COMBINING PREFERENCE AND FEASIBILITY IN MULTI AGENT LOCAL SEARCH

by

Tedi Susanto
B.A.Sc, University of British Columbia, 1996

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
July 2004

APPROVAL

Name:	Tedi Susanto
Degree:	Master of Science
Title of thesis:	Combining Preference and Feasibility in Multi Agent Local Search
Examining Committee:	Dr. Funda Ergun Chair
	Dr. William S. Havens Associate Professor School of Computing Science Senior Supervisor
	Dr. Michael Brydon Assistant Professor Faculty of Business Administration Supervisor
	Dr. Lou Hafer Associate Professor School of Computing Science SFU Examiner
Date Approved:	July 15, 2004

Abstract

Within a multi agent system, rational agents communicate and negotiate to solve a common global constraint satisfaction problem. Given that there are many possible solutions and that each agent has its own preference, such negotiation should not only lead to just any solution, but one that is fair and mutually beneficial to the all the participants. The Nash bargaining solution to the bargaining problem can be applied to a cooperative multi agent environment in order to achieve pareto optimal solutions, where no party can benefit without causing harm to others. However, given the complexity of most real-world problems, computing the so-called bargaining set is NP-hard. Furthermore, while it would be preferred to use local search algorithms to ensure fairness, as they do not impose ordering of agents, the presence of non-binary constraints has a detrimental effect. That is, any attempt to repair a variable in a constraint where the current states of the other variables already violate the constraint is futile.

We propose a marginalization strategy that approximates the bargaining set and allows for a more effective local search in the presence of non-binary constraints. This technique considers both feasibility and preferences and is able to better distinguish between infeasible states while searching for solutions. We evaluate this method on sport scheduling problem using all-different constraints with preferences and on random binary CSPs with preferences. Experimental results show that it has significant advantage over local search using only min-conflict heuristic.

To my family

Acknowledgments

I would like to thank my senior supervisor Bill Havens for his guidance, motivations and patience throughout this research. I must thank my supervisor Michael Brydon for providing me with the business perspective of this research and his conscientious attention to detail while reviewing this thesis. I am grateful to Lou Hafer for being in my examining committee, and Michael Horsch for providing me with the source code of random CSP generation.

Finally, I would like to thank my parents Djoni Susanto and Lindawati for their support, and my wife Alicia Lin for her love and encouragement.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivations

In constraint optimization problems, we are interested in finding reasonably good, if not the best, solutions based on some evaluation criteria. For example, in sports league scheduling, there could be many schedules that satisfy the constraints imposed, but we may want to find ones that minimize the travel cost of the teams. When solving such problems, there are two factors that have to be considered: feasibility and preference. These two often do not go hand-in-hand, what is feasible may not be desired, similarly what is desired may not be feasible. Traditional approaches such as branch and bound, consider feasibility first and then preference in its search. That is, branch and bound searches for feasible (partial) solutions first, then evaluates these to determine their desirability. However, given both of these criteria, one may benefit by considering both simultaneously, resulting in a more efficient search for good solutions.

Balancing the two requirements gets even more difficult when many parties are involved. Each will have its own preferences and the solution that will be acceptable will be one that is fair and mutually beneficial to all. A negotiation strategy is necessary in order achieve this and the Nash bargaining solution is a suitable candidate, given its pareto optimality and independence of utility scales. Pareto optimality means that no one party can take action unilaterally without causing harm to the others and independence of utility scales means that different parties can use different method of valuation of the agreement and yet the same solution can be obtained regardless. In order to apply the Nash bargaining solution to the problem, one must first compute the bargaining set. However, computing the global bargaining set is equivalent to taking the cross-product of all the constraints, and therefore it is NP-hard. Imagine a roomful of team managers trying to schedule a season of a sport tournament, and they have to consider all possible configurations of games which are exponential in numbers.

Instead of applying the Nash bargaining solution to the global problem, we can apply it locally. In terms of sport scheduling, we can think of it as having every pair of teams negotiating just for the match between the two of them instead of having all the teams negotiating for the complete tournament schedule. The result of the reduction in computational complexity is that all the locally pareto optimal agreements taken together do not necessarily produce a globally optimal solution. For problems where the true global optimum is not required, this approach provides fast sub-optimal solutions that maybe are good enough. Another difficulty with this approach is, before we could negotiate locally, we would need to express our complex preference in terms of the current possible bargains. Due to the interdependence between different negotiations, computing an informative local bargaining set can be difficult.

Before going further and addressing this problem, a brief discussion relating preferences, bargaining set, and multi agent constraint optimization is in order. In constraint optimization, an objective function is used to evaluate the feasible solutions. This could be minimizing cost, maximizing profit or others. A single agent (which can represent a team in sport scheduling, for example), can be represented by a constraint which lists all the feasible solutions with their associated preferences. That is, the objective function or preferences of an agent can be represented as a constraint with preferences. In a multi agent system (MAS) environment, there are many of these agents and they are related by shared constraints or variables. As each of these agents has their own preferences, solving a multi agent constraint optimization is equivalent to finding a consistent global solution such that all the preferences are maximized[1]. When a variable is shared

[1]These preferences may be maximized according to some social welfare function, such as utilitarian, egalitarian, etc.

between two agents, they can negotiate as to which value that variable should take. Prior to the negotiation, the bargaining set has to be computed, that is, for each of the domain values of the variable, a valuation is required from either agents. Therefore, each agent needs to *marginalize* its preference (which can be derived from many variables) and obtain an approximation of what each domain value of the negotiated variable is worth.

Hence, the problem lies in obtaining an accurate marginalization. One approach to solve this is to use the traditional constructive search marginalization. However, such marginalization requires constructive search, where there exists ordering of agents, and it considers feasibility first, and then preference. Using the sport scheduling example, we would start with an empty schedule and assign some sort of order to the teams, so team 1 will first negotiate with team 2 on their game. Then team 1 will negotiate with team 3 on their game (keeping in mind the agreement made with team 2), and so on. Thus the resulting schedule may arbitrarily favor the higher order teams and is hardly fair. The alternative is a local search approach, where all the teams are equal and all of them start with their own best schedules. Of course, it is highly unlikely that all the best schedules taken together will give a feasible solution, so the teams have to negotiate to find a preferred compromise. A team may then have to perform several local negotiations in order to achieve this compromise. As rounds of negotiations continue, a team may find that the previously made agreements might no longer look as good as they were and re-negotiations may be necessary. This is an iterative repair approach where the negotiation has to be done repeatedly until everyone is satisfied. We have adopted this approach since it does not assign precedence to the parties and we have assumed a multi agent environment where all agents are equal.

Now, how can one perform marginalization in local search? Particularly one that considers both feasibility and preference simultaneously? In other words, how can one compute a local bargaining set that reflects both the feasibility and preference of an agent? We argue that such bargaining set should not only include feasible bargains but also preferred infeasible bargains. The reason behind this strategy is that the result of local agreements can be viewed as a tentative (perhaps infeasible) solution, which may become feasible as we re-negotiate. When negotiating locally, the parties would have to estimate how the agreements that were made previously are going to affect the current one, due to the interdependence between the agreements. Therefore, each party has to consider the preferences of itself and others (as reflected in previous deals), as well as the feasibility of the problem, while negotiating for the current deal. It turns out that this is not an easy task because we typically can only value a complete set of deals. In our sport scheduling example, if we are interested in minimizing the traveling cost, we need to know our complete tour in order to calculate the cost. When negotiating with only one other team at a time, we need to estimate what the expected cost would be given the previous deals. To further complicate the matter, what if all the previous deals taken together does not lead to a feasible solution? That is, we are faced with a problem of not being able to distinguish infeasible solutions (i.e., they are all just *bad*). Further, what if the feasible bargain is very unattractive? We may want to explore more promising infeasible bargains.

