e^{-11}$ |
| 75 | 879.5 | 1772.3 | $2.20e^{-16}$ | 80.0 | 364.1 | $2.20e^{-16}$ |
| 100 | 1320.7 | 2407.9 | $2.20e^{-16}$ | 101.7 | 389.4 | $2.20e^{-16}$ |

Table 3.7: Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 3.5.

As we begin to come out of the phase transition region, at a density of 4.0 (Figure 3.25 and Table 3.8), the VSR-DSA algorithm is still more efficient in terms of message passing and cycles, but we begin to see the percentage of gain tail off. Variances are slightly tighter in this region due to a high number of frozen pairs, but the average gain against the DSA algorithm falls to only $37\%$ for message passing and about $58\%$ for cycle usage.

| | Messages | | | Cycles | | |
|---|---|---|---|---|---|---|
| Number Variables | VSR-DSA Mean | DSA Mean | $p(\text{DSA} \leq \text{VSR-DSA})$ | VSR-DSA Mean | DSA Mean | $p(\text{DSA} \leq \text{VSR-DSA})$ |
| 50 | 622.3 | 853.3 | $5.50e^{-5}$ | 65.9 | 148.5 | $8.71e^{-6}$ |
| 75 | 1036.6 | 1618.6 | $6.69e^{-10}$ | 76.7 | 232.7 | $2.41e^{-12}$ |
| 100 | 1480.2 | 2320.6 | $1.05e^{-10}$ | 90.8 | 251.6 | $2.11e^{-13}$ |

Table 3.8: Welch two sample t-test for the number of cycles need to solve instances of 3-coloring problems of various sizes at density of 4.0.

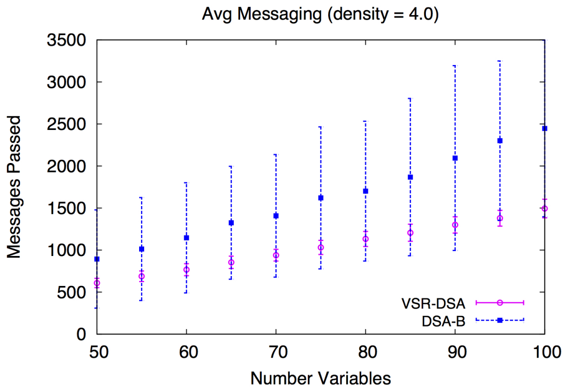


Fig. 3.25: Messages passed at density = 4.0.

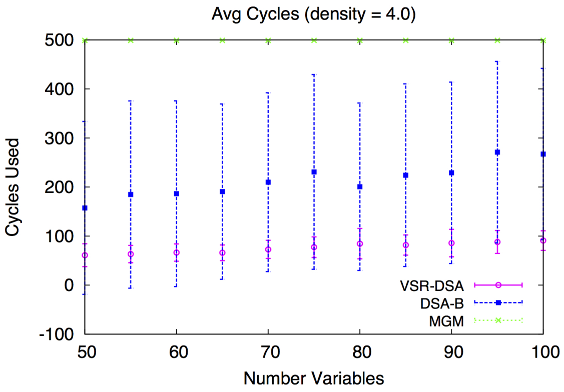


Fig. 3.26: Cycles used at density = 4.0.

### *Real-time Running Results*

In Figure 3.27, we see that VSR-DSA drastically outperforms the basic DSA-B and MGM algorithms in terms of real computation time. This performance gain is due to the reduced number of agents that the system has to wait on for negotiation to occur. Due to agents that have relinquished control of their variable assignments, there is less communication and a hastened turnaround time until the set of agents in the system get to re-negotiate.

We would like to point out that the real-time running results presented here only reflect the average amount of running-time in the system. These results do not accurately reflect how a real distributed system might behave in the elapsed computation time of to find a solution. Because the MASON multi-agent system is being run on a single system, each agent's code is actually called in succession. Reducing the number of active agents in the active set will naturally reduce the overall running time. Less agents being active means less code being run, which leads to this result. However, if we were to field this system onto a real distributed cluster of computing components, removing one computing component from the group will not affect the amount of time that the others use. If this were a true system, what our sample of real-time running result indicates is that the overall system would be using less *average* computation cycles across the entire distributed system. Our cycle use results reported above show that there is a benefit in terms of computational loops that each agent performs, but those computational loops may be a more expensive process

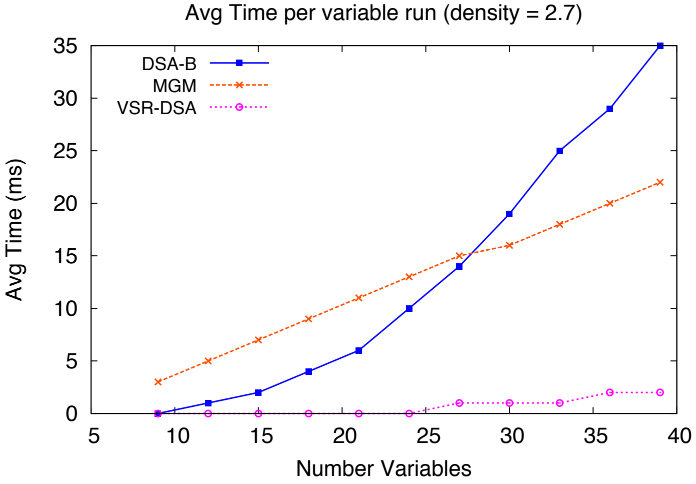


Fig. 3.27: Average number of msec required to solve various instances of k-coloring DCOPs. Performed on a 2.4Ghz Intel Core i7.

due to the coalition search and discovery segments of code. The computing components of the system that are not actively attempting to solve the problem due to their inclusion in a coalition are free to do whatever they choose. It might be the case that they go to sleep or run cycles for another process.

# CHAPTER 4

# PROPENSITY-BASED COALITIONS

In the previous chapter we presented the VSR coalition formation method which is based on the concept of frozen pairs. Agents in a constraint network independently discover relationships between one another such that they must be assigned the same value when a satisfiable solution to a problem is found (*i.e. the joint probability of their variable assignments being equal was always 1.0*). The VSR algorithm is a powerful method for distributedly identifying frozen pairs and forming coalitions within the phase transition of $k$-coloring problems. Upon formation of the proper coalitions, the integrated DSA solver is able to find high quality results with a reduced number of messages passed and cycles utilized.

In this chapter we investigate the question, "*what is the proper measure for coalition formation before the phase transition*"? Although VSR-DSA outperforms DSA and MGM within the phase transition, Figure 3.13 hinted at the fact that the VSR algorithm might actually reduce the performance of DSA in lower density regions. In fact, performance before the phase transition was hindered slightly due to the overhead associated with searching for frozen pairs that might not exist, delaying the convergence to good solution quality. We introduce the concept of *probabilistic frozen pairs* in this chapter to address the VSR-DSA's lack of performance at densities before the phase transition has begun. A probabilistic frozen pair may not necessarily require that the joint probability of two variables values be equal to 1.0 in every solved instance of a problem. In fact, we will discuss many substructure instances in which a coalition can be formed which will allow for positive gain. When necessary, we will refer to the previous chapter's definition of a frozen pair as a *strict (or strictly) frozen pair* to capture the strict assumption of the joint probability of same variable assignments being 1.0. Using probabilistic frozen pairs, we introduce the $\alpha$VSR-DSA algorithm and study its performance on problems before the phase transition where there is a lack of strict frozen pairs.

We introduce the concept of *propensity* which drives the probabilistic frozen pairs relationship and captures the idea of the joint probability between two agents having the same value assignment in a solved instance of a problem. We first introduce a Markov chain formulation of $k$-coloring which motivates a definition of propensity based on the amount of shared neighborhood between two agents. We show that our definition of propensity holds for any 2-hop pairwise agent relationship where the two agents shared a common neighborhood. As the amount of the shared neighborhood grows, the propensity for the two agents to have the same value grows.

We introduce a Markov network definition of propensity which serves two purposes: (1) It verifies our definition of propensity based on shared substructure and (2) It offers us a way to generalize propensity for coalition formation beyond the $k$-coloring problem. In problem domains that do not have only the "not-equals" constraint that we have in $k$-coloring problems, we can use a Markov network formulation to find propensity values for other shared substructures that are composed of different constraint types.

This chapter is arranged as follows. In the next section we introduce the Markov chain representation of $k$-coloring and define a propensity value which drives the probabilistic frozen pair phenomenon. We then generalize the propensity value to work for pairs of agents that share more than one neighbor and are related in a cycle. Following that, we introduce a Markov network formulation of the problem and discuss the use of the Markov network for problems other than $k$-coloring. We then present the $\alpha$VSR-DSA algorithm which uses a minimal value of propensity, $\alpha$, as its basis for coalition formation. Empirical results are presented regarding the solution quality, message passing, and cycle use for problems before the phase transition.

## 4.1 Markov Properties of $k$-Coloring

In this section, we discuss the Markov properties for the $k$-coloring problem and formulate the $k$-coloring problem as a Markov network by which we will derive a model for probabilistic frozen pairs. First, we present a Markov chain formulation for $k$-coloring problems from *an independent agents point of view*. This is an important point worth clarifying. There has been much work in the use of Markov chains for discovering random proper colorings for a graph, where a proper coloring is one in which no two vertices connected by an edge have the same coloring. Most work to date that relates Markov chains and $k$-coloring problems comes from the statistical physics community and focuses on studying the *Glauber dynamics* of an antiferromagnetic Potts model [27]. The Glauber dynamics is modelled as a Markov chain, $X_t$, with a state space of all possible colorings, $\Omega = |k|^{|V|}$, where a random walk from a time step $t$ to $t+1$ proceeds as follows:

1. Choose a vertex $v \in V$ uniform randomly.
2. Choose a color $c \in D$ at random from the set of colors in domain $D$.
3. If the selection of $c$ provides a proper coloring, set $X_t = X_{t+1}$, otherwise $X_t = X_t$.

Given the Glauber dynamics, researchers have focused on finding polynomial time algorithms for generating an approximate set of all the possible proper colorings of a given graph, $G$. Introduced by Jerrum in 1995 [27] to works as recent as Dyer et al [17] and Hayes [25], researchers have provided polynomial time algorithms for estimating the number of proper colorings in $G$ which rely on bounding average degree of the graph with $k$.

What we investigate here is the pairwise relationship between a fixed node in a graph to all other nodes in that graph, termed *propensity*.

**Definition 5.** *Given a constraint graph G, two nodes, $x_i \in G$ and $x_j \in G$ have a* ***propensity*** *value, $\mathcal{P}$, for one another which is the probability that the two nodes are colored the same when a solution to the problem is found.*

A strict frozen pair is a special case of the probabilistic frozen pairs where $\mathcal{P} = 1.0$. A propensity value $\mathcal{P} = 0.0$ indicates that there are no instances of a solution to a problem in which the two variables have the same value. Adjacent pairs in a $k$-coloring problem always have a propensity for one another which is 0.0 due to the fact that they will never have the same coloring. From the definition of propensity, we can now define a probabilistic frozen pair as follows.

**Definition 6.** *Given a constraint graph G, two nodes, $x_i \in G$ and $x_j \in G$ are a* ***probabilistic frozen pair****, $wFP(x_i, x_j)$, iff there exists sufficient shared substructure between $x_i$ and $x_j$ such that $\mathcal{P} \geq \alpha$, where $0 \leq \alpha \leq 1.0$.*

The $\alpha$ value is a minimum threshold we will use to allow agents to determine if they should attempt to form/join a coalition with another agent. What remains to be defined is the actual derivation of propensity. To do so, we will use Markov networks and define a value of propensity based on the inference problem in small Markov networks. We start our discussion by looking at a simple example of propensity in $k$-coloring chained arrangements.

### 4.1.1 $k$-Coloring as a Markov Chain

Given a constraint graph $N = \langle V, E \rangle$, where $V = \{v_1, \ldots, v_n\}$ is a set of $n$ variables and $E = \{e_1, \ldots, e_m\}$ is the set of $m$ binary constraints, we pose the question, "*Given the set of all satisfiable instances in $N$, what is the percentage of those satisfiable instances where two nodes, $i$ and $j$, have the same joint variable assignment?*" To begin, we consider a

DCOP instance of $k = 3$ coloring organized in a chained arrangement. As a combinatorial problem, we know that there $k^n$ total combinations of joint assignments over the entire graph. Since we are interested in only those variable assignments which provide satisfiable solutions to the problem, we reduce the set of final joint assignments to be $3 \times 2 \times 2$. More generally, for a Markov chain of length $n$ in any $k$-coloring problem, we have

$$\binom{k}{1}_1 \times \binom{\text{k}-1}{1}_2 \times \ldots \times \binom{\text{k}-1}{1}_n$$

satisfiable instances for $k \geq 2$ and $n \geq 2$. We see this is true if we consider this as a Markov Decision Process (MDP) and walk sequentially from left to right along the chain, selecting a non-violating color assignments at each node. The first node selects from the full set of possible variable assignments, $k$, while every subsequent node has its choice reduced to choosing from the set of $k - 1$ variable assignments. As an example, Figure 4.1 illustrates all proper colorings for a network with three nodes and three colors.

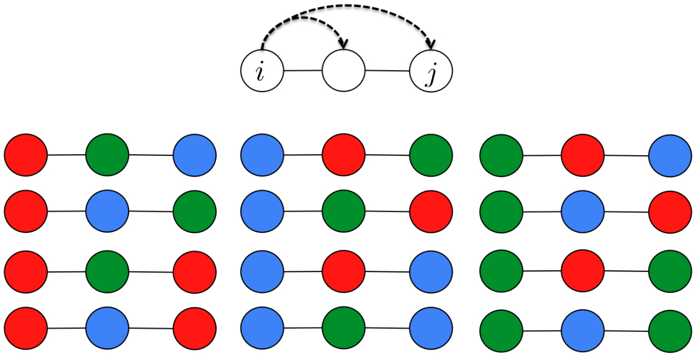


Fig. 4.1: Full enumeration of all possible proper colorings for $k = 3$, $n = 3$.

By brute force we can generate every possible satisfiable instance for the graph and record the frequency of equivalent joint variable assignment.

In Figure 4.2, for this highly degenerate case, we find that variable $i$ has $0\%$ chance of having the same assignment with its adjacent node, and a $50\%$ chance of having the same assignment with the node 2 hops away. If we extend the $k$-coloring problem to a

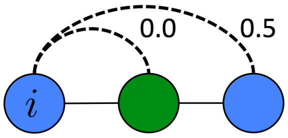


Fig. 4.2: A simple chain example of the 3-coloring problem. Dashed lines indicate a propensity measure.

Markov chain of unbounded length, $n$, we observe a pattern of conditional dependencies that exists between non-adjacent nodes in the graph. We extend the graph to have a length $n \to \infty$ with $k = 3$ and without loss of generality focus on the probability of equivalent joint variable assignments between variable 1's position in the network to all other nodes. In Figure 4.3 we find that the probability of the same value being assigned to any two variables in the network converges in a non-monotonic fashion as the distance between the variables grows. The probability fluctuates around the value $\frac{1}{k}$, or a random assignment.

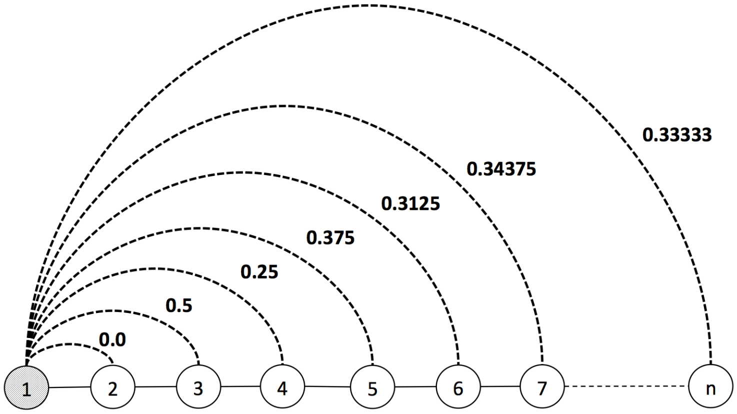


Fig. 4.3: A chain example of the 3-coloring problem with associated probabilities of variables having the same joint assignment. Dashed lines indicate this probability.