We propose a strategy that maps the complex agent preferences into a valuation used for bargaining. When the preferences are represented as constraints with preference, this process is also called constraint projection or constraint marginalization. We can view marginalization as a procedure that summarizes or approximates the preference that depends on many variables into one that depends only on one variable[2]. However, existing marginalization functions are only used in constructive search. As such, they are fairly simple and do not distinguish between infeasible states or considers infeasible states that may lead to preferred solutions. Instead of presenting specific functions, we identified four properties that are desirable when solving multi agent constraint optimization problem. The marginalization function should: be able to distinguish between feasible states, be able to distinguish between infeasible states, be able to explore promising infeasible states even at the cost of feasibility, and contain the elements of both preference and feasibility. We argue that it is sometimes reasonable to choose preference over feasibility since we are performing repeated negotiations and other parties may move to their preferred states and eventually arrive at a good feasible

[2]In general, one can marginalize the preference with respect to a subset of the original variables.

solution. However, we can also expect that such policy can often force us into infeasible region and never arrive at a solution. This is caused by the infeasible states having too high of preference values and the feasible states having too low of preference values. In order to address this issue, we introduce a preference skewing mechanism. Here, the preference values are updated such that difference between high and low values is reduced. This method was successful in overcoming the over-dominance problem and we were able to find good solutions.

We called the previous approximation strategy *combined marginalization*. Armed with this marginalization, each party can locally negotiate and re-negotiate until a solution is found. Using sport scheduling and random CSP as our test problems, we compared our technique against simple min-conflict heuristic, where only feasibility is considered during search. We found that our approach outperforms min-conflict, both in solution quality and fairness. Furthermore, our experiments showed that the technique is capable of finding close to optimal solutions, producing solutions with average maximum of about 90% of the optimal.

1.2 Thesis Outline

This thesis investigates the use of Nash bargaining solution in solving multi agent constraint optimization problem. However, prior to applying the Nash bargaining solution, the bargaining set would have to first be computed. When a multi agent system consists of numerous agents, each with many interdependent variables and constraints, computing this set is equivalent to computing multiple-items, multiple-players bargaining set. This is NP-hard and may not be practical. An approximation using local negotiation and marginalization strategy which combines both preference and feasibility is proposed and its performance evaluated on two classes of problems: sports league tournament scheduling and random CSPs.

In Chapter 2, we first describe the concept of constraint satisfaction and multi agent system. Then we introduce the notion of valued CSP and describe constraint combination and projection in terms of semiring-based CSP. We define a bargaining problem in the context of MAS and valued CSP and outline how the Nash bargaining solution can be applied to multi agent optimization problem. This is followed by a review of sports scheduling problem and random CSP.

In Chapter 3, we first formalize our multi agent model and describe the current constraint marginalization methods. We presented an example to show how it is inadequate for non-binary constraints. We then outline the requirements for our combined marginalization and provide specific functions for both sport scheduling and random CSP. We also describe our preference skewing mechanism in order to escape from strongly preferred but infeasible regions. Chapter 4 presents the algorithm used in our experimental study, along with its results and discussions. We showed that our approach is capable of finding fair and close to optimal solutions, for both the sport scheduling and randomly generated CSPs. Finally, the conclusion and future work are given in Chapter 5.

Chapter 2

Background

2.1 Constraint Satisfaction Problem

2.1.1 Definition and Representation

A Constraint Satisfaction Problem (CSP) is a general framework for modeling various problems in Artificial Intelligence (AI). Such problems include scheduling, resource allocation, and configuration. Formally, a CSP is a triple $(V, \mathcal{D}, C)$ where

- $V = \{x_1, ..., x_n\}$ is a set of variables,
- $\mathcal{D} = \{D_1, ..., D_n\}$ is a set of domains, such that x_i takes value from D_i, and
- C is a set of constraints, such that a constraint c is a subset of the Cartesian product $D_1 \times \cdots \times D_n$ consisting of compatible values or *tuples* for the involved variables.

When the number of involved variables in constraint c is one or two, c is called *unary* or *binary* constraint respectively. All other constraints are considered non-binary. A binary CSP is one that only contains unary and binary constraints. Solving a CSP is equivalent to finding an assignment to all variables such that all constraints are satisfied. As CSP is NP-complete, a trial and error search for a solution is inevitable for solving it.

A classical example of a CSP is the n-queens problem. The objective is to place n chess queens on an $n \times n$ board such that these queens do not threaten each other. A solution for 4-queens problem is shown in figure 2.1.

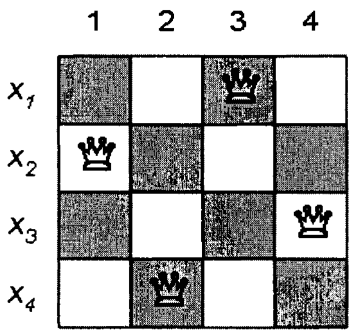


Figure 2.1: 4-queens problem

c_{12}

x_1	x_2	allowed
1	1	no (same column)
1	2	no (same diagonal)
1	3	yes
1	4	yes
2	1	no (same diagonal)
2	2	no (same column)
2	3	no (same diagonal)
2	4	yes
⋮	⋮	⋮

Figure 2.2: Example of a constraint table

To formulate the 4-queens problem, we can assign each row a variable, that is $V = \{x_1, x_2, x_3, x_4\}$. Each of this four variables can take one of the four columns as its value, i.e., $D_1 = D_2 = D_3 = D_4 = \{1, 2, 3, 4\}$. The constraints can represented as inequalities: $\forall i, j, x_i \neq x_j$ which prohibits the queens to be in the same column, and $\forall i, j$, if $x_i = a, x_j = b$, then $|i - j| \neq |a - b|$ which disallows queens to be in the same diagonal. Equivalently, we can represent the constraints as a table that allows compatible tuples. Another alternative is to represent it as a boolean *conflict matrix*, where allowed tuples are given the value of 1's and tuples that are forbidden are given the value of 0's. An example of constraint table and constraint matrix for constraints between x_1 and x_2 is given in figures 2.2 and 2.3.

c_{12}		x_2			
		1	2	3	4
	1	0	0	1	1
x_1	2	0	0	0	1
	3	1	0	0	0
	4	1	1	0	0

Figure 2.3: Example of a constraint matrix

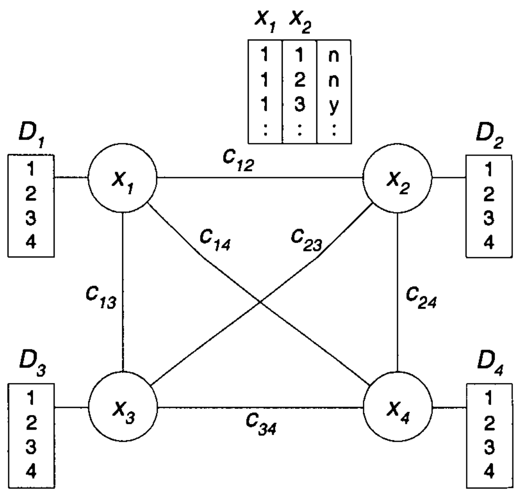


Figure 2.4: Constraint graph for 4-queens problem

A CSP can also be represented as *constraint graphs* where each vertex represents a variable in V and each edge represents a constraint in C. The constraint graph for the above 4-queens problem is shown in figure 2.4.

2.1.2 Algorithms

There are two main classes of algorithms within CSP research: consistency and search. Consistency or filtering algorithms take the constraints of the problem and use them to remove any domain values that cannot be part of any solution. Such algorithms are very powerful since they can greatly reduce the search space and minimize futile search. One family of such algorithms is Arc-Consistency or

AC*x* (where *x* represents a number, e.g., AC1, AC3) [16]. An arc (x_i, x_j), which corresponds to a binary constraint c_{ij} in a constraint graph of a CSP, is *arc-consistent* if and only if for every value of v_i in the domain of x_i, there exists a value in the domain of v_j that is compatible with $x_i = v_i$. AC1 achieves an arc-consistent CSP by revising the domain of variables by removing incompatible values for each constraints. Note that the arc consistency algorithm only checks for local consistency and therefore it does not guarantee to find solutions.

Typically, consistency alone cannot solve the CSP thus a search is required. Search algorithms can be divided into two groups: constructive and local. Constructive search or backtrack search solves the problem by extending a partial assignment ensuring that this assignment or instantiation never violates any constraints. In chronological backtrack search, if the partial assignment cannot be extended any further, the algorithm undoes its last decision and select the next feasible value. Using the n-queens example, constructive search will start with an empty board, place the queens one at a time (in a systematic order) ensuring that the current placement does not conflict with previously placed queens. Constructive search systematically explores the search space and is complete. That is, the search can find all the solutions if required or can determine if in fact there is no solution. It can also take advantage of consistency algorithms by applying them after each assignment is made. This has the effect of pruning the search space further thus reducing the search effort.

Local search or iterative improvement, on the other hand, starts with a full instantiation that may not be feasible, and iteratively repairs or reassigns values to the variables to improve the current instantiation according to some objective function. Usually, there is a heuristic that guides the improvement; a popular heuristic is to minimize the number of conflicts, or *min-conflict* [18]. The min-conflict procedure is to select a variable that is in conflict, and assigning it a value that minimizes the number of conflicts (breaking ties randomly). Minton et al [18] have shown that an iterative improvement algorithm using min-conflict heuristic as its repair method, performed more efficiently compared to traditional constructive backtracking algorithms for certain large-scale problems. Similar results were also obtained by Selman et al [30] with their GSAT algorithm, an iterative improvement method for solving hard satisfiability problems. Such algorithms, however, can get stuck at locally optimal points that are not solutions. These are points where no more change in any variable assignments can improve the current instantiation and yet the current instantiation is still infeasible. Techniques for escaping from these points include random restarts [30] and Morris [19] breakout algorithm where weights are assigned to violated constraints. The breakout method dynamically updates the weights during the algorithm execution and it allows the search to violate more constraints with lesser weights at the cost of satisfying constraints with heavier weights.

There are also *meta-heuristics*, or master strategies that guide and modify the search heuristics, commonly associated with local search for optimization problems. Two well-known meta-heuristics are Tabu Search (TS) and Simulated Annealing (SA). In TS [8], the search evaluates neighboring states according to some criteria and moves to the better state, similar to min-conflict local search. However, TS allows for moving to states that is worse than the current state. Repeating this idea creates the possibility of endless cycle, and to prevent this, TS uses a *tabu list* to maintain a list of forbidden moves. That is, if we made a move $x \rightarrow x'$, the reverse move $x' \rightarrow x$ is forbidden for the next s moves (s is the size of the first-in-first-out list). There is also an *aspiration criterion* which allows forbidden moves if certain conditions are met, for example, when a tabu solution is better than any previously seen solutions. TS has been shown to perform well in many applications [8], including sport scheduling [9]. SA is motivated by the physical annealing process, where material is heated and slowly cooled into a uniform structure. SA mimics this process and is used for local search optimization [14]. As with TS, SA allows for moving to worse states. However, the proposed next state is chosen at random. If the next state is better, it is always accepted. If it is worse, the probability that it is accepted depends on the change in the cost function and the current "temperature" of the system. That is, worse move is accepted with probability $p = e^{\triangle/T}$, where $\triangle$ is the change in the objective function and T is a control parameter called the temperature. As the temperature decreases, the probability of accepting worse moves decreases. SA algorithm requires a cooling schedule whereby the temperature is slowly decreased, and at each temperature a certain number of iteration (random moves) is performed. The argument is that with slow enough cooling schedule, a global optimum can be found. SA has also been used extensively in many applications, including solving VLSI layout problems and factory scheduling.

Local search is non-systematic, therefore it can be very efficient on some large problems, but it is also incomplete. Going back to our n-queens problem, a local search will start with some initial placement (typically random) of all the queens on the board, choose a queen that is being threatened, and move it to a new position so as to minimize the number of attacks. It repeatedly performs such moves until a solution is found. If it discovers that no more such moves can be made and the current placement is not a solution, it has hit a local optimum. The search then can simply restart with a new random placement of all queens and start all over again.

The two search methods have their advantages and disadvantages. Their effectiveness is also dependent on the problems to be solved. There are also hybrid approaches that combine the advantages of either algorithms, such as that proposed by Jussien and Lhomme [13]. Their technique performs a local search over partial assignments instead of complete assignments, and uses filtering techniques and conflict-based techniques to efficiently guide the search.

Two areas in which the CSP formalism has been extended are Multi Agent Systems (MAS) [36] and Valued CSP (VCSP) [2]. MAS attempts to address the fact that problems can be distributed and a centralized approach may not be practical or feasible, while VCSP deals with problems that may have non-crisp constraints–that is some constraints are more important than others or there are varying degree of constraint satisfaction or violation. These are discussed in sections 2.2 and 2.3.

2.2 Multi Agent Systems and Distributed CSP

A distributed CSP (DCSP) is a CSP in which the variables and constraints are distributed among multiple autonomous agents [36]. An agent is an entity, such as a software process, that can sense its environment and act to change it. A multi agent system (MAS) consists of many such agents in a shared environment. Often, these agents are designed to be rational, that is, their behavior is consistent with maximizing their own preferences over the states of the environment. When a common global solution is desired, they can also interact and coordinate their actions in order to reach this goal. There are two kinds of constraints in a DCSP. *Inter-agent* constraints are the ones between agents and *intra-agent* constraints are the ones within one agent. Intra-agent constraints can be solved with traditional CSP methods, but it is the solving of inter-agent constraints that is the main research goal of DCSP. Thus the objective is to find a value assignment to variables that satisfies these inter-agent constraints, which can be viewed as to achieve coherence or consistency among agents [36].

It must be noted that MAS is different from distributed/parallel processing. The latter is primarily concerned with efficiency and is typically deployed by a single designer. MAS is concerned with solving distributed CSP, where the knowledge of the problem (i.e., variables and constraints) is distributed among automated agents. Each of these agents may be designed and owned by different organization thus a centralized algorithm necessitates significant communication cost for gathering all the parts of the problem, and translation cost for converting these parts to an exchangeable format. Furthermore, there can be organizational security or privacy issues which makes such full disclosure undesirable.

The key problem of MAS is therefore one of coordination. How does one ensure that the agents act coherently in making their decisions and avoid harmful interactions? The agents in a MAS[1] can be in cooperation or competition. Cooperation implies a shared goal amongst the agents whereas competition implies that one agent can gain only at the expense of another [17]. Cooperative agents will therefore "behave themselves" by acting for the common good, and ensuring that they coordinate their interactions to achieve the global objective. Competitive agents typically have their own preference for how things should be and are interested in maximizing these preference. Often, these individual preferences are in conflict with one another, and in order to achieve a stable equilibrium, coordination through negotiation, bargaining or mediated consensus is required.