Changing the value of $k$ will change the probability measure between variables in a chain arrangement, but we find that the probability will still converge to the value of $\frac{1}{k}$. In Figure 4.4 we find the same non-monotonic behavior for $k = 4$.

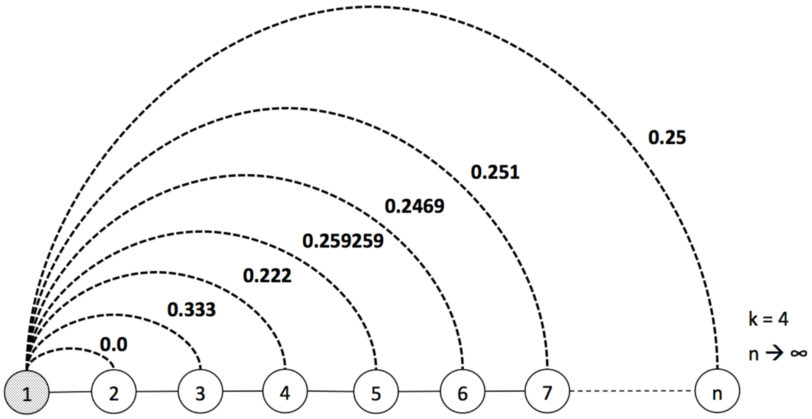


Fig. 4.4: A chain example of the 4-coloring problem with associated probabilities of variables having the same joint assignment. Dashed lines indicate this probability.

Let us define the $n$-step propensity measure, $\mathcal{P}_{i,j}^{(n)}$, to capture the observed probability of equivalent joint variable assignment. For mathematical convenience, we let $\mathcal{P}_{i,i}^{0} = 1.0$, as a variable $i$ has the same assignment as itself every time. We then define $\mathcal{P}$ recursively as

$$\mathcal{P}_{i,j}^{(n)} = \frac{1 - \mathcal{P}_{i,j-1}^{(n-1)}}{k-1}, \tag{4.1}$$

where $i$ and $j$ act both as variable IDs as well as graph indices. We observe that this function behaves in accordance with the observed frequency in the brute force, exhaustive method. The probability of node $i$ equaling node $j$ is strictly dependent on the observed value of node $i$ equaling node $j-1$. Variables that are close to each other in the chain have a higher influence on one another. However, this value quickly converges to random, or $\frac{1}{k}$, with convergence rates increasing as a function of $k$, shown in Figure 4.5. Intuitively, as $k$ grows, there is more freedom in the selection of a variable assignment, and therefore the probability of equivalent joint assignment decreases rapidly.

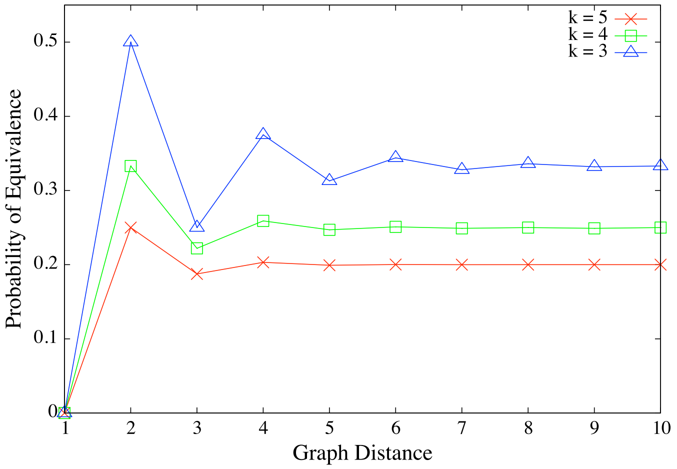


Fig. 4.5: Probabilities of equivalent joint assignments based on graph distance between two nodes in a chain, random = $\frac{1}{k}$.

While Equation 4.1 works for defining Markov chain structures, we need need to expand our definition of propensity to handle more robust structures. We find many cycles in typical constraint graphs and need to calculate the probabilistic relationship of equivalence between variables in these graphs as well. Using the same method as above of generating the sets of all satisfiable instances of a $k = 3$ coloring problem and finding the equivalence probability, we see that the same $n$-step propensity defined in Equation 4.1 does not hold. Highlighted in Figure 4.6 are cyclic constraint graph instances. The 1-step propensity measure is indeed the same due to the not-equals constraint, but the 2-step propensity is different in all cases. This is due to the assignment of a variables value being influenced by multiple variables as opposed to a single variable as in the chain structure. We observe that as the cycle size grows so too does the convergence of the equivalence probability to the related $n$-step measure from the Markov chain instances. For instance, the values of the 2-step propensity measures, $P_{i,j}^{(2)}$, seem to be converging to the value of 0.5 as the size of the cycles grows, again demonstrating a fluctuating behavior. Figure 4.7 shows the observed behavior as cycle size grows. Based on the observation of behavior in limited cycle sizes, it seems that the propensity between an agent and its neighbors on "either side" of it in the cycle follow with Equation 4.1. Unfortunately, this only holds with sufficiently large

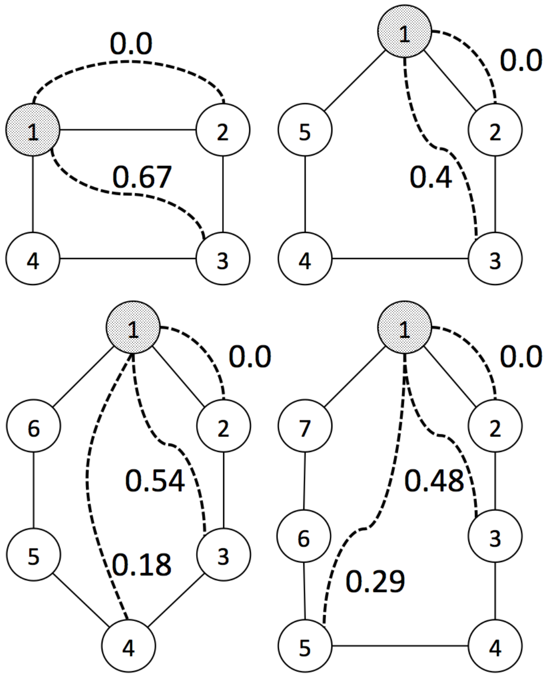


Fig. 4.6: Propensity measures (dashed lines) between agents with growing cycle structure.

$n$, but it still gives us insight as to the relationship of two agents in a cycle; the farther away two agents are in a cycle, they lower the propensity value will be. This idea holds for the Markov chains as well. If we were to chain out two large enough cycles and connect them at their ends, as in Figure 4.3, the propensity values look to converge towards $\frac{1}{k}$ at their connecting point, $\frac{n}{2}$.

The point to take away from the large chains and cycles is that as distance grows in either, the propensity value between two agents drops quickly to random. There is no reason to look out beyond a second neighbor (i.e. a neighbor's neighbor), even in $k = 3$ problems. An agent only needs to share local neighborhood information in order to have a reasonable idea about the propensity relationship between itself and it's second neighbors. Therefore, we concentrate only on the information about shared structure between two agents.

The chain structures shown in Figures 4.3 and 4.4 are straight forward and the propensity values follow from Equation 4.1. We now turn to identify a propensity computation that can handle the case of shared structure of more than one neighbor. Considering at least

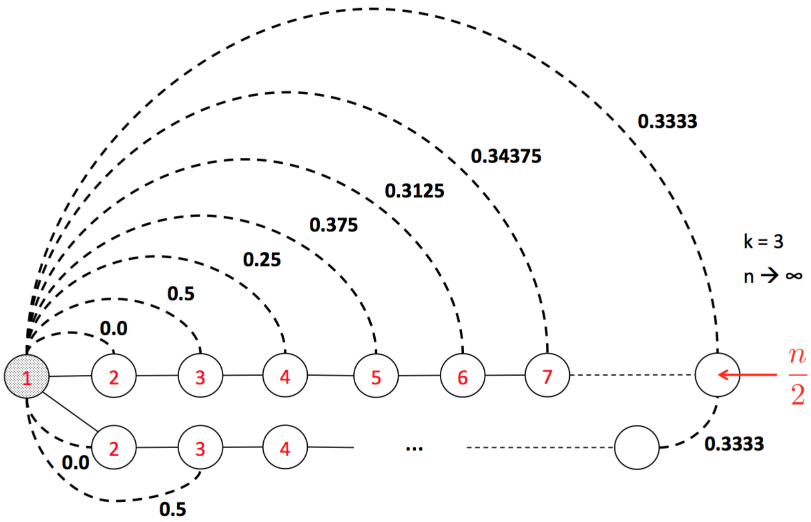


Fig. 4.7: As $n$ grows sufficiently large in a cyclic structure, propensity is observed to converge to that found in the Equation 4.1.

two neighbors will guarantee a cycle in the graph which we have shown Equation 4.1 can not handle. Consider the four node structure in Figure 4.6. From agent 1's perspective, there is no benefit in attempting to form a coalition with either agent 2 or 4. All adjacent nodes have a propensity of 0.0. From the brute force method, we find that the propensity between agent 1 and 3 is 0.67. The majority of the solved instances evaluate with these two agents with the same variable value. As we increase the number of shared neighbors (or structure), the propensity rises quickly, as we show by running the brute force method on the structures in Figure 4.8.

The reason for this increasing propensity is due to the total number of permutations of variable assignments within the shared neighborhood between two agents, $S(i, j)$. Imagine we begin coloring from a centralized approach, starting at the top node, $i$, for any of the structures in 4.8. In a $k = 3$ problem, one color is chosen, at random for $i$'s assignment. As we flow down to the shared neighborhood, there are exactly $(k-1)^b$ permutations, where $b$ is the number of variables in $S(i, j)$, of variable assignments that will allow for a proper (sat) coloring. After that permutation is realized, the variable at the bottom, $j$, of the structure can be assigned whatever value is remaining, based on the shared structure's

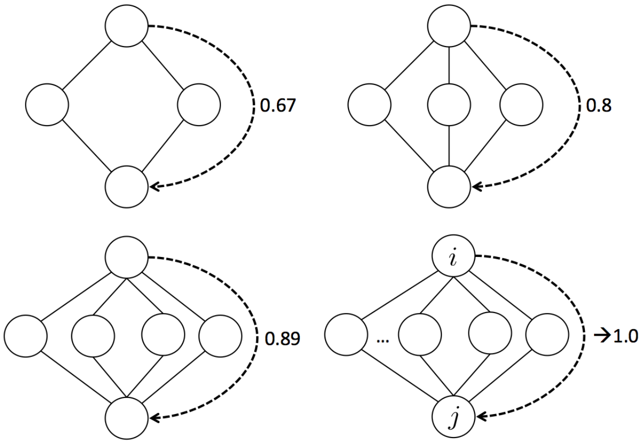


Fig. 4.8: Increasing shared neighborhoods between two variables leads to rapidly increasing propensity values.

assignments. The question then becomes, what permutations of the shared neighborhood's variables will allow $i$ and $j$ to be equal? Let us ask the opposite question: what permutations of the shared neighborhood's variables do not allow the variables $i$ and $j$ to be equal? It turns out, that the only state in which $i$ and $j$ do not have the same assignment is when all the nodes within $S(i, j)$ have been assigned the *same value for their variables*. For example, in any of the structures above, if variable $i$ is assigned "red", then the neighbors of $i$ can only choose from the values "green" and "blue". As long as both "green" and "blue" are selected by any number of agents in the shared neighborhood, then variable $j$ has no choice but to be "red". However, even in the event where the shared neighborhood is assigned all the same value is not enough to imply that $i$ and $j$ will not be the same. There is a chance that the value could be either one of the two remaining values that the shared neighborhood did not utilize.

Using this information, we setup a definition of propensity that is a function of $b$, the number of shared neighbors. Let us define propensity as,

$$\mathcal{P}_{i,j} = \frac{(k-1)^b}{(k-1)^b + (k-1)(k-2)^b}, \tag{4.2}$$

where $k$ is the domain size of the variables and $b$ is the number of shared variables between $i$ and $j$. The numerator of our propensity definition takes into account the total number of permutations that the shared neighborhood can take on. For a cycle of size 4 with $k = 3$, this number is 4 (assuming "R" selected by $i$, the 4 permutations are {G,G}, {B,G}, {B,B}, and {G,B}). When the shared neighborhood takes on one of the $(k-1)^b$ states, it is again possible, but not guaranteed to lead to $i$ and $j$ having the same value. Now we normalize this number by the total number of possible satisfiable instances, where again, $(k-1)^b$ is the set of all permutations where $i$ can equal $j$ and add in the states where $i$ and $j$ could choose differently; when $i$ selects from the $(k-1)$ values not used in $S(i,j)$ and where $j$ selects from the remaining $(k-2)^b$ choices leftover.

For the 4-node, 5-node, and 6-node examples in Figure 4.8, the calculations of propensity using Equation 4.2 are:

$$\mathcal{P}_{i,j} = \frac{4}{6} \approx 0.67$$

$$\mathcal{P}_{i,j} = \frac{8}{10} \approx 0.80$$

$$\mathcal{P}_{i,j} = \frac{16}{18} \approx 0.89$$

respectively, which are exactly the values given by our brute force method. For a chain configuration such as Figure 4.2, the propensity using our new equation is $\mathcal{P} = \frac{2}{4} = 0.5$, which is equivalent to the recursive chain definition of propensity in Equation 4.1. We use Equation 4.2 in the $\alpha$VSR-DSA algorithm to provide propensity values between an agent and his second neighbor based on the shared neighborhood between them.

In Figure 4.9 we show the behavior of Equation 4.2 as the size of the shared neighborhood, $b$, increases. Notice that propensity quickly converges to 1.0 in $k = 3$ problems. When there is no shared structure (at $b = 0$) between two nodes, note that the propensity value is equal to $\frac{1}{k}$, or a random assignment. This is the same as we saw in the Markov chain

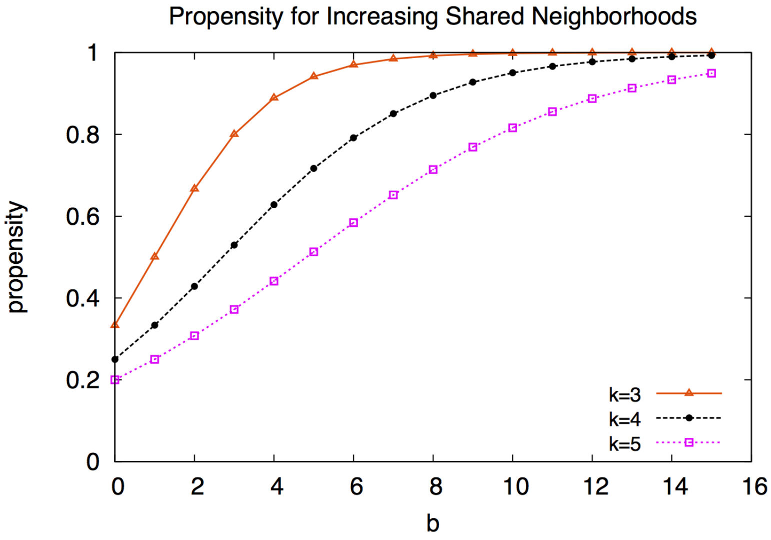


Fig. 4.9: Propensity for increasing sized shared neighborhoods. In the figure, $b$ represents the size of the shared neighborhood between two nodes in the graph.

arrangements. As the size of $k$ increases however, more shared neighbors are needed to increase propensity between two agents. This is due to the fact that there are more possible combinations of variables in the shared neighborhood that could lead to two probabilistically frozen agents to have different values. This is similar to the results we presented in Figure 4.5. With more choice of $k$ comes a weaker relationship between two probabilistically frozen agents.