[1]A finer distinction can be made on DCSP or DAI (distributed artificial intelligence) systems. DPS or distributed problem solving assumes a single system architect, with an overall global goal, and agents that are responsible for solving particular subgoals. On the other hand, MAS are made up of heterogenous, autonomous agents of distinct origin, that share the same environment. However, we have followed the current practice of using the term "multi agent systems" to describe all of the DAI systems.

There are two approaches in solving this coordination problem. One is to design the individual agent such that it reasons about its local actions and the anticipated actions of others and ensure that the community acts in a coherent manner [12]. Here, the designer's job is to define a protocol (i.e., agent communication language, negotiation process, etc) for the agent interaction. Rosenschein and Zlotkin [25] investigated the process whereby agents iteratively negotiate in order to arrive at mutually beneficial agreement. They defined a protocol and negotiation strategy based on the Nash bargaining problem [20], that results in agents reaching consensus at a single point within the negotiation set. The second approach is to design the rules of public interactions such that the system as a whole produces a desirable behavior regardless of the private behaviors of the individual agents [25]. An environment must be developed to govern the allowable interactions between the agent and environment. In this case, coordination results from indirect agent interactions with simulated markets or ecosystems. An example of this is WALRAS [35], a system for defining MAS as computational economies, a concept which the author called *market-oriented programming.* A computational economy is created by populating the system with consumer agents that consumed a set of goods produced by producer agents. These goods are exchanged (bought and sold through auctions) such that the consumers maximize their utilities and the producers maximize their profits. Equilibrium is achieved when the total amount consumed equals the amount produced, plus the initial endowment. This concept was also applied to solving propositional satisfiability in MarketSAT [34]. Although the performance of MarketSAT is poor compared to the centralized GSAT algorithm, it shows that highly decentralized market-based technique is capable of solving combinatorial problems.

Both constructive and local search techniques can be extended to solving DCSP. Among the algorithms presented by Yokoo [36], are the asynchronous backtracking (ABT) and distributed breakout. ABT allows agents to make decisions asynchronously by taking advantage of the multiple processors available in a distributed environment. This is an improvement over synchronous backtracking, which is a simple extension of centralized backtracking. In contrast to synchronous backtracking, where only one agent at a time can assign a value to its variable, ABT allows multiple agents to assign values to their variables in parallel. However, there exists ordering of agents, and when two agents are connected by a constraint the lower agent has to accommodate the higher agent. Of course, if two agents are not connected, they both can run concurrently. The distributed breakout algorithm extends Morris [19] centralized breakout concept to distributed agents. In this algorithm, neighboring agents exchange values of possible improvements, and only the agent that can maximally improve the evaluation value can change its value. Here, all agents are equal and non-neighbor agents can simultaneously change their values.

Many real-world problems can be modeled into DCSP. One application problem that used DCSP techniques is concert venue equalization problem [22]. Typically, due to the spatial arrangement of the speakers, shape and size of the room, the sound quality produced is different in different parts of the room. The objective is then to provide an even sound quality by using a set of equalizers. In this problem, the agents corresponds to microphones scattered throughout the room, and they coordinate their actions by controlling the equalizers. Another problem is the nurse time-tabling task [32]. This problem involves assigning nurses to shifts in each department of a hospital and planning for their transportation from their homes to the hospital. As the departments are essentially independent, the time-tabling can be handled by different agents. However, there exists inter-agent constraints involving their transportation in order to minimize the transportation cost.

2.3 Valued CSP

Many real world problems often cannot be naturally expressed using traditional/classical CSP framework in which the constraints are either satisfied or not satisfied. For example, some problems may be over-constrained and no solution exists, but a close approximation may still be desirable. Perhaps there are degrees of importance between constraints and it is acceptable to violate some but not others (i.e., soft and hard constraints). There may be cases where knowledge of the problem is incomplete therefore constraints may or may not exists in the actual problem. Weighted CSP, probabilistic CSP, and fuzzy CSP are among the many frameworks that deals with such issues. For example, in weighted CSP, each constraint is assigned an associated weight. The more important the constraint is the higher its weight. The objective is to find solutions that maximize the sum of all constraint weights. In such cases, we can afford to violate less important constraints in order to satisfy the more important ones. A finer approach is to assign weight or preference to each tuple in the constraint relations. For example, in our n-queens problem, we can assign some numeric values to each tuple in the constraint table (see figure 2.2 on page 8). Rather than simply indicating whether the tuple is allowed or not, we can express our preference and search for solutions that maximize this preference. One of the well-studied instances of weighted CSP is the *MAX-CSP* (or *maximal* CSP), where objective to satisfy maximal number of constraints [5].

There are also problems where many solutions exist and it is desirable to find the best one with respect to some evaluation criteria. These are constraint optimization problems or COPs, where objective function exists and the quality of a solution can be measured. Optimization problems are often solved by the so called branch and bound algorithm. This algorithm requires a heuristic function that can estimate the quality of each partial assignments. Branch and bound search behaves like a constructive search except that as soon as a value is assigned to the variable, the value of the heuristic function for the assignment is computed. If this value is less than the bound (usually the current best value of the objective function), the sub-tree under the current partial assignment is pruned to avoid its useless exploration.

All these frameworks contain the notion of preference and can be seen as instances of valued CSP [2]. A valued CSP is simply a CSP with constraints or constraint tuples having some associated values. Algorithms for solving valued CSPs usually define two additional operators: constraint *combination* and constraint *projection* (or *marginalization*). In the following, we present brief formal definitions of these operators in terms of semiring-based CSP, and interested readers should see [2] for further details. On the other hand, we note that the semiring-based CSP formalism is used here for completeness and due to its convenient descriptions. Therefore, readers may also skip forward to the example shown in figure 2.5 and get an understanding of constraint projection and combination by simply following the example.

Definition (structure) A *semiring* is a tuple $\langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that $\mathcal{A}$ is a semiring set; $\mathbf{0}, \mathbf{1} \in \mathcal{A}$; $+$ is commutative, associative and $\mathbf{0}$ is its unit element (i.e., $a + \mathbf{0} = a = \mathbf{0} + a$); $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element (i.e., $a \times \mathbf{0} = \mathbf{0} = \mathbf{0} \times a$). A *c-semiring* (constraint-based semiring) is a semiring $\langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that $+$ is idempotent (i.e., $\forall a \in \mathcal{A} : a + a = a$) with $\mathbf{1}$ as its absorbing element and $\times$ is commutative.

Definition A *constraint system* is a tuple $CS = (S, D, V)$ where S is a c-semiring, D is a finite set (the domain of the variables) and V is an ordered set of variables.

Definition (constraint) Given a semiring $S = \langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and a constraint system $CS = (S, D, V)$, a *constraint* is a pair (def, con) where $con \subseteq V$ and $def : D^{|con|} \to \mathcal{A}$.

Therefore, a constraint specifies a set of variables (the ones in con), and assigns to each tuple of values over these variables an element of the semiring.

Definition (tuple projection) Given two sets of variables $X = \{v'_1, ..., v'_k\}$ and $Y = \{v_1, ..., v_l\}$ such that $X \subseteq Y \subseteq V$ holds, and any l-tuple $(d_1, ..., d_l)$ of values for variables from Y, the *tuple projection* of $(d_1, ..., d_l)$ from Y to X written $(d_1, ..., d_l) \downarrow^Y_X$, is defined as the tuple $(d'_1, ..., d'_k)$ with $d'_i = d_j$ if $v'_i = v_j$.

Example Consider the tuple $(1, 2, 3, 4)$ corresponding to variables (a, b, c, d), then $(1, 2, 3, 4) \downarrow^{(a,b,c,d)}_{(c,a)} = (3, 1)$.

Definition (combination) Given $c_1 = (def_1, con_1)$ and $c_2 = (def_2, con_2)$, their combination $c_1 \otimes c_2$ is the constraint (def, con) defined by $con = con_1 \cup con_2$ and $def(t) = def_1(t \downarrow^{con}_{con_1})$.

Hence, combining two constraints results in a new constraint that involves all the variables of the original ones, associating to each value tuple over such variables a certain semiring element. This semiring element is obtained by multiplying the elements associated by the original constraints to the appropriate sub-tuples.

Definition (projection) Given constraint $c = \langle def, con\rangle$ and a subset I of V, the projection of c over I, written $c \Downarrow_I$ is the constraint $\langle def', con'\rangle$ where $con' = con \cap I$ and $def'(t') = \Sigma_{t|t\downarrow^{con}_{I\cap con}=t'} def(t)$.

In other words, projecting eliminates some variables, and associates each tuple over the remaining variables a semiring element which is the sum of the elements associated with the original constraint.

The combination and projection operations has also been used for inferring knowledge in belief networks and probability models [15]. If we think of constraint as representing certain knowledge, constraint projection or marginalization corresponds to coarsening of the knowledge, and combination corresponds to aggregation of knowledge. In order to illustrate these operations, let us consider the weighted CSP example shown in figure 2.5.

Example Weighted CSP is represented by the semiring $S = \langle \mathcal{A}, +, \times, 0, 1\rangle = \langle \mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0\rangle$. Here we assume that the weights represent cost or penalties associated with the tuples and the objective is to minimize this quantity. Given the constraint tuples weights as shown in the figure and assuming that we are currently deciding on the value of x_2, we obtain the following:

Projection of c_{12} wrt $x_2 = 1$: $c_{12} \Downarrow_{x_2} (1) = min(9, 5) = 5$

Projection of c_{23} wrt $x_2 = 1$: $c_{23} \Downarrow_{x_2} (1) = min(8, 7, 6) = 6$

Combination of c_{12} and c_{23} when $x_2 = 1$: $c_{12} \otimes c_{23}(1) = 5 + 6 = 11$

Similarly, combination of c_{12} and c_{23} when $x_2 = 2$: $c_{12} \otimes c_{23}(2) = 3 + 3 = 6$

Thus, the algorithm for solving weighted CSP will prefer the value 2 (since we are minimizing the weights) for the variable x_2.

Given a valued CSP or constraint with preferences, a typical algorithm uses the constraint projection and combination to utilize the preference information in order to guide the search for solutions. As shown in the example, these operations can be used as a heuristic for selecting which value is the most promising for a particular variable. There are, however, different projection and combination operations depending on types of CSP to be solved. Since the semiring-based

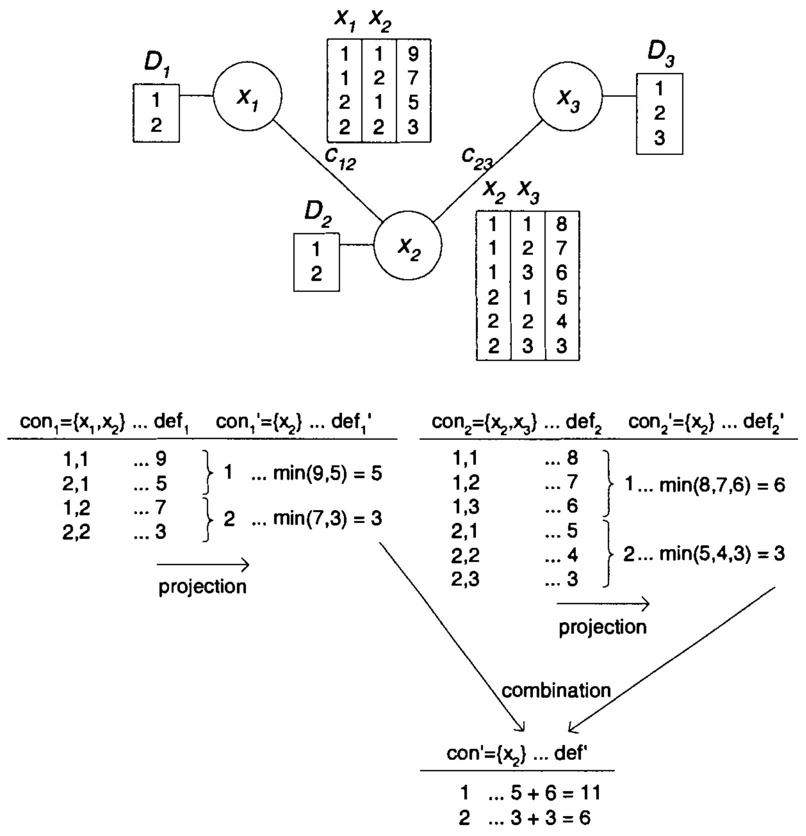


Figure 2.5: Weighted CSP example

framework provides a convenient description of different classes of CSP with preferences, figure 2.6 lists the different operations used in terms of the semiring structure.

Framework	$\mathcal{A}$	$+$	$\times$	$\mathbf{0}$	$\mathbf{1}$
CSP	$\{0,1\}$	$\vee$	$\wedge$	0	1
Weighted CSP	$\mathbb{N} \cup \{+\infty\}$	min	$+$	$+\infty$	0
Probabilistic CSP	$\langle 0,1 \rangle$	max	$\times$	0	1
Fuzzy CSP	$\langle 0,1 \rangle$	max	min	0	1

Figure 2.6: Semiring representation of classes of CSP

2.4 Bargaining Problem

Imagine that there are two agents sharing a variable. Each agent has its own valued CSP that it is trying to solve or optimize and both have different preferences as to which value the variable should have. This scenario of solving valued CSP within a MAS can be seen as a bargaining problem, which can be stated as follows [26]:

> "Two individuals have before them several possible contractual agreements. Both have interests in reaching agreement but their interests are not entirely identical. What will be the agreed contract, assuming that both parties behave rationally?"

The bargaining problem has been studied extensively in the field of Game Theory[2] [7]. Traditionally, game theory can be divided into two branches: non-cooperative and cooperative. Non-cooperative game theory is concerned with specific games with well defined rules and strategies, and with finding rational outcome using the notion of equilibrium strategies. Examples of such strategies are "dominant" strategy, where the outcome is optimal regardless of the strategies of

[2]Game theory is the study of how people interact and make decisions. It applies mathematical models to analyze problems that feature "strategic interaction" between individuals or "players."

other players, and "Nash" equilibrium[3], where no players can benefit by unilaterally changing its strategy.

Cooperative game theory, on the other hand, abstracts away from specific rules of a game, and is concerned with finding a solution given a set of possible outcomes. For bargaining problem, the solution is typically given in terms of utilities. In case of two-player games, the problem becomes finding the outcome given the set of all possible utility pairs or bargaining set. There are many "solution concepts," or functions that map a bargaining problem to a single outcome, and they are usually valid only for a certain subset of all possible bargaining problems. The Nash bargaining solution [20], for instance, only applies to convex and compact bargaining sets. Other solution concepts include *utilitarian* or maximizing the sum of utilities, and *egalitarian* or maximizing the minimum utility. Note that utilitarian policy corresponds to finding solutions that maximize the sum of weights in weighted CSPs.