### 4.1.2 $k$-Coloring as a Markov Network

To both verify and generalize the propensity values between variables in the constraint graphs, we utilize a Markov network formulation of the problem. Markov networks, or Markov Random Fields, originated from the study of statistical physics and probability theory and are essentially an undirected version of Bayesian networks. They were formally studied by Preston in 1974 [53], but were first introduced in Ising's work, in 1925, of which he was attempting to model empirical observations in ferromagenetic materials which resulted in his now famous Ising Model [26]. More recently, Koller and Friedman [31] provide a thorough treatment of Markov networks and demonstrate the usefulness of these graphical models for computer science application. We use the Markov network formulation for two particular reasons: (1) for verification of our propensity definition, Equation 4.2, and the use of ($k$-1) cliques for identifying strict frozen pairs, and (2) to extend the applicability of the frozen pair concept to problems with heterogeneous constraint types.

Let $H = \langle V, E \rangle$ be a (possibly cyclic) undirected graphical model with variables represented by the set of vertices, $V$, and conditional dependencies between the variables represented by the set of edges, $E$. A set of Markov network *factor functions* can be defined for DCOP $k$-coloring problems that represent the local utility, or propensity between any two adjacent variables in the constraint graph. Each edge, or constraint, $e_i \in E$, represents the "value" to each of the agents related by this edge. We define a factor function for a 3-coloring problem as shown in Table 4.1.

| | $\phi(e_i)$ | Value |
|---|---|---|
| $a^{red}$ | $b^{red}$ | 0 |
| $a^{red}$ | $b^{blue}$ | 1 |
| $a^{red}$ | $b^{green}$ | 1 |
| $a^{blue}$ | $b^{red}$ | 1 |
| $a^{blue}$ | $b^{blue}$ | 0 |
| $a^{blue}$ | $b^{green}$ | 1 |
| $a^{green}$ | $b^{red}$ | 1 |
| $a^{green}$ | $b^{blue}$ | 1 |
| $a^{green}$ | $b^{green}$ | 0 |

Table 4.1: A factor function for $k = 3$ coloring.

Given a set of factors that represent every constraint in the $k$-coloring instance, $\Phi = \{\phi_1(e_1), \ldots, \phi_m(e_m)\}$, we can generate a Gibbs measure over the entire space of the DCOP instance. This distribution will provide us with the same measures we empirically observed in the brute force method as well as the propensity values calculated by Equation 4.2. First we generate the unnormalized measure, $\widetilde{P}$, using a product of all the factors representing the constraints.

$$\widetilde{P}_{\Phi}(X_1, \ldots, X_m) = \prod_{i=1}^{k} \phi_i(X_i) \tag{4.3}$$

To normalize Equation 4.3 we define a partition function, $Z_{\Phi}$, which is simply the sum over all the values in the joint distribution table represented by $\widetilde{P}_{\Phi}$.

$$Z_{\Phi} = \sum_{i=1}^{n} \widetilde{P}_{\Phi}(x_i) \tag{4.4}$$

The partition function $Z_{\Phi}$ is then used to give us our desired Gibbs measure for any DCOP instance as:

$$P_{\Phi}(x_1, \ldots, x_n) = \frac{1}{Z_{\Phi}} \widetilde{P}_{\Phi}(x_1, \ldots, x_n) \tag{4.5}$$

Following from the principle of maximum entropy, the Gibbs measure provides for us the complete set of joint variable assignments for our network as well as a way to verify the propensity values from Equation 4.2. For comparison, we ran an exact inference query over the 4,5, and 6-node structures from Figure 4.8 as well as on 7 and 8 node structure of similar topology (i.e. with shared neighborhoods of 5 and 6). The values obtained from the Markov network calculations were exactly equal to our propensity calculations. For the 7 and 8-node Markov networks, the computations came out as:

$$MN(i, j, 5, 3) \approx 0.941 = \mathcal{P}_{i,j} = \frac{2^5}{2^5 + (3-1)(3-2)^5} \approx 0.941$$

$$MN(i, j, 6, 3) \approx 0.97 = \mathcal{P}_{i,j} = \frac{2^6}{2^6 + (3-1)(3-2)^6} \approx 0.97$$

where $MN(i, j, b, k)$ is the Markov network calculation of a $k$-coloring between two nodes, $i$ and $j$, with shared structure size $b$. We also tested our propensity equation against Markov networks with $k = 4$. The Markov network calculations again verify the correctness of the propensity equation, Equation 4.2.

$$MN(i, j, 2, 4) \approx 0.429 = \mathcal{P}_{i,j} = \frac{3^2}{3^2 + (4-1)(4-2)^2} \approx 0.429$$

$$MN(i, j, 3, 4) \approx 0.529 = \mathcal{P}_{i,j} = \frac{3^3}{3^3 + (4-1)(4-2)^3} \approx 0.529$$

### ***Extending Propensity***

The Markov network formulation of a DCOP problem allows us to extend our propensity-based coalition formation approach beyond simple $k$-coloring problems with homogeneous “not-equals” constraints. If we were to change the $k$-coloring problem to include “equals” constraints, the equation for propensity of canonical $k$-coloring problems will no longer hold. Instead of having to reformulate the propensity equation, we can model the factor functions as a representation of the new constraint types that are that makeup a given DCOP. Given a set of constraints relating a set of variable assignments, we can build a factor function that represents those constraints. In Figure 4.2, we show a factor function for a $k$-coloring problem with the addition of an “equals" constraint for the two variables having the same color “blue", which equates to $\phi(a^{blue}, b^{blue}) = 1$.

|  | $\phi(e_i)$ | Value |
|---|---|---|
| $a^{red}$ | $b^{red}$ | 0 |
| $a^{red}$ | $b^{blue}$ | 1 |
| $a^{red}$ | $b^{green}$ | 1 |
| $a^{blue}$ | $b^{red}$ | 1 |
| $a^{blue}$ | $b^{blue}$ | 1 |
| $a^{blue}$ | $b^{green}$ | 1 |
| $a^{green}$ | $b^{red}$ | 1 |
| $a^{green}$ | $b^{blue}$ | 1 |
| $a^{green}$ | $b^{green}$ | 0 |

Table 4.2: A factor function for $k = 3$ coloring with the added "equals" constraint being valued for two variables adjacent and having the value "blue".

Now, instead of attempting to reconfigure our Equation 4.2 to fit the new constraints, we simply a series of graph topologies such as those in Figure 4.8, record the propensities based on the constraints recognized and the number of variables in the shared neighborhood between two variables, and record those special topologies in a look up table for an agent. While this is not as general of an approach as we had for the canonical $k$-coloring problem, it does serve as a meaningful way of encoding graph topologies with their associated propensity values as agents detect them.

### 4.1.3 Strict Frozen Pairs and Propensity

As we have shown, unconstrained agents that have a shared neighborhood between each other share a positive propensity value with one another. Our propensity equation shows that as the size of the shared neighborhood between two agents grows, so too does the propensity between those two agents. We point out that Equation 4.2 does not handle the case where there is a constraint between any two agents *within* the shared neighborhood.

In Chapter 3 we have shown that the strict frozen pair must have the same joint variable assignment, or propensity equal to 1.0, as in Figure 4.10.

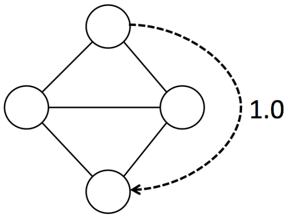


Fig. 4.10: A strict frozen pair for $k$=3 coloring. Dashed line indicates propensity.

If we were to calculate this graph topology as a Markov network, we indeed will obtain a value of 1.0. We point this out as in the next section we present the $\alpha$VSR-DSA algorithm, which uses our propensity equation. As the propensity equation does not account for the strict frozen pair, we must still search for it. If a $(k - 1)$ clique is detected, it is a strict frozen pair, and we set the propensity between the two agents sharing that clique to 1.0.

## 4.2 $\alpha$VSR-DSA

We now present the $\alpha$VSR-DSA algorithm, Algorithm 5, which uses the concepts of both strict and probabilistic frozen pairs to identify potential coalitional partners. We use the tuning parameter, $\alpha$, to control the minimal acceptable level of propensity that the VSR will use to identify frozen partners. In terms of propensity, $\alpha = 1.0$ will only allow the agent to identify the strict frozen pair relationship where it is guaranteed that the two variables must have the same variable joint assignment. As we lower $\alpha$, agents will consider coalition formations with agents with less and less shared structure. For example, at $\alpha = 0.67$, an agent running $\alpha$VSR-DSA will take a coalition partner that it shares a 2-node neighborhood with, as that shown in Figure 4.8.

As with the VSR-DSA algorithm, agents the coalition formation works in a synchronized 3-phase approach with all agents initialized as active. The algorithm initializes by having all agents share their local structure with their neighbors. This information includes the agents neighbors, domain, and constraint information. From this information, each agent can independently calculate the structure they share with each of their second neighbors. This is done in phase 1, frozen pair discovery. During frozen pair discovery, an agent looks for a second neighbor that shares a common structure, $S_{i,j}$. If $\alpha = 1.0$, each agent is searching for a ($k$-1)-clique. A shared ($k$-1)-clique indicates the strict frozen pair, and once detected, the search can terminate. For $\alpha < 1.0$, shared structure is calculated by looking for the number of neighbors that are shared with a particular second neighbor. During the evaluation of propensity from and agent $i$ to an agent $j$, if a strict frozen pair is detected, the search terminates and $\mathcal{P}_{i,j}$ gets set to 1.0. Otherwise, agent $i$ records the number of shared neighbors, $b$, and uses $b$ in Equation 4.2 to calculate $\mathcal{P}_{i,j}$. The propensity values calculated for each second neighbor are stored in a vector of 2-tuples, $F_i$, which holds the second neighbor identifier and propensity value.

As with the VSR-DSA algorithm, each agent independently decides which of them will act as the surrogate of the to-be-formed coalition in the SurrogateSelection algorithm, Algorithm 3. If an agent has determined that it is not a surrogate and will be joining another agent to form a coalition, or join an existing one, the agent sends a collapsing message to all its neighbors. This message indicates that neighbor agents should update their internal data structures to reflect the proper agent to now negotiate with during the search for a viable variable assignments. If the agent is a potential surrogate, it will wait and see if it has been selected to be a surrogate by another agent in the second phase.

In the second phase, agents that are going to join a coalition send the coalition surrogate agent a “collapse” message that includes all that agents local constraint information. The surrogate agent will receive this information and update its internal data structures accordingly. Once the agent has collapsed with the surrogate, it becomes inactive and simply waits for any value assignments that get passed to it from only its surrogate. In the last phase, the Assignment phase, all agents, surrogate or independent, will attempt to negotiate a variable setting using the DSA-S algorithm, Algorithm 4. Local structure changes have been detected by either "collapsing" or "collapse" messages, so updates to the internal data structures would allow proper negotiation with all active agents. Surrogate agents will not send an update message to the members of their coalition until a certain amount of time, $\omega$, has elapsed where no other negotiating agents have sent value update messages. This prevents unnecessary messaging between surrogate and coalition members.

### 4.2.1 Internal Conflict

The propensity values we calculate are estimations of the true value. When two agents are related by a shared neighborhood, they only can take into account that local information when calculating the propensity values. The definition of propensity we provide treats the value calculation as if the network were in a vacuum with no external dependencies. This assumption is fine the majority of the time, however, there is the possibility that a conflict within the coalition itself can occur, which we call *internal conflict*.

Consider the constraint network presented in Figure 4.11. If agent 2 had full knowledge of the problem, she would never chose to form a coalition with both agents 4 and 6 as that would create a coalition in which one solution does not work for all members. Unfortunately, agent 2 only has local knowledge, so the constraint between agents 4 and 6 is unknown, and 4 and 6 do not know they will be simultaneously collapsing to agent 2. Once the collapse occurs, internal conflict is present. This can have a direct effect on solution quality if the agents on the left and right attempt to join the agent in the middle at the same time.

While internal conflict can theoretically occur, in practice they do not occur frequently. In fact, the only way an internal conflict can occur is if there are at least three agents, arranged as in Figure 4.11 (i.e. shared neighborhoods between three consecutive agents), and the agents on the "ends" attempt to form a coalition with the middle agent selected as the surrogate. The constraint densities before the phase transition consist of structures that have "sparse" mean degree, and the occurrence is low where two agents having a

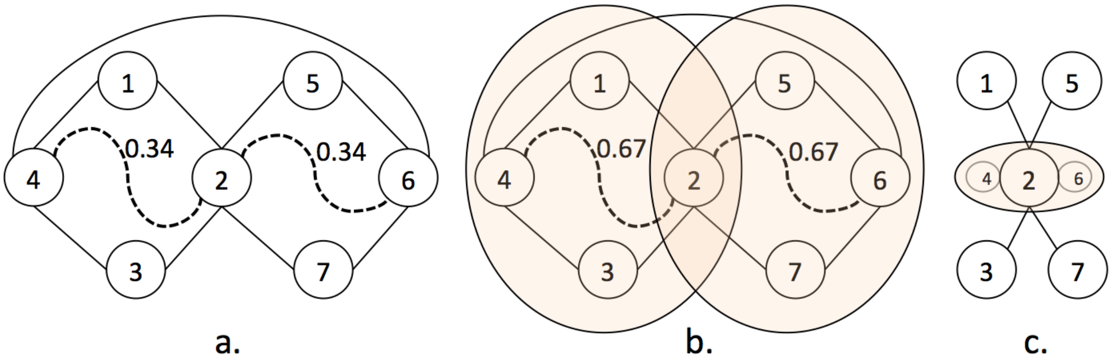


Fig. 4.11: a. True propensity, b. Local propensity estimation, c. Resulting coalition with internal conflict between agents 4 and 6 if this coalition is formed.

probabilistically frozen relationship to the same second neighbor, and then having a direct constraint between each other that would cause internal conflict. We detect internal conflict in our empirical evaluation and find that the occurrence is less than 3%.

## 4.3 Empirical Evaluation

To evaluate the usefulness of coalition formation in the low constraint density regions before the phase transition, we tested our approach in the distributed $k$-Coloring Min-CSP problem discussed in Chapter 2 on 50, 75, and 100 variable problems with a constraint density of 2.25. We chose this density as it represents the area of the problem region which the VSR-DSA algorithm did not at least match the solution quality of the DSA algorithm. The under-performance of VSR-DSA in terms of solution quality at this density is due to the lack of triadic closures which enable a strict frozen pair to form, and therefore result in coalitions. As agents search for coalitions that are not usually there, wasted cycles result in a longer time to convergence than DSA which was constantly looking for a solution. The overhead of the coalition search and formation did not result in “easier” structures, only longer assignment times. We present results using both the DSA algorithm and $\alpha$VSR-DSA algorithm at various settings for minimal propensity. The $\alpha = 1.0$ setting is the VSR-DSA algorithm which will only form a coalition with agents that have strict frozen pair relationship. The $\alpha = 0.67$ setting will have agents forming coalitions with one another if they detect a shared neighborhood that has a minimum of two shared neighbors between any two agents. As the $\alpha$ values go towards 1.0, more and more shared neighbors have to exist for agents to consider them as coalition partners.

We begin by investigating the solution quality performance of the algorithms at a problem size of 50. The DSA algorithm converges at a faster rate than the $\alpha$VSR-DSA algorithm, just as we observed in Chapter 3 due to the coalition formation process eating up cycles that could be used to search for a solution. At this number of variables, all the $\alpha$VSR-DSA implementations converge to roughly the same point as DSA, with the exception of $\alpha = 0.67$. Agents utilizing probabilistic frozen pairs at this density level tend to find coalition partners that offer benefit in terms of finding better solution quality. Once the probabilistic frozen pair coalitions form, at roughly 150 cycles, convergence to a higher average solution quality can be found. The use of probabilistic frozen pairs allows agents to break out of local maximums. For the other $\alpha$ settings, there are not enough occurrences of larger shared neighborhoods that allow for coalitions to form. This results in these implementations getting stuck in the same local maximum as the DSA algorithm.