Depending on the underlying structure of the MAS (i.e., non-cooperative or cooperative), one may apply the results from either branch of game theory. Game theoretic tools have been seen as a particularly suitable match for MAS since computer agents make idealized rational players (whereas humans do not). Besides bargaining and negotiating, rational game theoretic agents in a MAS can also be designed to have the ability to vote, to be involved in auctions, to form coalitions, etc [29]. The primary focus of this thesis is in cooperative MAS and it investigates the use of a negotiation strategy inspired by the Nash bargaining solution for solving multi agent constraint optimization problem. This is similar to Rosenschein and Zlotkin [25] *Product Maximizing Mechanism*, where the strategy is in equilibrium with itself and it is efficient, that is, the deal is the best possible.

Nash Bargaining Solution

In solving the bargaining problem, Nash [20] used an axiomatic approach to obtain a unique solution. Given a Nash bargaining problem (i.e., a compact and convex bargaining set[4]), he proposed four properties, now known as the "Nash axioms"

[3]John F. Nash, Jr. was among the recipients of 1994 Nobel Prize in Economic Sciences for this very contribution.

[4]This assumption is required otherwise the method will not yield a unique solution.

[20][7]:

1. Independence of linear utility transformation, that is the final outcome should not depend on how the player's utility scale is calibrated. So when a function models a player's preference, any strictly increasing affine transformation[5] of that function represents the same preference and therefore will yield the same outcome.

2. The agreement is pareto efficient, that is no player can gain without causing a loss to the other. Figure 2.7 illustrates this concept. The first part shows the utility values of two players given some agreement points, and the second shows their utility imputation (the plotting of utility vs. utility) for agreements 5 to 20. The northeast side represents the pareto frontier and any agreement on this frontier is pareto efficient.

3. Independence of irrelevant alternatives. Given that $c(S)$ represents the solution point for the bargaining set S, if the set T contains the set S and $c(T)$ is in S, then $c(T) = c(S)$.

4. In symmetric situations, both players will get the same payoff. In other words, if the bargaining set S is symmetric (the graph becomes symmetrical with respect to the line $u_1 = u_2$), then $c(S)$ is point of the form (a, a).

Nash proved that the only solution that satisfies these four properties is characterized by $c = (x_1, x_2)$ which maximizes the so-called Nash product $(x_1 - d_1)(x_2 - d_2)$, where d_1 and d_2 represent the disagreement payoffs or the players' utility outcomes when no agreement is reached. Figure 2.7 also illustrates the construction of the Nash bargaining solution for a symmetric two-player bargaining problem.

The Nash bargaining solution has several attractive properties that are desirable when solving a multi agent COP. It yields the same solution regardless of the agent's preference scale, which can be different as it may designed and owned by different organization. Utilitarian and egalitarian, in contrast, will require a comparable preference scale. It is pareto efficient, thus the solution does not waste

[5]From utility theory, $U(W)$ and $a + bU(W)$ with $b > 0$ describe the same preference.

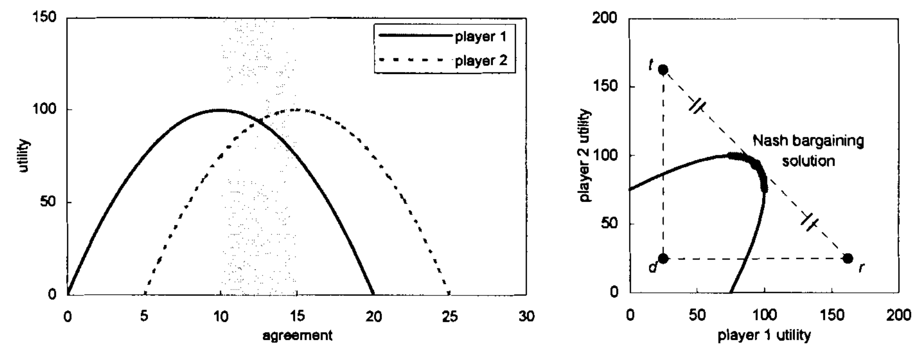


Figure 2.7: Pareto efficiency and the Nash bargaining solution

Case 1: Comparable utilities

agreement	pl utility	p2 utility	Nash	utilitarian
A	2	9	18	**11**
B	5	5	**25**	10

Case 2: Incomparable utilities

agreement	pl utility	p2 utility	Nash	utilitarian
A	2	90	180	-
B	5	50	**250**	-

Figure 2.8: Nash vs. utilitarian

any utility. Furthermore, it constitutes a "fair bargain," unlike utilitarian, for example, where an agent has to agree with smaller payoff because it will benefit the other agent a lot more, that is it should sacrifice for the greater good. Figure 2.8 shows an example that illustrates the advantages of Nash bargaining solution when compared to utilitarian solution. Shown in bold is the accepted agreement and it is assumed that the disagreement payoff is zero.

2.5 Sports League Scheduling

Sports league scheduling has been extensively studied within the CSP research [10][21][23][9]. There are many variations of this problem and many researchers

	rd 1	rd 2	rd 3
t1	2(home)	3(away)	4(home)
t2	1(away)	4(home)	3(away)
t3	4(away)	1(home)	2(home)
t4	3(home)	2(away)	1(away)

Figure 2.9: Example of a round robin schedule

have tackled these with different approaches. One of the most common variations is minimizing the number of *breaks* in a round robin schedule. Figure 2.9 illustrates a typical round robin schedule for 4 teams. A break is defined as a consecutive home-home or away-away games in a particular schedule. Henz [10] used a constraint programming approach, Nemhauser and Trick [21] used combination of both integer and constraint programming approach, and Régin [23] developed filtering algorithms to efficiently prune or reduce the search space. While Henz used constructive search method, Hamiez and Kao [9] used local search with Tabu list. Other common objectives include minimizing the total distance traveled [4] and minimizing the *carry-over effects* [27]. Carry-over effects occur when a match between two teams has an impact on their performances in the next round. An ideal schedule would then only contain a single occurrence of any sequence of two teams, leading to a balanced schedule with respect to carry-over effects. For example, a team may first play against team a then team b. If there are many of such a,b sequence and if team a is a very strong team, team b may receive an unfair advantage because of the resulting low morale.

Typical sports league tournament consists of n teams playing each other in a round robin fashion. When a team plays only once against every other teams, and all the matches has to be played in $n - 1$ rounds (assuming that there are $n/2$ matches in each round), it is called dense single round robin (DSRR) tournament [11]. A dense double round robin (DDRR) is one where each team plays the others twice, usually once at home and once away, within $2(n - 1)$ rounds[6]. Further additional constraints can be imposed such as disallowing a certain match on a

[6]The solution for DDRR can also be obtained by simply repeating a DSRR schedule and swapping the home-away teams.

certain round, considering the availability of stadiums, and so on. Note that we can assume that n is even without loss of generality. When there are odd numbers of teams, we can introduce an additional dummy team. A game played against this dummy team is then considered as a *bye*.

The search for a good schedule typically entails three steps [21]. The first is to find a home-away pattern set. This is a set of n strings of length $n - 1$ (corresponding to the number of rounds in the schedule) which represents the home-away sequence for the teams. For instance, for a four-team round robin with teams $\{a, b, c, d\}$, one feasible pattern set is:

```
1: H A H
2: A H A
3: A H H
4: H A A
```

The next step is to assign games consistent with the pattern set (so if i plays j in a round, then either i is home and j is away, or the reverse). For the above, one possible timetable ("+" denotes at home and "-" denotes away) is:

```
1: +2 -3 +4
2: -1 +4 -3
3: -4 +1 +2
4: +3 -2 -1
```

The third step is to assign teams to the pattern set or timetable, based on, say, their preferences for being home at certain rounds. For example, the teams $\{a, b, c, d\}$ is assigned to 3,1,2,4 respectively, resulting in the following schedule:

```
b: +c -a +d
c: -b +d -a
a: -d +b +c
d: +a -c -b
```

Alternative approaches to sports scheduling differ in the order in which these subproblems are solved and the method employed to solve each of the subproblems. Russel and Leung [28] used combinatorial design theory to obtain the timetables and assign teams in step 3 using enumeration. Nemhauser and Trick [21] solved the subproblems in the above order and used integer programming for steps 1 and 2, and complete enumeration for step 3. Henz [10] improved on this using constraint programming for all three steps, and achieved a significant performance advantage on the last step.