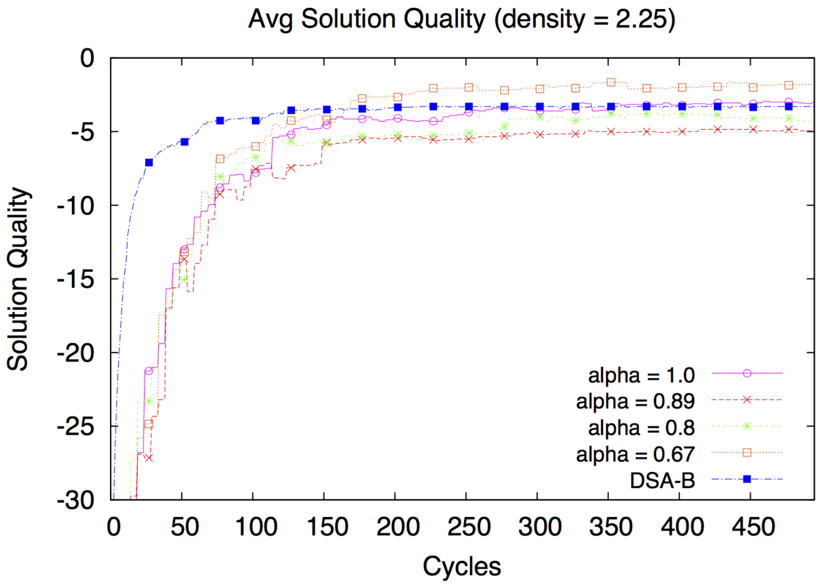


Fig. 4.12: Solution quality for $n = 50$ at a density of 2.25.

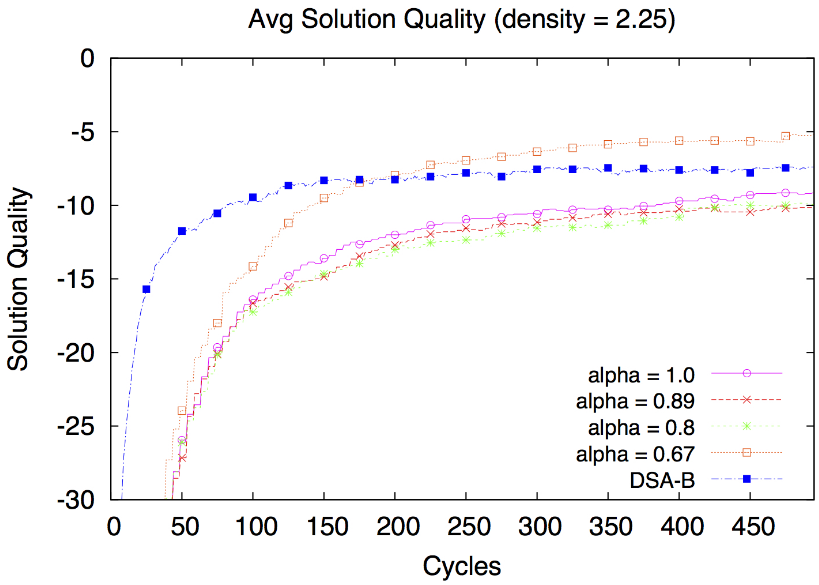


Fig. 4.13: Solution quality for $n = 75$ at a density of 2.25.

At both the 75 and 100 variable problem sizes, we see similar results, where the lack of coalition structures limits the performance boosting effects of VSR method. Again, at $\alpha = 0.67$, there are enough probabilistic frozen pairs to allow coalitions to form, and provide a benefit in terms of overall solution quality.

## 4.4 Summary

In this chapter, we have introduced the concept of a probabilistic frozen pair, which is driven by the propensity value between two agents. Propensity is the probability of the variables having the same joint assignment in satisfiable instances of a given constraint graph. We present a Markov chain formulation of the $k$-coloring problem that motivates the use of only local knowledge as the farther away two agents are in $k$-coloring chain, the more random the propensity between two agents becomes. Propensity values between agents behave differently in a graph cycle than a chain, and we present a definition of propensity to capture the relationship between two variables that exist in a cycle and have a certain amount of shared substructure, or neighborhood. We verify the correctness of

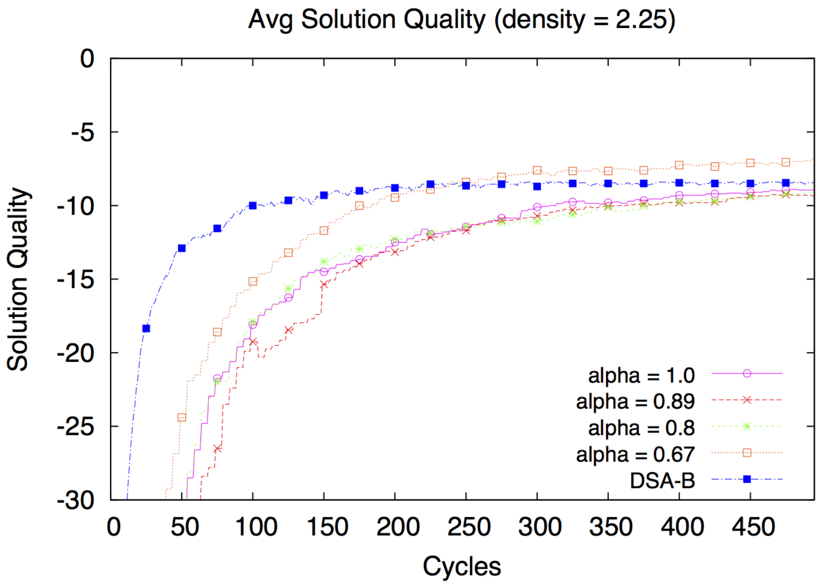


Fig. 4.14: Solution quality for $n = 100$ at a density of 2.25.

our propensity measure by formulating the $k$-coloring problem as a Markov network and showing that structures in both equate to the same propensity values. In addition, we show how the Markov network can allow us to generalize the concept of propensity in problems other than canonical $k$-coloring with only the “not-equals” constraint. This can allow us to set up a Markov network to derive special sub-structures worth exploiting for coalition formation to occur between two agents. We present solution quality results in low density problems where strict frozen pairs do not frequently occur, and the VSR-DSA algorithm struggled. The $\alpha$VSR-DSA uses probabilistic frozen pairs to form coalitions, boosting the performance of the VSR-DSA for increased overall solution quality.

**Algorithm 5** $\alpha$VSR-DSA as executed by agent$_i$

```
{Given: set of neighbors, $\mathcal{N}_i$, tuning parameter $\alpha$}
call ProcessMessages() {Process all new messages}
if active = true then
    if state = Targeted Local Structure Sharing then
        for each neighbor $j \in \mathcal{N}_i$ do
            if local structure update necessary for $j$ then
                send 'local structure' message to $j$
        state ← Frozen Pair Discovery
    if state = Frozen Pair Discovery then
        for each neighbor $j \in \mathcal{N}_i$ do
            Compute $S_{i,j}$
        for each neighbor $j' \in \mathcal{N}_j$ do
            $val_j = mathcalP_{i,j'}$
            if $val_j \geq \alpha$ then
                $F_i \leftarrow (i', val_j)$
        if $|F_i| > 0$ then
            surrogate ← SurrogateDetection($F_i$)
        if surrogate != agent $i$ then
            for each neighbor $j \in \mathcal{N}_i$ do
                send 'collapsing' message to $j$
            canCollapse = true
        else
            canCollapse = false
        state ← Collapse
    if state = Collapse then
        if canCollapse = true then
            send 'collapse' message to surrogate
            active ← false
        state ← Assignment
    if state = Assignment then
        call DSA
        if time since last value update $\geq \omega$ then
            send update message to all coalition members
    else
        do nothing this phase
    state ← Frozen Pair Discovery
```

# CHAPTER 5

# THE STABILITY OF COALITIONS FOR $k$-COLORING

As has been shown in the previous chapters, the use of coalition structures for solving distributed constraint optimization problems with an iterative approximate solver results in both solution quality gains as well as efficiency gains in terms of both the amount of information exchanged as well as the amount of time it takes to find a solution. The use of identifying both strictly, as well as probabilistically, frozen pairs allows agents to distributedly identify potential coalition partners. In this chapter, we detail the cooperative nature of the frozen pairs relationship and study the formal aspects of coalition formation between frozen pairs as that of a *hedonic game*.

We define a new value function for use under the hedonic game assumption. Our value function uses the propensity function to derive value of a potential coalition to an agent. We discuss the stability of the coalitions based on the new value function and introduce a new algorithm for coalition formation under the hedonic game assumption called the Hedonic Game DSA (HG-DSA) algorithm. In the HG-DSA algorithm, agents selectively evaluate potential coalitions and make a decision about joining or not joining based on the composition of agents within (or potentially within) a coalition.

This chapter is organized as follows. We first provide an introduction to hedonic games and their formal definition. Then, we discuss preference and value functions and their properties. We then introduce the Nash stability concept that have been proven in the literature to exist under the use of particular types of value functions. We discuss the stability of the coalitions under the strict and probabilistic frozen pairs relationships. We introduce the HG-DSA algorithm and provide empirical results supporting the value of its use for solving $k$-coloring problems. We point out that sometimes selecting what might seem as the "best" coalition to join does not always result in the most effective strategy for coalition formation.

## 5.1 Hedonic Coalition Formation Game

The study of coalitions in game theory has played an important role since the von Neumann and Morgenstern released their book the *Theory of Games and Economic Behavior* in 1944. Coalitional games, also referred to as cooperative games, address two basic questions: (1) Which coalitions will form and (2) How are the payoffs divided up fairly among the members of the coalitions. Traditional work in the area of cooperative game theory has focused on the second of the two questions, as most of the time it is assumed that either the grand coalition (the group of all members of a game) should and will form, or that the set of players will be pre-arranged in specialized partitions for study. Under these assumptions, analysis concepts such as the Shapley value and core were used to study the distribution of payoffs to members in a partition. To answer the first question, Dreze and Greenberg introduced *coalition formation games* [16] to study the formation of coalitions themselves. Extending the work of Dreze and Greenberg, both Bogomolnaia and Jackson [7] and Banerjee et al [6] introduced the hedonic assumption about the coalition formation process, called *hedonic coalition formation games* or more simply *hedonic games*. The hedonic assumption is that a member of a coalition is only concerned about the composition of the coalition he is a part of and not how external coalitions are organized.

A *hedonic game*, $G$, is a pair, $(N, (\succeq_i)_{i \in N})$, where $N$ is a finite set of all players, $N = \{1, \ldots, n\}$, and $\succeq_i$ is a complete, transitive, and reflexive binary relation (preference profile) over the set $S_i(N) = \{S \in 2^N : i \in S\}$. Note that a strict preference relation, $\succ_i$, and indifference relation, $\sim_i$, are given by $S \succ_i T \Leftrightarrow [S \succeq_i T \textit{ and } T \not\succeq_i S]$ and $S \sim_i T \Leftrightarrow [S \succeq_i T \textit{ and } T \succeq_i S]$). A *coalition partition*[1] is a set $\pi = \{S_k\}_{k=1}^{K}$ over $N$ such that $S_k \subset N$ are disjoint and $\bigcup_{k=1}^{K} S_k = N$. Each subset, $S_k$, is called a coalition of $N$. We denote the coalition which $i$ is a part of in $\pi$ as $S_\pi(i)$ Let the set of all possible coalition partitions in $N$ be denoted as $\pi(N)$. Also, let all partitions of which an agent $i$ is a part of in $\pi(N)$ be denoted as $\pi(i)$.

### 5.1.1 Preferences and Value Functions for Hedonic Games

Each agent $i \in N$ has a value for being in a coalition with another agent $j \in N$ reflected by a real numbered value function $v_i(j)$. We assume that $v_i(i) = 0$ unless otherwise stated. We can extend the value function to apply over coalitions of agents, $v_i(S)$ for all $S \in \pi(i)$. This restricts the evaluation of a coalition to only those coalitions which $i$ can be a part of.

There have been many ways of extending the value function to apply to a coalition for which an agent $i$ finds itself in. In particular, researchers have studied the *min* [59] and *max* [24] hedonic games where a coalition's value to $i$ is based off the least preferred and most preferred member, respectively, *fractional* hedonic games [5] where the coalition's value to $i$ is based off the average value of the members in the coalition, and *additively separable* hedonic games where a coalition's value to $i$ is based off the sum of all members values in the coalition. We will focus our discussion on additively separable games as they are most representative to the coalitions formed under the VSR utility function presented later in the Chapter. First, we present some mathematical properties about value functions.

Given a preference profile for the game $\succeq_i$, and a value function $v_i : N \to \mathcal{R}$, we

---

[1] Also referred to as a coalition structure.

require that,

$$S \succeq_i T \Leftrightarrow v_i(S) \geq v_i(T), \tag{5.1}$$

for all coalitions $S, T \in \pi(i)$. If a coalition is preferred by an agent $i$, then the value function $v_i$ will reflect that preference. We say that a value function exhibits *rationality*, or *individual rationality*, if for all coalitions $S_k \in \pi(i)$,

$$v_i(S_k) \geq v_i(i). \tag{5.2}$$

Individuals in a coalition are behaving rational if the coalition which they are a part of provides as good as or better situation for them than acting alone. Corollary to Equation 5.2, a partition $\pi$ is said to satisfy individual rationality if for all $i \in N$,

$$\pi(i) \succeq_i \{i\} \tag{5.3}$$

We will be discussing the value function an agent uses and how it effects the stability of resulting coalitions. Bogomolnaia and Jackson [7] showed that if a value function demonstrates both additive separability and symmetry that Nash stability. Here we introduce the properties necessary for Nash stable coalitions to form.

**Definition 7.** *A game $G$ satisfies* ***anonymity*** *iff for any $i \in N$, for any $S, T \in S_i(N)$ with $|S| = |T|$ we have $S \sim_i T$.*

Alternatively, we can look at anonymity from the individual player's perspective. We say that player $i$'s value function satisfies *anonymity* if $\forall S_1, S_2$ where $i \in S_1$ and $i \in S_2$, $|S_1| = |S_2| \Rightarrow v_i(S_1) = v_i(S_2)$.

**Definition 8.** *A value function is* ***symmetric*** *if $v_i(j) = v_j(i)$ for all agents $i, j \in N$ and* ***strict*** *symmetric if $v_i(j) \neq 0$.*

As discussed in Chapter 3, the frozen pairs relationship is symmetric by nature. Our value function should not violate that symmetry. Next we introduce the concept of *additive separability*.

**Definition 9.** *A value function is* ***additively separable*** *if for each coalition $S \in \pi(N)$ and all agents $i \in N$,*

$$v_i(S) = \sum_{j \in S} v_i(j)$$

*if $i$ finds himself in the same coalition as $j$ and for any two coalitions $S, T \in \pi(N)$, if $S \succeq_i T$ then,*

$$\sum_{j \in S} v_i(j) \geq \sum_{j \in T} v_i(j)$$

### 5.1.2 Nash Stability

Here we introduce the Nash stability concept which is typically used in the literature. Stability of the system is a key concern for the coalition formation process. Stability, which we can also view as equilibrium, is detrimental to the efficiency of the system and quality of the solutions. If the system were out of equilibrium, we would see tremendous increases in both message and cycle counts as well as degradation in solution quality.

**Definition 10.** *A coalition partition $\pi$ is* ***Nash stable*** *if there does not exist an $i \in N$ and a coalition $S_k \in \pi \cup \{\emptyset\}$ such that $S_k \cup \{i\} >_i S_\pi(i)$ and $S_k \cup \{i\} \geq_j S_k$ for all $j \in S_k$.*

A Nash stable coalition is one in which no player can benefit by defecting from his current coalition $S$ to a new coalition $T$. Nash stability was shown to exist in [7] for additively separable games with symmetry.