Round robin tournaments can be viewed as a graph theory problem [33]. We first present some basic terminologies. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A *complete* graph is one where there exists an edge from any vertex to any other. A *matching* on a graph is a set of edges such that no two of them share a vertex in common. The largest possible matching on a graph with n vertices consists of $n/2$ edges, and such a matching is called a *perfect matching.* Given a complete graph K_n (n even), a *one-factor* is a perfect matching of K_n. A *one-factorization* is a set of one-factors which are pairwise edge-disjoint and whose union is the set of all edges of the graph. The round robin tournament can then be modeled as follows. Each team is represented as a vertex and the match between two teams as an edge of the graph. Therefore each one-factor corresponds to matches in one round and finding a schedule for unconstrained DSRR (without any side constraints) for n teams is equivalent to finding a one-factorization of K_n. Figure 2.10 shows the previous typical schedule for 4 teams and its corresponding one-factorization.

	rd 1	rd 2	rd 3
t1	2	3	4
t2	1	4	3
t3	4	1	2
t4	3	2	1

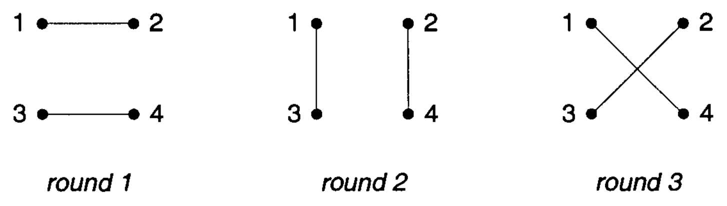


Figure 2.10: DSRR schedule and its one-factorization

Two most important constraints for round robin scheduling are the *all-different* and one-factor constraints. The all-different expresses that a row in the schedule contains every team only once and the one-factor groups the teams in a column into matches. Hence a solution that satisfy these constraints is a valid schedule for unconstrained DSRR. Using a graph theoretic approach, Régin developed arc-consistency propagation algorithms for both constraints [23][24]. By combining a constructive search method with Régin's consistency algorithms, Henz showed that a significant speedup, up to one order of magnitude, can be achieved when compared to simply encoding the problem with equality/non-equality constraints [11]. In their approach, Hamiez and Kao [9] added another constraint to DSRR to only allow a maximum of two matches in a same period for each team. That is, for each round there are $n/2$ periods and each match is assigned to a particular period. They used a different formulation of the sport scheduling problem and did not consider home or away games. An example of their formulation is shown in figure 2.11. Using Tabu search, they were able to solve instances of up to 40 teams, where constructive approaches were typically limited to 24 teams. However, they also found that the computing times required were much greater than those obtained using the most efficient algorithms.

	rd 1	rd 2	rd 3	rd 4	rd 5
period 1	1,2	2,6	3,4	5,6	1,5
period 2	4,6	1,3	2,5	1,4	3,6
period 3	3,5	4,5	1,6	2,3	2,4

Figure 2.11: DSRR schedule with period assigned for six teams

2.6 Random CSP

A class of problem that is often used to evaluate CSP algorithms is the randomly generated binary CSP. A set of parameters characterize a random CSP:

- n the number of variables,
- m the domain size of the variables (typically all variables will have the same

domain values and domain size),

- p_1 or d the density of the constraint graph which is the probability or proportion that a constraint exist between any two variables (value of 0 means that no constraint exist in the problem, value of 1 means the constraint graph is a complete graph), and

- p_2 or t the tightness of a constraint which is the probability or proportion that a tuple in the $m \times m$ relation is disallowed (e.g., value of 0.33 means that about one-third of the possible combination of values is disallowed, and two-third is allowed).

2.6.1 Generation Models

Most experimental and theoretical studies use one of four simple models of random problems. In each of these models, a constraint graph G is generated, and then for each edge in this graph, pairs of incompatible values are chosen. The models differ in how the graph is generated and how the incompatible values are chosen. The four models are [6]:

Model A: We independently select each one of the $n(n-1)/2$ possible edges in G with probability p_1, and for each selected edge we pick each one of the m^2 possible pairs of values, independently with probability p_2, as being incompatible.

Model B: We randomly select exactly $p_1 n(n-1)/2$ edges for G, and for each selected edge we randomly pick exactly $p_2 m^2$ pairs of values as incompatible.

Model C: We select each one of the $n(n-1)/2$ possible edges in G independently with probability p_1, and for each selected edge we randomly pick exactly $p_2 m^2$ pairs of values as incompatible.

Model D: We randomly select exactly $p_1 n(n-1)/2$ edges for G, and for each selected edge we pick each one of the m^2 possible pairs of values, independently with probability p_2, as being incompatible.

Achlioptas et al. [1] identified a deficiency with all four random models. They proved that if $p_2 \geq 1/m$ then, as $n \to \infty$, there almost surely exists a *flawed* variable. A flawed variable is one where each value in its domain is *flawed*, that is there exists an adjacent variable in the constraint graph that cannot be assigned a value without violating the constraint between the two variables. Thus problem with a flawed variable cannot have a solution. They argue that therefore these model are asymptotically uninteresting except, perhaps, for a small space of their parameter space (i.e., when $p_2 < 1/m$). They proposed an alternative model which has better asymptotic properties. This model does not separate the generation of the constraint graph from the selection of nogoods.

Model E: We select uniformly, independently and with repetitions, $pm^2n(n-1)/2$ nogoods out of the $m^2n(n-1)/2$ possible.

However, this model is not without its shortcoming. In particular, it cannot control the resulting constraint graph and for small values of p, it generates a complete constraint graph. Therefore, it is a much less flexible model compared to models A to D. Gent et al [6] proposed a new way of generating conflict matrices which are *flawless*, since the resulting problems are guaranteed not to be trivially insoluble. They showed how their method can be adapted to the standard models and proved its desirable asymptotic properties. For models B and C, given a pair of variables between which a constraint is to be constructed, a random permutation π of $1, 2, ..., m$ is chosen. The set of goods based on this permutation is simply $\{(1, \pi(1)), (2, \pi(2)), ..., (m, \pi(m))\}$, and a conflict matrix that contains these goods cannot give a flawed value. These goods are removed from the set of all possible conflicts and p_2m^2 elements are randomly chosen from the remainder. For models A and D the process is similar, except that having removed the set of goods, p_2 is increased to $mp_2/(m-1)$ before selecting conflicts.

2.6.2 Problem Hardness

As noted by Cheeseman, Kanefsky and Taylor [3], the hardest instances of many NP-complete problems often occur around the phase transition area. This is the region where the problem changes from being under-constrained to being over-constrained. Typically, low p_2 values will produce under-constrained problems and high p_2 values will produce over-constrained problems. Both of these are easy to solve because the former will have many solutions and the latter will have no solution that can be easily proven using propagation or pruning techniques. Thus, it is the value in between that will generate hard problems and is of most interest for evaluating CSP algorithms. For random CSP, Smith [31] identified this as the *mushy* region where the expected number of solution is 1. Equation 2.1 gives the critical value for p_2.

$$\hat{p}_{2crit} = 1 - m^{-2/((n-1)p_1)} \tag{2.1}$$

Similar results were obtained by Gent et al [6] using κ values which represent the *constrainedness* of combinatorial problems. $\kappa = 1$ corresponds to random problems where the expected number of solution is 1, and the formula for computing κ is given in equation 2.2.

$$\kappa = \frac{n-1}{2} p_1 \log_m(\frac{1}{1-p_2}) \tag{2.2}$$

In this thesis, we have used randomly generated CSP as one of the problems to evaluate the performance of our approach. For our experiment, we generated flawless random CSP according to model B, and selected the parameters such that κ is close to 1. This ensures that experimental results obtained are valid and our technique is both scalable to large problems and applicable to hard problems.