### 5.1.3 Frozen Pairs Graph

As we are only concerned with the coalition formation process as it pertains to the set of all frozen pairs, we introduce a *frozen pairs graph* model of the frozen pairs relationships. A frozen pairs graph, $FPG = (V, E)$, represents the frozen agents and their propensities towards one another. We derive $FPG$ from the original constraint graph where $V$ represents our frozen agents (i.e. $FP(i, j) \forall i, j \in N$), and $E = \{(i, j) : i, j \in V, FP(i, j)\}$ represents the propensities between agents. For example, Figure 5.1 shows a simple constraint graph for a $k = 3$ coloring problem with a single frozen pair and the corresponding frozen pairs graph.

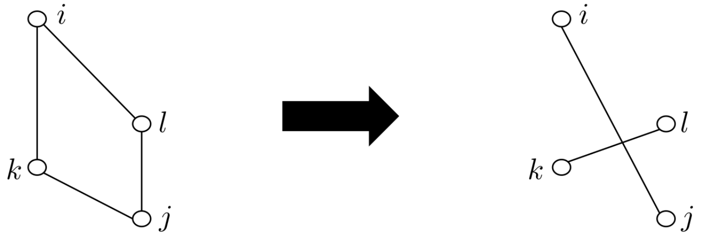


Fig. 5.1: A simple constraint graph (left) with $\alpha = 0.67$. Resulting frozen pairs graph (right) with two edges.

The edges in a frozen pairs graph between two agents, $i$ and $j$, is assumed to have a value of 1.0 if $\mathcal{P}(i, j) \geq \alpha$. In Figure 5.1, the edge between $i$ and $j$ exists only if $\alpha = 0.67$. If we set $\alpha = 1.0$, then we get the resulting frozen pair graph with no edges in Figure 5.2. If unspecified, we assume $\alpha = 1.0$, the strict frozen pairs case.

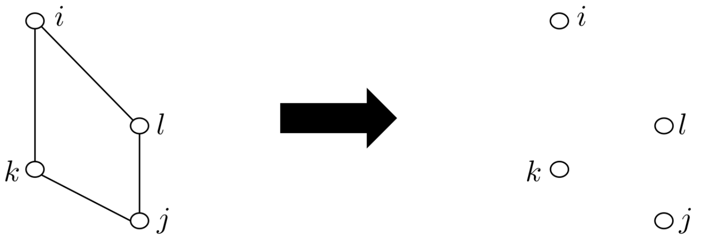


Fig. 5.2: A simple constraint graph (left) with $\alpha = 1.0$. Resulting frozen pairs graph (right) with no edges as there are no strict frozen pairs.

## 5.2 Additively Separable Value Function for Coalition Formation

In this section we cover the two additively separable value functions we use for an agents evaluation of which coalition to join. The general structure of the value function we use is derived directly from Definition 9 and re-stated here:

$$v_i(S) = \sum_{j \in S} v_i(j), \tag{5.4}$$

where,

$$v_i(j) = \begin{cases} 1 & : \mathcal{P}(i,j) \geq \alpha \\ -\infty & : \text{adjacent in constraint graph} \end{cases}$$

The value of $-\infty$ is necessary to maintain consistency in the solutions and make a coalition structure which contains a conflict for an agent $i$ undesirable for that agent. Note that the value of a coalition $S$ is equivalent to the size of the coalition due to the transitivity property of frozen pairs. If two agents $i$ and $j$ are frozen, then any agent $k$ which $i$ or $j$ is frozen with must be frozen with coalition $\{i, j\}$. Given the choice between two equally sized coalitions, the agent would be indifferent. Therefore, given two coalitions of equal size, $U$ and $T$, the value function exhibits anonymity,

$$v_i(U) = \sum_{j \in U} v_i(j) = |U| = |T| = \sum_{k \in T} v_i(k) = v_i(T) \tag{5.5}$$

since the only members of $U$ or $T$ are members where $FP(i,j) = 1.0$. The value function for strict frozen pairs is derived directly from the definition of additive separability, and we show symmetric evaluation below.

**Proposition 3.** *The value function in Equation 5.2 is symmetric between for agents $i$ and $j$, where $v_i(j) = v_j(i)$.*

*Proof.* Part 1: The first part of the proof requires us to show that $v_i(j) = v_j(i)$ for the strict frozen pairs. In the event that an agent detects a $(k-1)$ clique in the frozen pair discovery phase, $\mathcal{P}(i,j)$ will return the value of 1.0 without running a calculation using Equation 4.2. As shown in Chapter 3, both agents in a strict frozen pair will detect each other and must have the same variable assignment.

Part 2: Next, we handle the probabilistic frozen pairs case. Recall that the calculation of $\mathcal{P}(i,j) = \frac{(k-1)^{b_{i,j}}}{(k-1)(k-2)^{b_{i,j}}}$ where $k$ is the size of the domain and $b_{i,j}$ is the size of the shared neighborhood. The calculation of the shared neighborhood, $\mathcal{S}$, between two agents $i$ and $j$ is calculated as $\mathcal{S}(i,j) = \mathcal{N}(i) \cap \mathcal{N}(j)$ (see Chapter 3) where $\mathcal{N}(x)$ is the entire neighborhood of an agent $x$. Then, we have

$$\begin{array}{ccc}
\mathcal{P}(i,j) & = & \mathcal{P}(j,i) \\
\frac{(k-1)^{b_{i,j}}}{(k-1)(k-2)^{b_{i,j}}} & = & \frac{(k-1)^{b_{j,i}}}{(k-1)(k-2)^{b_{j,i}}} \\
\frac{(k-1)^{(\mathcal{S}(i,j))}}{(k-1)(k-2)^{(\mathcal{S}(i,j))}} & = & \frac{(k-1)^{(\mathcal{S}(j,i))}}{(k-1)(k-2)^{(\mathcal{S}(j,i))}} \\
\frac{(k-1)^{|\mathcal{N}(i) \cap \mathcal{N}(j)|}}{(k-1)(k-2)^{|\mathcal{N}(i) \cap \mathcal{N}(j)|}} & = & \frac{(k-1)^{|\mathcal{N}(j) \cap \mathcal{N}(i)|}}{(k-1)(k-2)^{|\mathcal{N}(j) \cap \mathcal{N}(i)|}}
\end{array}$$

for which we know $|\mathcal{N}(j) \cap \mathcal{N}(i)| = |\mathcal{N}(j) \cap \mathcal{N}(i)|$ and therefore $b_{i,j} = b_{j,i}$.

Part 3: Finally, we note that any two agents that are directly constrained with each other (adjacent) both know they are adjacent, and therefore $v_i(j) = v_j(i) = -\infty$ □

### 5.2.1 Strict frozen pairs

When two agents are in a strict frozen pair relationship, they have a propensity of 1.0, a state where there is a guarantee that they will have the same value assignment. Otherwise, there is no such guarantee available and we assign a value of $-\infty$ to prevent any coalitions from forming outside of the strict frozen pairs. Coalitions formed in the strict pairs VSR algorithm are formed in a pairwise, synchronous manner. When an agent determines it is frozen with another, the surrogate selection process indicates to an agent, independently, which one will become the lead. Once an agent joins another agent's coalition, the relationships (constraints) that the joining agent has are transferred to the surrogate. At this point, the surrogate has the information necessary to determine if additional frozen pairs exist from the addition of a coalition member. This is the transitivity property discussed in Chapter 3 Section 3.4 and allows for larger coalitions to be formed over time. As this is the case, as coalition size grows, information about the set of coalition partners to a given individual grows. Consider the game with the frozen pairs shown in Figure 5.3.

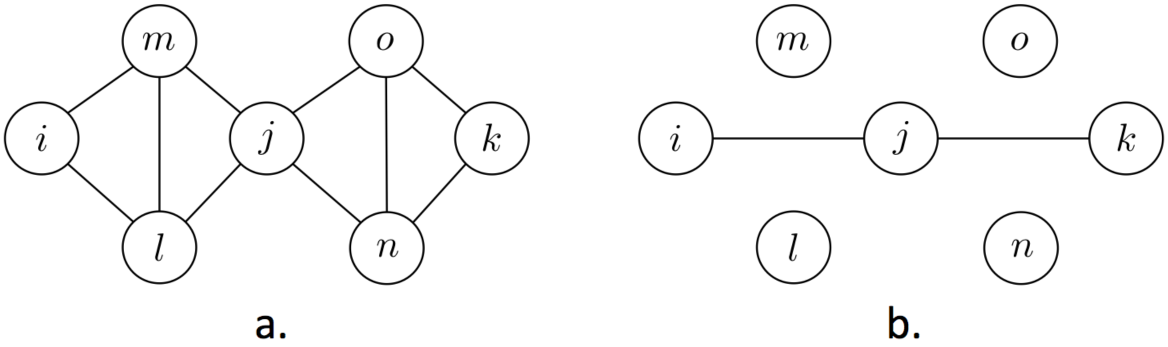


Fig. 5.3: (a) A simple constraint graph with two frozen pairs. (b) A simple frozen pair graph.

We represent the values for this game in Table 5.1 and omit the values for agents with no frozen pair relationships, $\{l, m, n, o\}$.

| $\pi(N)$ | $v_i(S_{\pi(i)})$ | $v_j(S_{\pi(j)})$ | $v_k(S_{\pi(k)})$ |
|---|---|---|---|
| {i}, {j}, {k} | 0 | 0 | 0 |
| {i, j}, {k} | 1 | 1 | 0 |
| {i, k}, {j} | 0 | 0 | 0 |
| {i}, {j, k} | 0 | 1 | 1 |
| {i, j, k} | 1 | 2 | 1 |

Table 5.1: Additive separable values for each agent in the coalition game from Figure 5.3.

The resulting coalition of $\{i, j, k\}$ is stable in terms of Nash stability and core stable. For Nash stability, we are looking for a coalition that one can deviate from their current coalition to better themselves, which does not exist. The partition $\{i, j, k\}$ also represents the unique core allocation for this game.

### 5.2.2 Probabilistic frozen pairs

The coalition formation process for the probabilistic frozen pairs utilizes the same value function given in Equation 5.4. The relaxation away from strict frozen pairs introduces the possibility that an agent $i$ could enter a coalition $S$ which violates $i$'s individual rationality, or $v_i(S_k) < v_i(i)$. This occurs when an agent enters a coalition with internal conflict, as introduced in Chapter 4. We present again the issue of internal conflict in Figure 5.4. As the entry into a coalition can occur simultaneously by an unknown number of agents in the system, we could get internal conflict between two individuals that join at the same time. In the Figure 5.4, agents 2, 4, and 6 would detect each other as probabilistic frozen pairs running the $\alpha$VSR-DSA algorithm with a minimum $\alpha = 0.67$. With the current identifiers of each agent in this configuration and the lexicographical surrogate selection algorithm, agent 2 would be selected as the surrogate for the coalition, and therefore would be the agent that both agents 4 and 6 would attempt to join with. However, notice that agents 4 and 6 have a direct constraint with each other, and therefore a value of $-\infty$ for each other. Unfortunately, neither agent 4 or 6 will detect this issue until they have joined the coalition with agent 2.

The coalition pictured in Figure 5.4c would evaluate to: $v_4(\{4, 2, 6\}) = v_6(\{4, 2, 6\}) = 1 - \infty$ and $v_2(\{4, 2, 6\}) = 2$. Note that this coalition is not a Nash stable solution. Both agents 4 and 5 can deviate from this coalition in order to better themselves. We present the frozen pair graph in Figure 5.5 and full payoff table for agents 2, 4, and 6 in Table 5.2.

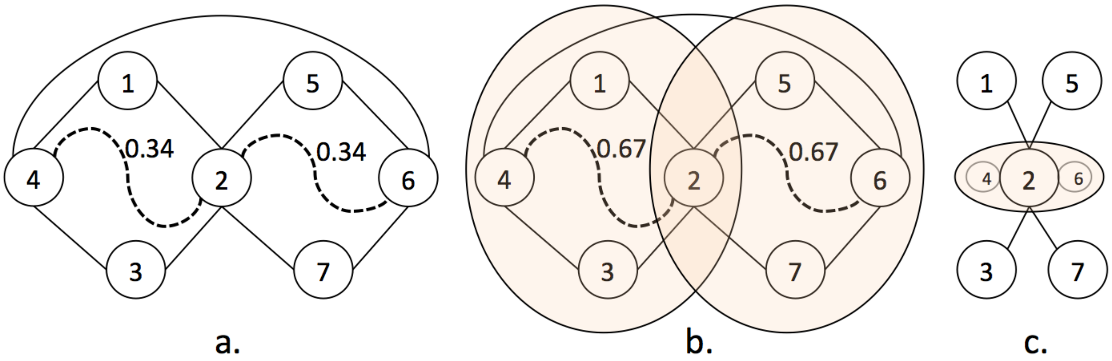


Fig. 5.4: a. True propensity, b. Local propensity estimation, c. Resulting coalition with internal conflict between agents 4 and 6 if this coalition is formed.

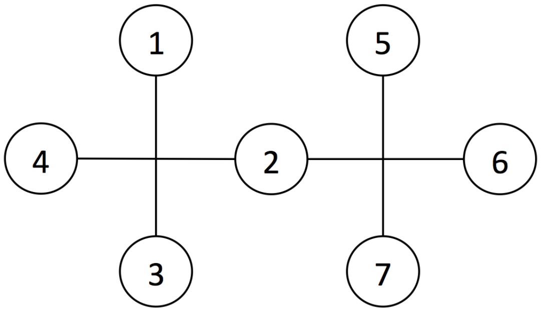


Fig. 5.5: The frozen pairs graph for the constraint graph in Figure 5.4.

| $\pi(N)$ | $v_4(S_{\pi(4)})$ | $v_2(S_{\pi(2)})$ | $v_6(S_{\pi(6)})$ |
|---|---|---|---|
| {4}, {2}, {6} | 0 | 0 | 0 |
| {4, 2}, {6} | 1 | 1 | 0 |
| {4, 6}, {2} | $-\infty$ | 0 | $-\infty$ |
| {4}, {2, 6} | 0 | 1 | 1 |
| {4, 2, 6} | $-\infty$ | 2 | $-\infty$ |

Table 5.2: Additive separable values for each agent in the coalition game from Figure 5.4.

Two Nash stable solutions for the game in Table 5.2; $\{\{4,2\},\{6\}\}$ and $\{\{4\},\{2,6\}\}$. Unfortunately, the $\alpha$VSR-DSA algorithm will not find either of the two in it's current form. Therefore, we must account for this and allow the evaluation of particular coalitions to be done by the agents before forming. For that, we introduce the Hedonic Game DSA algorithm.

## 5.3 Hedonic Game DSA Algorithm

To avoid violating individual rationality, we modify the VSR algorithm to allow agents to evaluate potential coalitions before joining, and therefore avoiding internal conflicts leading to non-Nash stable solutions. We introduce the HG-DSA (Hedonic Game - DSA) algorithm, presented in Algorithm 6. The HG-DSA algorithm overhauls the $\alpha$VSR-DSA algorithm by adding two new states to the coalition formation phase: **Surrogate Response** and **Check Coalition**. The HG-DSA algorithm is structured as such:

1. Targeted Local Structure Sharing
2. Frozen Pair Discovery
3. Surrogate Response
4. Check Coalition
5. Collapse
6. Assignment

Targeted Local Structure Sharing every cycle, but “local-structure” messages are only sent when the local neighborhood of an agent has changed. A “local-structure” message is sent by every agent only in the first cycle and then on a as needed basis. This step is the same as in the original VSR and $\alpha$VSR coalition formation methods.

The Frozen Pair Discovery state is computes the set of all frozen agents. This is done the same as with the $\alpha$VSR-DSA algorithm. The change in the Frozen Pair Discovery comes with a “request” message from any agent that is frozen with another agent but has not determined that it will be acting as a surrogate agent. If the agent has determined it will be acting as a surrogate, it does nothing until the next phase, waiting on the "request" messages from all potential coalition partners.

In the Surrogate Response state, non-surrogate agents do nothing while the surrogate agents collect all "request" messages from the previous cycle. At this point, surrogate agents combine the list of new agents possibly joining the coalition with the set of agents already in the coalition and send a "request-response" message in response to the agents that sent "request" messages. The "request-response" message contains the list of potential and existing members. It is only necessary to send this message to agents requesting to join, as they will be able to determine if they are in conflict with a member already in the coalition without having to ask the current members to additionally check.

In the Check Coalition state, surrogate agents do nothing while the non-surrogates wishing to joining a particular coalition, $\mathcal{C}$ check for any members or potential members which they will have a conflict with. In the case where a conflict is detected, the agent will not attempt to join and will mark a propensity value of $-\infty$ in a history table to prevent future consideration of a coalition with that potential surrogate. If there is no conflict detected, then the algorithm proceeds as it did before, with the agent sending out a "collapsing" message to all neighbors to indicate that future negotiations should be directed to this agent's new coalition surrogate.

All agents proceed to the fourth state, Collapse, agents update their neighborhood information if they received in a "collapsing" message from any of their neighbors. Agents that are joining a coalition send a "collapse" message to their coalition surrogate, and then set their *active* flag to false.

The final state is the Assignment state where all active agents call the DSA method to attempt to find a solution. If no new members have joined the coalition or no new local structure messages have been received in the last $\omega$ cycles, then any surrogate agents send the members of their coalition the newly updated value. By default we set $\omega = 6$ which allows for a complete rotation of states to occur.

### 5.3.1 Empirical Results for HG-DSA

We report mixed results for the performance of the HG-DSA algorithm. In Figure 5.6 and Figure 5.7 the use of the HG-DSA algorithm offers us no advantages in terms of either cycles or message passing. This result is not completely unexpected as we have increased the overhead of the coalition formation process by 2 cycles each iteration. The increased message passing is also a direct result of the extra “request” and “request-response” messages sent by the agents before a collapse.

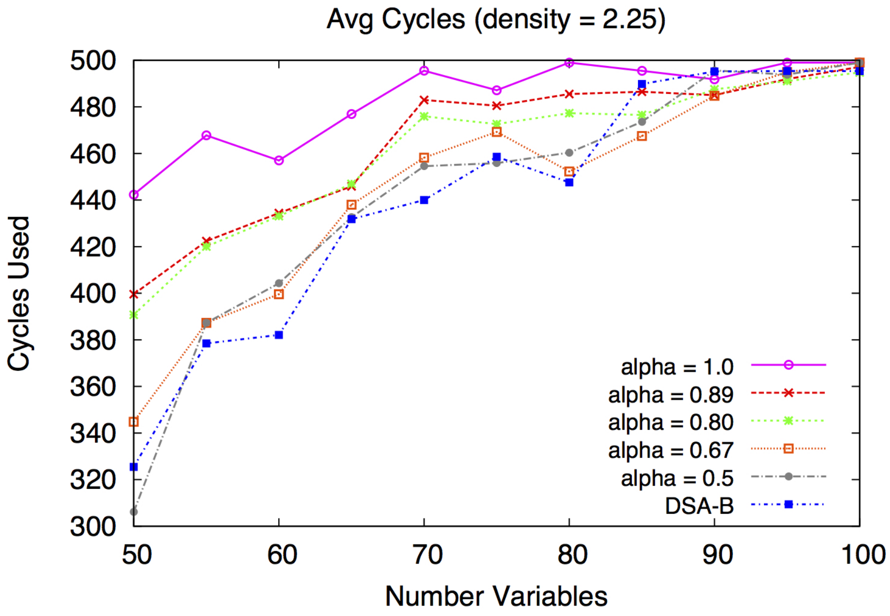


Fig. 5.6: Average number of cycles used to find solutions using HG-DSA.

In terms of the number of cycles used, the HG-DSA algorithm unfairly punishes the strictly frozen pairs case (where $\alpha = 1.0$). As the agents in a strict frozen pairs relationship will always be able to form a coalition, there is no need for the added overhead with requesting and responding. Couple that with the lack of sufficient density to allow enough strict frozen pairs to form, and the HG-DSA with $\alpha = 1.0$ suffers greatly. The same phenomenon also hinders the performance of HG-DSA with $\alpha = 0.89$ and $\alpha = 0.8$.

At $\alpha = 0.67$ and $\alpha = 0.5$, there are enough coalitions that form so that solutions can be found more efficiently, putting performance on par with the DSA-B algorithm.

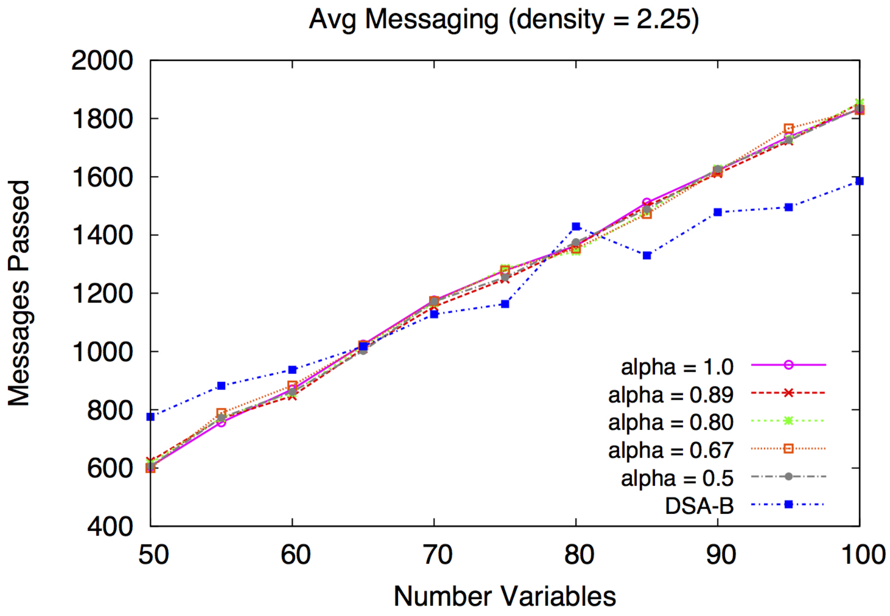


Fig. 5.7: Average number of messages sent within the system using HG-DSA.

For message passing, all the HG-DSA implementations performed approximately the same. In the strict frozen pairs case, there is no need for agents to send request messages if there are no agents they have a strict frozen relationship with. This saved the $\alpha = 1.0$, $\alpha = 0.89$, and $\alpha = 0.8$ from having too many "request" and "request-response" messages being sent throughout the system. For the lower two $\alpha$ settings, more "request" and "request-response" messages were sent, but there was savings since coalition formation was occurring. This evened out the number of overall messages being sent within the system.

A more positive result is the solution quality for regions before the phase transition using our hedonic game coalition formation. In Figures 5.8, 5.9, and 5.10 we provide solution quality results for problem sizes $n = 50$, $n = 75$, and $n = 100$, respectively.

At $n = 50$, we notice that the value of using coalitions that accept members with lower propensity increases the solution quality slightly for $\alpha = 0.67$ and $\alpha = 0.5$. The

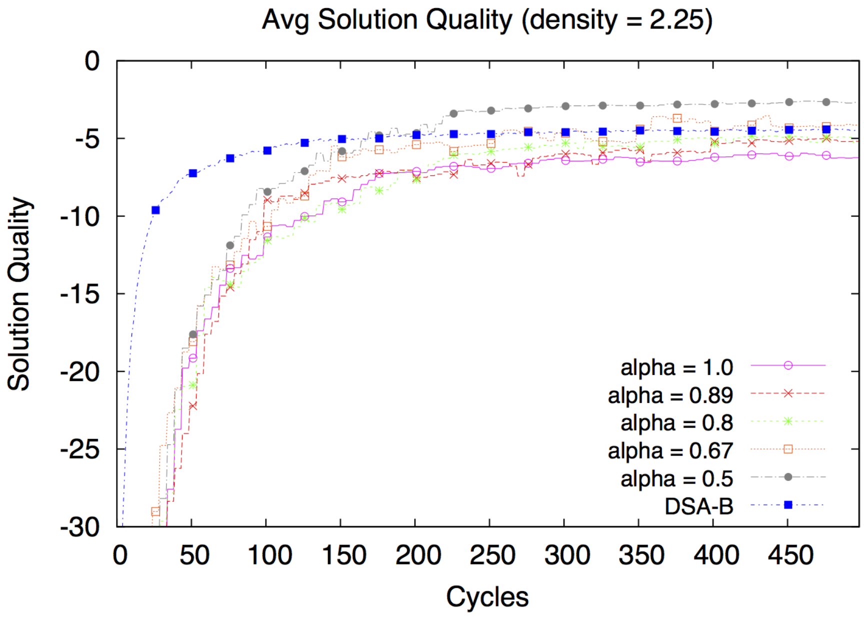


Fig. 5.8: Solution quality for $n = 50$.

solution quality at higher propensities provides insignificant gain. While the formation of these coalitions should increase the solution quality due to the fact that there can be no internal conflict using the HG-DSA algorithm, the additional overhead for searching for those coalitions with no actual formation results in much slower convergence to the same level as DSA-B.

We notice substantial improvement in solution quality as problem size becomes larger. In the larger instances with lower density, there are more chains and loosely coupled agents. This allows the lowest propensity relationships ($\alpha = 0.5$) to form effective coalitions and work their way out of local maximum. We find that the $\alpha = 0.67$ formation also results in increased solution quality, but the structures seem to not occur often enough at such low densities.

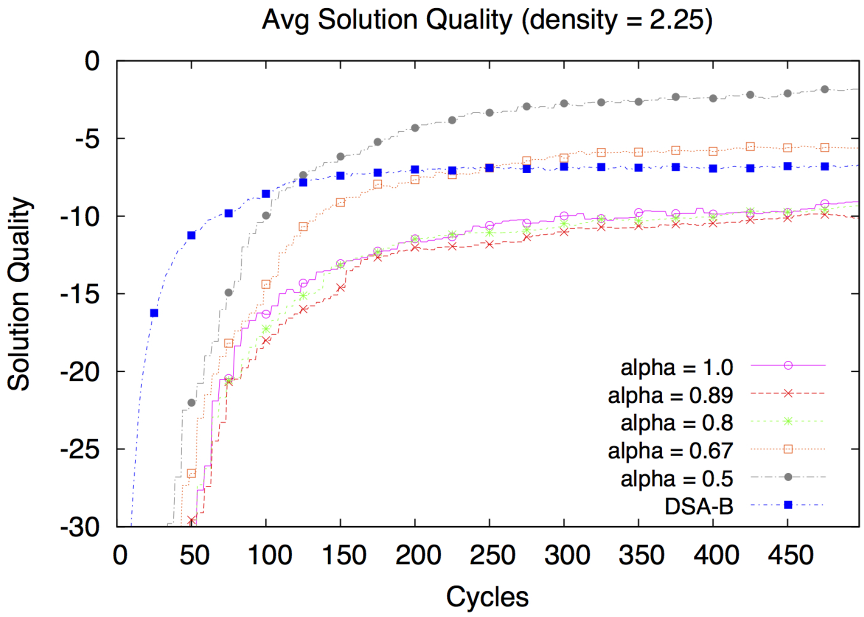


Fig. 5.9: Solution quality for $n = 75$.

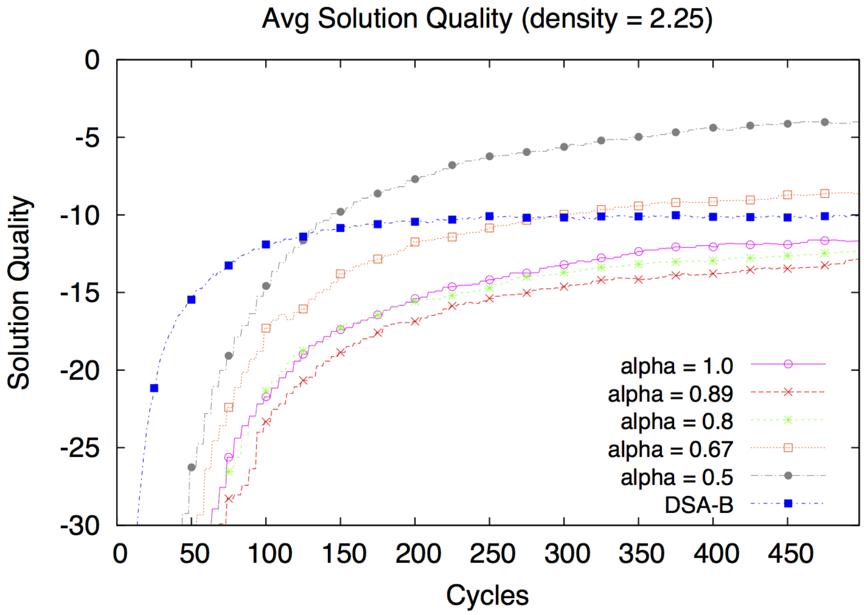


Fig. 5.10: Solution quality for $n = 100$.

As problem size continues to grow, we see sustained gains in solution quality using low propensity coalitions against DSA-B, and little value added using higher propensity coalitions due to lack of shared structure.

## 5.4 Effects of Coalition Partners with Low Propensity

Up until this point, it has been assumed that during the search for coalitional partners, an agent will search for other agents where the propensity relationship, $\mathcal{P}$, evaluates to a value greater than the minimum threshold, $\alpha$. Given a choice between multiple agents of varying propensity, the agent will select the one with the highest value. Intuitively this makes sense. As propensity is a measure of the likelihood that two agents will end up with the same value once the instance is solved, joining a coalition in which this value is the highest would seem to lead to the greatest chance of success. Indeed, for the strict frozen pair, an agent would be acting irrational to not choose a coalition partner where they must have the same value. But can the same be said for the probabilistic frozen pair relationship? In Figure 5.11 we re-present the internal conflict graph along with the potential coalitions that could form based on propensity at certain $\alpha$ values.

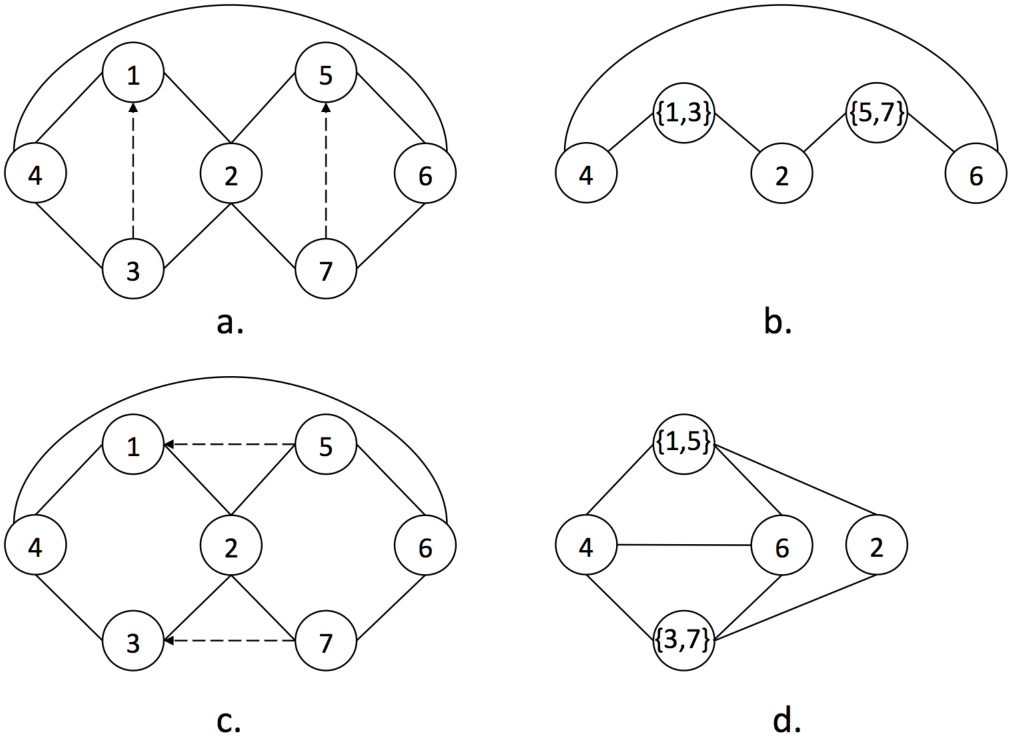


Fig. 5.11: (a) Original constraint graph with $\alpha = 0.67$. (b) Resulting coalition from using $\alpha = 0.67$. (c) Original constraint graph with $\alpha = 0.5$. (d) Resulting coalitions from using $\alpha = 0.5$.

In Figure 5.11b, we see that if the agents were using the value of $\alpha = 0.67$, the coalitions $\{1, 3\}$ and $\{5, 7\}$ would form, leading to a resulting coalition which has five coalitions in a single cycle. As covered previously, the coalitions $\{4, 2\}$ and $\{2, 6\}$ are not possible based on the HG-DSA algorithms method for avoiding internal conflict. Once the coalitions in 5.11b are formed, the coalition formation process will stop because there are no probabilistic frozen pair relationships that evaluate to a minimum of $\alpha = 0.67$. However, if we were to utilize a lower value of acceptable propensity at $\alpha = 0.5$ as shown in Figure 5.11c, we would get the coalition structures shown in Figure 5.11d where a strict frozen pair has formed. The coalition formation process would not end as agent 2 could join the coalition with agents 4 or 6 and the strict frozen pair would collapse to form the coalition $\{1, 5, 3, 7\}$. In fact, from the original constraint graph, we can derive a perfect $k$-clique, where every variable is satisfied by the assignment which surrogate agent negotiates with its neighbors.

To test the effects of coalition formation with low propensity partners, we look at the solution quality resulting in agents selecting to join coalitions with which they have high, low, and exact propensities with. In the presence of a strict frozen pair, agents will always choose to form those coalitions, but outside of the strict frozen pair relationship, we find results vary depending on the preference.

In Figure 5.12, we report results pertaining to coalition selection with agents either preferring the highest propensity relationships found (High), lowest propensity relationships found (Low), or the exact propensity of $0.5$ (Exact). The agents with preference towards selecting the highest found propensity coalitions are those we have studied up until this point. We find that when we enable agents to form coalitions by preferring the lowest propensity relationships, we can still find an uptick in solution quality over the previous preference of highest possible propensity. Additionally, looking for exactly the propensity value of $0.5$ leads to approximately equal results as the high propensity preference. This is a positive result for low density problems, but further investigation is necessary to see if preferring

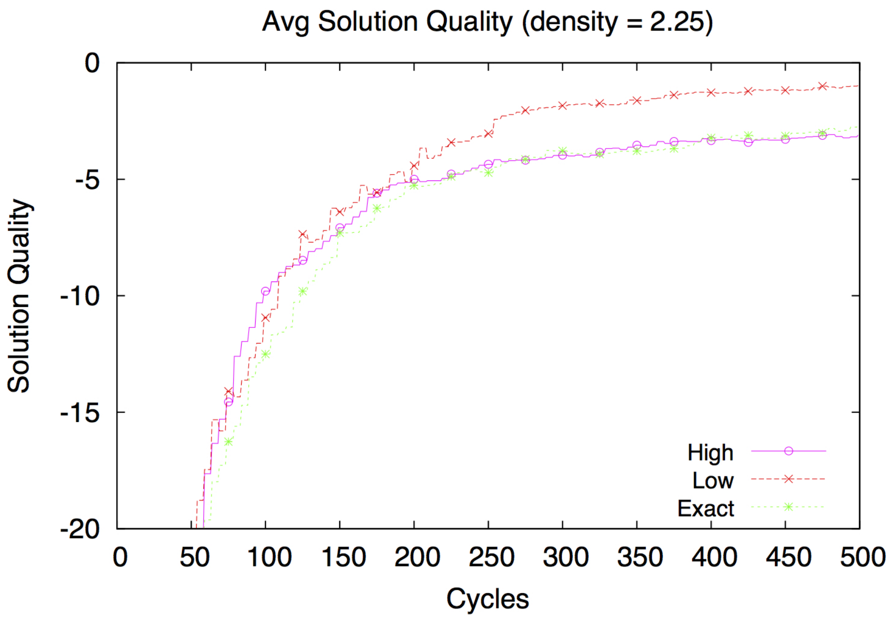


Fig. 5.12: HG-DSA at $\alpha = 0.5$ with $n = 75$ using a surrogate selection that looks for highest propensity first (High), lowest propensity first (Low), and exact propensity first (Exact).

lower propensity coalitions would offer any benefit in terms of better solution quality for problems with a density that falls within the phase transition.

## 5.5 Summary

The hedonic game based algorithm, HG-DSA, allows agents to only join coalitions for which they have conflict free relationships. The coalitions that form are stable in terms of Nash stability due to the properties of additive separability and symmetry, but finding those coalitions comes at a cost of increased cycles and messaging. Using the HG-DSA algorithm in problem instances where constraint density is low is an attractive option when using lower propensity thresholds for coalition formation to occur. The lower propensity coalitions lead to greater structure reduction as a result of interesting series of coalition formations such as those shown in Figure 5.11.

**Algorithm 6** HG-DSA as executed by agent$_i$

```
{INPUT: set of neighbors, $\mathcal{N}_i$, tuning parameter $\alpha$}
call ProcessMessages() {Process all new messages}
if active = true then
    if state = Targeted Local Structure Sharing then
        for each neighbor $j \in \mathcal{N}_i$ do
            if local structure update necessary for $j$ then
                send 'local structure' message to $j$
        state ← Frozen Pair Discovery
    if state = Frozen Pair Discovery then
        surrogate ← frozenPairDiscovery($\mathcal{N}_i$, $\alpha$)
        if surrogate != agent $i$ then
            {this agent is not a surrogate agent}
            send 'pre-coalition' message to surrogate
        state ← Check Coalition
    if state = Check Coalition then
        if received 'coalition' message then
            Check Coalition $\mathcal{C}$ for conflicts
            if $\mathcal{C}$ is conflict free then
                for each neighbor $j \in \mathcal{N}_i$ do
                    send 'collapsing' message to $j$
                canCollapse = true
            else
                historyTable[surrogate] = $-\infty$
                canCollapse = false
            if surrogate = = agent $i$ then
                canCollapse = false
            state ← Collapse
    if state = Collapse then
        if canCollapse = true then
            send 'collapse' message to surrogate
            active ← false
        state ← Assignment
    if state = Assignment then
        call DSA
        if time since last value update $\geq \omega$ then
            send update message to all coalition members
    else
        do nothing this phase
    state ← Frozen Pair Discovery
```

**Algorithm 7** frozenPairDiscovery

```
{INPUT: set of neighbors, $\mathcal{N}_i$, tuning parameter $\alpha$.}
{OUTPUT: surrogate agent as determined by SurrogateDetection().}
for each neighbor $j \in \mathcal{N}_i$ do
    Compute $\mathcal{S}_{i,j}$
for each neighbor $j' \in \mathcal{N}_j$ do
    $val_j = \mathcal{P}_{i,j'}$
    if ($val_j \geq \alpha$) && (historyTable[j'] $\neq -\infty$) then
        $F_i \leftarrow (i', val_j)$
if $|F_i| > 0$ then
    surrogate $\leftarrow$ SurrogateDetection($F_i$)
return surrogate
```

# CHAPTER 6

# CONCLUSIONS AND FUTURE WORK

In this work, we have shown that the use of coalition formation combined with an iterative approximate solution method for solving distributed $k$-coloring problems has shown to be of benefit. This thesis has introduced three new algorithms for coalition formation in specialized instances of Distributed Constraint Optimization Problems (DCOP), distributed $k$-coloring. The Virtual Structure Reduction (VSR) algorithm uses the concept of strict frozen pairs to allow agents to independently identify coalition partners with which they must eventually be assigned the same variable value. We then weaken the idea of strict frozen pairs and look for probabilistically frozen pairs in the $\alpha$VSR algorithm. Propensity captures the idea of two agents having a certain probability of being assigned the same value when the problem instance is solved. This was modelled by both our propensity equation based on shared neighborhoods in a constraint graph, as well as a Markov network inference model of $k$-coloring. We then presented the Hedonic Game (HG) coalition formation algorithm which uses the $\alpha$VSR propensity model, but allows agents to search out for Nash stable coalitions. We have shown that by forming coalitions we can make distributed problems smaller and less complex in most instances. This is especially true within the phase transition region of a $k$-coloring problem where it is known that many difficult to solve instances lie. Efficiency gains can be seen in terms of time, message passing, and solution quality.

There are many opportunities for further investigation using the concept of surrogate led coalitions. An immediate next step is in implementing the coalition formation process with other existing solvers. We chose the Distributed Stochastic Algorithm (DSA) in this work due to the ease of integration. Other iterative approximate solvers, such as the Maximum Gain Messaging (MGM) and Distributed Breakout (DBO) algorithm, should certainly gain from an integration with any of the Virtual Structure Reduction (VSR), $\alpha$VSR (Propensity-based Virtual Structure Reduction), or Hedonic Game (HG) coalition formation techniques. There may even be room for use of the coalition formation process to be integrated with existing optimal solvers such as ADOPT. More so, new solvers specially designed to take advantage of the coalition formation process itself could lead to enhancements of performance for $k$-coloring and other DCOPs.

Another area for future work is the extension of coalition formation using the Markov network-based propensity measures. As we pointed out in Chapter 4, the Markov network can model relationships between agents in a DCOP that the propensity model currently developed can not. Due to the strict reliance of the propensity model on the “not-equals” relationship in $k$-coloring, in order to use our coalition formation technique, one would have to develop a new propensity measure for each new kind of DCOP they are wishing to solve. To avoid this, putting together a Markov network factor table which represents all the possible relationships of a different problem (see Table 4.2) would be an easier task and more general approach. The task then would be to identify particular shared structures from the Markov network, using inference, that have high propensity relationships. Agents could then utilize particular substructures with high propensities in the coalition formation process.

The coalitions that form under our approach are additively separable, and therefore we cannot guarantee a maximum bound on coalition size. This may not be either feasible or desirable in many different problems. To bound coalition size, we would need to integrate a new value function for coalition membership. One possible way of doing so is through the use of fractional hedonic games [5]. A fractional hedonic game uses the average value of coalition membership based on coalition size. Instead of using the additive separability concept we have for the HG-DSA ($v_i(S) = \sum_{j \in S} v_i(j)$), a fractional game uses a value function where $v_i(S) = \frac{\sum_{j \in S} v_i(j)}{|S|}$. This type of value function gives rise to a whole new set of possible partitions for the coalition formation process. For solving DCOPs, we briefly investigated the use of a fractional value function and found that coalitions became locally bounded where agents only wanted to be in coalitions with others which that had common frozen pair relationships. Unfortunately, the stability guarantees for fractional games have only been shown to exist for tree structures with large girth and graphs with large cycles [5]. This is not the case for general graphs and $k$-coloring problems which we investigated in this research.

Another promising way of bounding coalition size under the use of propensity is by implementing a value function that uses the Landscape Theory of Aggregation (LTA). LTA was introduced by Axelrod [4] as a way of studying how political coalitions form in trade agreements and during global conflicts. Axelrod's model uses a *frustration* value that represents each agent's level of satisfaction under a given configuration. Under a configuration, $X$, each agent, $i$, determines how frustrated they are via

$$F_i(X) = \sum_{j \neq i} s_j p_{ij} d_{ij}(X)$$

where $s_j$ is the size (such as Gross Domestic Product, or size of country) of agent $j$, $p_{ij}$ is the propensity between agents $i$ and $j$ to "get along" together, and $d_{ij}$ is the (travel) distance between $i$ and $j$ under configuration $X$. Then, the goal is to minimize the frustration in the system. Although Axelrod was not looking at DCOPs, we can see a natural extension using his formulation of the problem where $p_{ij}$ is our propensity measure, $s_j$ could be modelled to represent the size of the coalition $j$ is in, and $d_{ij}$ could be the distance between $i$ and $j$'s preferences over values assigned to their variables (such as cosine similarity).

A final area of investigation, which we have already begun, is in the development of a better surrogate selection mechanism. In its current state, the VSR, $\alpha$VSR, and HG coalition formation techniques all use a lexicographical selection mechanism. Agents compare IDs and independently determine which of them will be surrogate based on those IDs. Some of our preliminary work suggests that using degree information to determine the surrogate could be useful. In Figure 6.1, we find some gain in the solution quality when agents with higher degree are chosen to represent the coalition. While interesting, it is still undetermined as to why it occurs. Possible reasons for this? It could be that when search terminates with the problem still unfinished, the coalitions with the highest degree, when satisfied, have a larger number of satisfied constraints than we would find with smaller coalitions. It could also be the case of "the rich get richer", or something tied to the concept of preferential attachment. It should also be considered that agents with higher degree have more to gain in taking responsibility for agents with lesser degree as they have more constraints to satisfy. Under degree-based surrogate selection, we anticipate a non-cooperative negotiation process should exist between agents vying for the surrogate position.

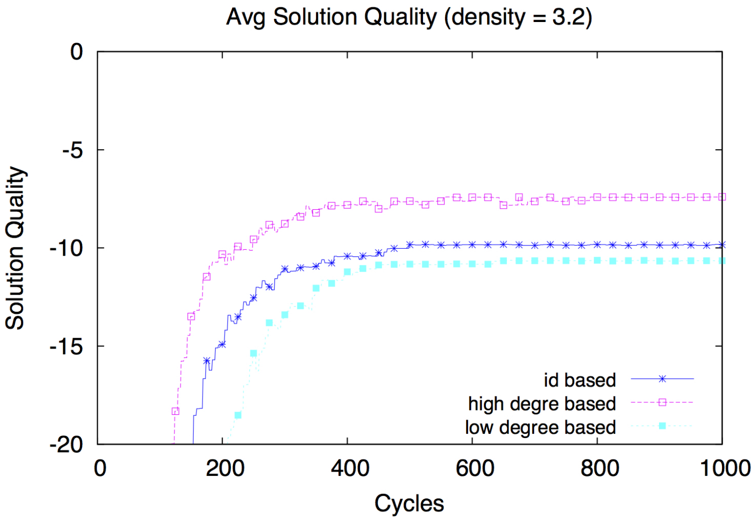


Fig. 6.1: Solution quality using degree-based surrogate selection for $n = 100$ using VSR-DSA.

# REFERENCES

[1] M. Abramson, W. Chao, and R. Mittu, “Design and evaluation of distributed role allocation algorithms in open environments,” DTIC Document, Tech. Rep., 2005.

[2] D. Achlioptas, A. Naor, and Y. Peres, “Rigorous location of phase transitions in hard optimization problems,” *Nature*, vol. 435, no. 7043, pp. 759–764, 2005.

[3] I. Amdouni, P. Minet, and C. Adjih, “Node coloring for dense wireless sensor networks,” *arXiv preprint arXiv:1104.1859*, 2011.

[4] R. Axelrod and D. S. Bennett, “A landscape theory of aggregation,” *British Journal of Political Science*, vol. 23, no. 02, pp. 211–233, 1993.

[5] H. Aziz, F. Brandt, and P. Harrenstein, “Fractional hedonic games,” in *Proceedings of the 2014 International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 5–12.

[6] S. Banerjee, H. Konishi, and T. Sönmez, “Core in a simple coalition formation game,” *Social Choice and Welfare*, vol. 18, no. 1, pp. 135–153, 2001.

[7] A. Bogomolnaia and M. O. Jackson, “The stability of hedonic coalition structures,” *Games and Economic Behavior*, vol. 38, no. 2, pp. 201–230, 2002.

[8] E. G. Boman, D. Bozdağ, U. Catalyurek, A. H. Gebremedhin, and F. Manne, “A scalable parallel graph coloring algorithm for distributed memory computers,” in *Euro-Par 2005 Parallel Processing*. Springer, 2005, pp. 241–251.

[9] C. T. Cannon, R. N. Lass, E. A. Sultanik, W. C. Regli, D. Šišlák, and M. Pechoucek, "Distributed scheduling using constraint optimization and multiagent path planning," in *Proceedings of AAMAS 2010 The 12th International Workshop on Distributed Constraint Reasoning. ACM Press*, 2010, pp. 22–34.

[10] A. C. Chapman, A. Rogers, N. R. Jennings, and D. S. Leslie, "A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems," *Knowledge Engineering Review*, vol. 26, no. 4, pp. 411–444, 2011.

[11] K. Chatterjee, B. Dutta, D. Ray, and K. Sengupta, "A noncooperative theory of coalitional bargaining," *The Review of Economic Studies*, vol. 60, no. 2, pp. 463–477, 1993.

[12] A. Chechetka and K. Sycara, "No-commitment branch and bound search for distributed constraint optimization," in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS).* ACM, 2006, pp. 1427–1429.

[13] P. Cheeseman, B. Kanefsky, and W. Taylor, "Where the really hard problems are," in *Proceedings of the 12th International Joint Conference on Aritificial Intelligence (IJCAI)*, 1991, pp. 331–337.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2001.

[15] J. Culberson and I. Gent, "Frozen development in graph coloring," *Theoretical Computer Science*, vol. 265, no. 1, pp. 227–264, 2001.

[16] J. H. Dreze and J. Greenberg, "Hedonic coalitions: Optimality and stability," *Econometrica: Journal of the Econometric Society*, pp. 987–1003, 1980.

[17] M. Dyer, A. Frieze, T. P. Hayes, and E. Vigoda, “Randomly coloring constant degree graphs,” *Random Structures & Algorithms*, vol. 43, no. 2, pp. 181–200, 2013.

[18] S. Fitzpatrick and L. Meertens, “An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs,” in *Stochastic Algorithms: Foundations and Applications*. Springer, 2001, pp. 49–64.

[19] N. Gemelli, J. Hudack, S. Loscalzo, and J. Oh, “Using coalitions with stochastic search to solve distributed constraint optimization problmes,” in *The 7th International Conference on Agents and Artificial Intelligence (ICAART)*, January 2015.

[20] N. Gemelli, J. Hudack, and J. Oh, “Virtual structure reduction on distributed k-coloring problems,” in *International Conference on Intelligent Agent Technology (IAT-2013)*, November 2013.

[21] N. Gemelli, J. Hudack, and J. C. Oh, “Virtual structure reduction for distributed constraint problem solving,” in *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[22] A. Gershman, A. Meisels, and R. Zivan, “Asynchronous forward bounding,” *Journal of Artificial Intelligence Research (JAIR)*, vol. 34, pp. 25–46, 2009.

[23] A. Gomes, “Multilateral contracting with externalities,” *Econometrica*, vol. 73, no. 4, pp. 1329–1350, 2005.

[24] J. Hajduková *et al.*, “Computational complexity of stable partitions with b-preferences,” *International Journal of Game Theory*, vol. 31, no. 3, pp. 353–364, 2003.

[25] T. P. Hayes, “Local uniformity properties for glauber dynamics on graph colorings,” *Random Structures & Algorithms*, vol. 43, no. 2, pp. 139–180, 2013.

[26] E. Ising, “Contribution to the theory of the ferromagnetism,” *Journal of Physics, Hadrons and Nuclei*, vol. 31, no. 1, pp. 253–258, 1925.

[27] M. Jerrum, “A very simple algorithm for estimating the number of k-colorings of a low-degree graph,” *Random Structures & Algorithms*, vol. 7, no. 2, pp. 157–165, 1995.

[28] E. Karmouch and A. Nayak, “A distributed constraint satisfaction problem approach to virtual device composition,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 11, pp. 1997–2009, Nov 2012.

[29] R. Karp, “Reducibility among combinatorial problems. complexity of computer computations,(re miller and jm thatcher, eds.), 85–103,” 1972.

[30] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe, “Asynchronous algorithms for approximate distributed constraint optimization with quality bounds,” in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 133–140.

[31] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[32] V. Lesser, C. L. Ortiz Jr, and M. Tambe, *Distributed sensor networks: A multiagent perspective*. Springer, 2003, vol. 9.

[33] R. T. Maheswaran, J. P. Pearce, and M. Tambe, “Distributed algorithms for dcop: A graphical-game-based approach,” *Proc. Parallel and Distributed Computing Systems PDCS*, pp. 432–439, 2004.

[34] R. Mailler and V. Lesser, “Solving distributed constraint optimization problems using cooperative mediation,” in *Proceedings of the Third International Joint Conference*

*on Autonomous Agents and Multiagent Systems (AAMAS)*. IEEE Computer Society, 2004, pp. 438–445.

[35] R. T. Mailler, “A mediation-based approach to cooperative, distributed problem solving,” Ph.D. dissertation, University of Massachusetts Amherst, 2004.

[36] S. Minton, M. Johnston, A. Philips, and P. Laird, “Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems,” *Artificial Intelligence*, vol. 58, no. 1, pp. 161–205, 1992.

[37] P. Modi, W. Shen, M. Tambe, and M. Yokoo, “Adopt: Asynchronous distributed constraint optimization with quality guarantees,” *Artificial Intelligence*, vol. 161, no. 1, pp. 149–180, 2005.

[38] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, “An asynchronous complete method for distributed constraint optimization,” in *AAMAS*, vol. 3, 2003, pp. 161–168.

[39] R. Mulet, A. Pagnani, M. Weigt, and R. Zecchina, “Coloring random graphs,” *Physical Review Letters*, vol. 89, no. 26, p. 268701, 2002.

[40] M. E. Newman, “Random graphs as models of networks,” *arXiv preprint cond-mat/0202208*, 2002.

[41] A. Okada, “A noncooperative coalitional bargaining game with random proposers,” *Games and Economic Behavior*, vol. 16, no. 1, pp. 97–108, 1996.

[42] M. Olsen, “On defining and computing communities,” in *Eighteenth Computing: The Australasian Theory Symposium (CATS 2012)*, Melbourne, Australia, 31 January – 3 February 2012, p. 97.

[43] J. Pearce, R. Maheswaran, and M. Tambe, “Solution sets for dcops and graphical games,” in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, 2006, pp. 577–584.

[44] J. P. Pearce, *Local optimization in cooperative agent networks*. ProQuest, 2007.

[45] J. P. Pearce, R. T. Maheswaran, and M. Tambe, “How local is that optimum? k-optimality for dcop,” in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM, 2005, pp. 1303–1304.

[46] J. P. Pearce and M. Tambe, “Quality guarantees on k-optimal solutions for distributed constraint optimization problems.” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007, pp. 1446–1451.

[47] A. Pectu and B. Faltings, “Odpop: an algorithm for open/distributed constraint optimization,” in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, no. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 703.

[48] A. Petcu, *A class of algorithms for distributed constraint optimization*. IOS Press, 2009, vol. 194.

[49] A. Petcu and B. Faltings, “A scalable method for multiagent constraint optimization,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 5, 2005, pp. 266–271.

[50] A. Petcu, B. Faltings, and R. Mailler, “Pc-dpop: A new partial centralization algorithm for distributed optimization.” in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 7, 2007, pp. 167–172.

[51] A. Petcu, B. Faltings, and D. C. Parkes, “Mdpop: Faithful distributed implementation of efficient social choice problems,” in *Proceedings of the Fifth International Joint*

*Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, 2006, pp. 1397–1404.

[52] C. Portway and E. H. Durfee, “The multi variable multi constrained distributed constraint optimization framework,” in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 1385–1386.

[53] C. J. Preston, *Gibbs states on countable sets*. Cambridge University Press London, 1974, vol. 96.

[54] P. Prosser, “An empirical study of phase transitions in binary constraint satisfaction problems,” *Artificial Intelligence*, vol. 81, no. 1, pp. 81–109, 1996.

[55] T. Rahwan and N. R. Jennings, “Coalition structure generation: Dynamic programming meets anytime optimization.” in *AAAI*, vol. 8, 2008, pp. 156–161.

[56] T. Rahwan, T. P. Michalak, E. Elkind, P. Faliszewski, J. Sroka, M. Wooldridge, and N. R. Jennings, “Constrained coalition formation.” in *AAAI*, vol. 11, 2011, pp. 719–725.

[57] T. Rahwan, S. D. Ramchurn, V. D. Dang, A. Giovannucci, and N. R. Jennings, “Anytime optimal coalition structure generation,” in *AAAI*, vol. 7, 2007, pp. 1184–1190.

[58] D. Ray and R. Vohra, “Coalition formation,” New York University, Tech. Rep., 2013.

[59] A. Romero-Medina *et al.*, “Stability in coalition formation games,” *International Journal of Game Theory*, vol. 29, no. 4, pp. 487–494, 2001.

[60] Y. Ruan, H. Kautz, and E. Horvitz, “The backdoor key: A path to understanding problem hardness,” in *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press;, 2004, pp. 124–130.

[61] S. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*. Prentice hall Upper Saddle River, NJ, 2010.

[62] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Coalition structure generation with worst case guarantees," *Artificial Intelligence*, vol. 111, no. 1, pp. 209–238, 1999.

[63] V. Schuppan, "Towards a notion of unsatisfiable cores for ltl," in *Fundamentals of Software Engineering*, ser. Lecture Notes in Computer Science, F. Arbab and M. Sirjani, Eds. Springer Berlin Heidelberg, 2010, vol. 5961, pp. 129–145.

[64] B. Selman, H. A. Kautz, and B. Cohen, "Noise strategies for improving local search," in *AAAI*, vol. 94, 1994, pp. 337–343.

[65] M. C. Silaghi and M. Yokoo, "Nogood based asynchronous distributed optimization (adopt ng)," in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, 2006, pp. 1389–1396.

[66] B. Smith and M. Dyer, "Locating the phase transition in binary constraint satisfaction problems," *Artificial Intelligence*, vol. 81, no. 1, pp. 155–181, 1996.

[67] C. Theocharopoulou, I. Partsakoulakis, G. A. Vouros, and K. Stergiou, "Overlay networks for task allocation and coordination in dynamic large-scale networks of cooperative agents," in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM, 2007, p. 55.

[68] S. Ueda, A. Iwasaki, M. Yokoo, M. C. Silaghi, K. Hirayama, and T. Matsui, "Coalition structure generation based on distributed constraint optimization," in *AAAI*, vol. 10, 2010, pp. 197–203.

[69] M. Vinyals, E. Shieh, J. Cerquides, J. A. Rodriguez-Aguilar, Z. Yin, M. Tambe, and E. Bowring, "Quality guarantees for region optimal dcop algorithms," in *The 10th*

*International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 133–140.

[70] R. Williams, C. Gomes, and B. Selman, "On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search," *Structure*, vol. 23, p. 4, 2003.

[71] R. Williams, C. P. Gomes, and B. Selman, "Backdoors to typical case complexity," in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 18. Citeseer, 2003, pp. 1173–1178.

[72] D. Y. Yeh, "A dynamic programming approach to the complete set partitioning problem," *BIT Numerical Mathematics*, vol. 26, no. 4, pp. 467–474, 1986.

[73] W. Yeoh, A. Felner, and S. Koenig, "Bnb-adopt: An asynchronous branch-and-bound dcop algorithm," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS).* International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 591–598.

[74] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 10, no. 5, pp. 673–685, 1998.

[75] M. Yokoo and K. Hirayama, "Algorithms for distributed constraint satisfaction: A review," *Autonomous Agents and Multi-Agent Systems (AAMAS)*, vol. 3, no. 2, pp. 185–207, 2000.

[76] M. Yokoo, *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-agent Systems*, 1st ed. Springer Publishing Company, Incorporated, 2012.

[77] M. Yokoo and K. Hirayama, “Distributed breakout algorithm for solving distributed constraint satisfaction problems,” in *Proceedings of the Second International Conference on Multi-Agent Systems*, 1996, pp. 401–408.

[78] W. Zhang, G. Wang, and L. Wittenburg, “Distributed stochastic search for constraint satisfaction and optimization: Parallelism, phase transitions and performance,” in *Proceedings of AAAI Workshop on Probabilistic Approaches in Search*, 2002.

[79] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg, “An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks,” in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 14, no. 18, 2003, pp. 185–192.

# VITA

NAME OF AUTHOR: Nathaniel Gemelli

PLACE OF BIRTH: Kailua, HI, USA

DATE OF BIRTH: September 19, 1978

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

- State University of New York at Oswego, USA
- Syracuse University, USA

DEGREES AWARDED:

- Computer Science BS, 2001, SUNY Oswego, USA
- Computer Science MS, 2006, Syracuse University, USA

PROFESSIONAL EXPERIENCE:

- June 2000 to August 2000 - Mathematician Aid, Air Force Research Laboratory, Information Directorate - Rome, N.Y.
- June 2001 to present - Computer Scientist, Air Force Research Laboratory, Information Directorate - Rome, N.Y.

GRANTS:

- AFRL LRIR (Laboratory Research Initiation Request) Grant from Air Force Research Laboratory of Scientific Research (AFOSR), 2014-2016, 3 years, $630k
- AFRL LRIR (Laboratory Research Initiation Request) Grant from Air Force Research Laboratory of Scientific Research (AFOSR) in 2009, 1 year, $100k

PUBLICATIONS:

- Proceedings

  N. Gemelli, J. Hudack, S. Loscalzo, and J.C. Oh. Using coalitions with stochastic search to solve distributed constraint optimization problems. In *International Conference on Agents and Artificial Intelligence (ICAART-2015)*, January 2015.

  N. Gemelli, J. Hudack, and J.C. Oh. Virtual structure reduction on distributed k-coloring problems. In *International Conference on Intelligent Agent Technology (IAT-2013)*, November 2013.

  J. Hudack, N. Gemelli, and J.C. Oh. Modeling spatial information diffusion with self-interested agents. In *IEEE/WIC/ACM International Conference on Web Intelligence*, November 2013.

  N. Gemelli, J. Hudack, and J.C. Oh. Virtual structure reduction for distributed constraint problem solving. In *The Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*, July 2013.

  J. Hudack, N. Gemelli, and J.C. Oh. Evolution of cooperation in packet forwarding with the random waypoint model. In *International Conference on Agents and Artificial Intelligence (ICAART)*, February 2013.

N. Gemelli, J. Hudack, and J.C. Oh. Adopting a risk-aware utility model for repeated games of chance. In *Proceedings of The Sixth Starting Artificial Intelligence Research Symposium (STAIRS 2012)*, August 2012.

R. Wright and N. Gemelli. Adaptive state space abstraction using neuroevolution. *Agents and Artificial Intelligence*, 67:84–96, 2010.

R. Wright and N. Gemelli. State aggregation for reinforcement learning using neuroevolution. In *International Conference on Agents and Artificial Intelligence (ICAART)*, pages 45–52, 2009.

M. Tavana, N. Gemelli, and R. Wright. A vehicle-target simulation model for network-centric joint air operations. In *Industrial Engineering and Engineering Management, 2007 IEEE International Conference on*, pages 1767–1771. IEEE, 2007.

N. Gemelli, R. Wright, J. Lawton, and A. Boes. Asynchronous chess competition. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1445–1446. ACM, 2006.

N. Gemelli, R. Wright, and R. Mailer. Asynchronous chess. In *Proceedings of The 2005 AAAI Fall Symposium Workshop on Coevolutionary and Coadaptive Systems*, November 2005.

J.C. Oh, N. Gemelli, and R. Wright. A rationality-based modeling for coalition support. In *Fourth International Conference on Hybrid Intelligent Systems*, pages 172–177. IEEE, 2004.

- Technical Reports

S. Loscalzo, N. Gemelli, and R. Wright. Machine intelligence in-house final technical report. Technical report, Air Force Research Laboratory, 2013.

J. Hudack, N. Gemelli, and M. Scalzo. Local utility estimation in model-free, multi-agent environments. Technical report, Air Force Research Laboratory, 2010. AFRL-IF-RS-TP-2011-42.

R. Wright, J. Hudack, N. Gemelli, S. Loscalzo, and T.K. Lue. Agents technology research. Technical report, Air Force Research Laboratory, 2010. AFRL-RI-RS-TR-2010-057.

N. Gemelli, J. Lawton, R. Wright, and A. Boes. Asynchronous chess: A real-time, adversarial research environment. Technical report, Air Force Research Laboratory, 2006. AFRL-IF-RS-RT-2006-116.