Chapter 3

Preference and Feasibility Marginalization

Typically, a constraint optimization problem is solved using a branch-and-bound algorithm. Such algorithm behaves like a backtracking search and explores the complete solution space. While this approach guarantees the optimality of the solution found, it may require exponential time which can be impractical for large problems. Within a multi agent setting, an approximate solution can be found using a local negotiation strategy between agents. This thesis investigates how good the approximation is when these local negotiations utilize marginalization that considers both preference and feasibility.

We first present the multi agent model with its corresponding valued CSP. We then describe the negotiation protocol for solving this distributed constraint optimization problem. We will show how the current marginalization methods are inadequate and outline the properties that a marginalization function should have. In order to better illustrate our approach, we define a multi agent sport scheduling problem and use it as an example application. Instead of working with just agents, variables and constraints, we can also think in terms of teams, games and schedules. Further, show how our technique can be applied to solving random binary CSPs.

3.1 Multi Agent Model

Agents in a distributed CSP share constraints and variables. Each agent can be as simple as representing a single variable or it can represent a complex local CSP with many local variables and constraints. A shared constraint is the inter-agent constraint that the agent must satisfy while trying to determine the values of its local variables. A shared variable between several agents can be viewed as if each agent has a different variable, and there exist constraints that these variables must have the same value. As in Yokoo's multi agent systems [36], we assumed the following model.

- Agents communicate by sending messages.
- The communication network is reliable, i.e., message sent is guaranteed to reach its destination.
- For the transmission between any pair of agents, messages are received in the order in which they were sent.
- Each agent knows all the constraints (local and shared) relevant to its variables.
- Each agent maintains an *agent_view* which consists of the current states of other agents that are involved with its shared constraints. This ensures an agent is aware whether its shared constraints are currently satisfied or not.

We also assumed that inter-agent constraints are equality constraints, that is, agents only share variables. The reason for this is because we intend to model the problem as a bargaining problem in which agents negotiate to agree on the value of the shared variable. In most cases, we can equivalently represent any inter-agent constraint as an equality constraint with a new intra-agent constraint added to one of the agents. In other words, moving the inter-agent constraint into either agent's local CSP and creating an equality constraint that connects the variables. Figure 3.1 shows an example of such a transformation. Furthermore, we have assumed the following in our MAS model.

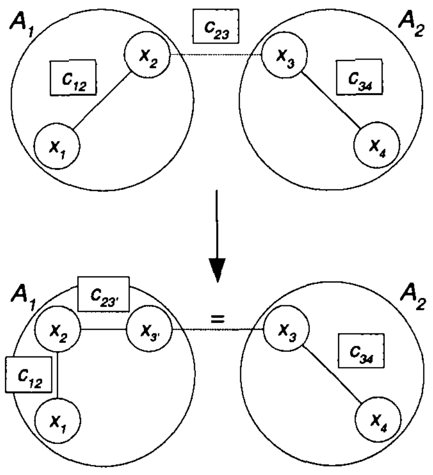


Figure 3.1: Representing an inter-agent constraint as an equality constraint

- Each constraint tuple in the local problem has some associated preferences, hence we are solving an optimization problem.
- All agents are equal in terms of rank, therefore local search is more appropriate since a constructive approach requires ordering of agents and may lead to solutions that favor higher ordered agents.
- An agent can only be involved in one negotiation at a time even if it has many shared variables. This ensures that it is making an up-to-date valuation of the bargaining set and avoids oscillation.

The objective is therefore to find a solution which maximizes the agents preferences and one that is fair to all the agents. A typical MAS with its valued CSP is given in figure 3.2 and the local search agent procedure is presented in figure 3.3.

A local search approach starts with some initial (random) assignment to all of the variables and then proceeds to improve the assignment. Note that as shown in line 03, the termination condition of the agent procedure is when all the local constraints are satisfied. We assume that the agent is repairing the assignment

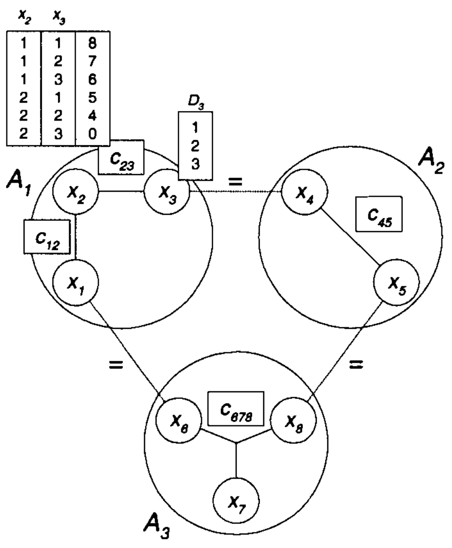


Figure 3.2: An example of MAS with valued CSP

```
proc agent_run()
    initialize()
    while (local constraints not satisfied)
        select a variable $x$ to be repaired
        if ($x$ is a local variable)
            repair($x$)
        else
            if (set_up_negotiation($x$) = success)
                marginalize($x$)
                negotiate($x$)
            end if
        end if
        if (in local maxima) { random_restart() }
    end while
return
```

Figure 3.3: Agent procedure

according to its preference and through negotiations with other agents, and therefore the first feasible solution obtained is not only considered to have maximized its preference, but also the preferences of others. First, a candidate variable to be repaired is chosen as shown in line 04. If the variable is internal to an agent, that agent can freely repair it according to its preference. However, if the variable is shared among several agents, a negotiation is required to ensure that all the involved parties are satisfied. We check for this condition in line 05. If it is a shared variable, a negotiation session has to be set up. This involves exchanging messages with other agents, and it is possible that some other agents are already involved in another negotiation session[1]. If the session is successfully set up, the agent marginalizes its preferences and negotiates for the value of the selected variable (lines 09-10). This process of negotiation and re-negotiation is repeated until all agents are satisfied. As with any local search algorithm, our search can be trapped in local maxima. While there are numerous ways to escape such points (such as using a nogood cache or tabu list), the algorithm simply restarts with a new random assignment, as shown in line 13. Figure 3.4 shows an example of the algorithm execution. Here, both agents A_1 and A_3 choose a shared variable to be repaired. They both attempt to set up a negotiation session with A_2. As A_2 can only negotiate for one shared variable at a time, it accepts A_1 and rejects A_3. Both A_1 and A_2 marginalize their preference to the shared variable and then exchange their valuations. The shared variable is then repaired to the agreed value.

This procedure is similar to a typical local search algorithm, except that we need to further define the marginalization and negotiation functions. Note that within the semiring-based CSP, these functions correspond to projection and combination operations respectively. It should be re-emphasized that constraint projection or marginalization is only an approximation of the actual preferences. Recall that from the above example, agent A_1 only has the exact preference of the constraint c_{23} (i.e., preference values for the tuples x_2, x_3). The marginalization essentially summarizes this information into a single valuation vector in terms of the domain of x_3. The agents subsequently negotiate and reach an agreement based on their approximated preferences of the shared variable. Therefore, the

[1]In our experiment, we have simplified this process by using a system agent that determines which negotiation session is to be performed.

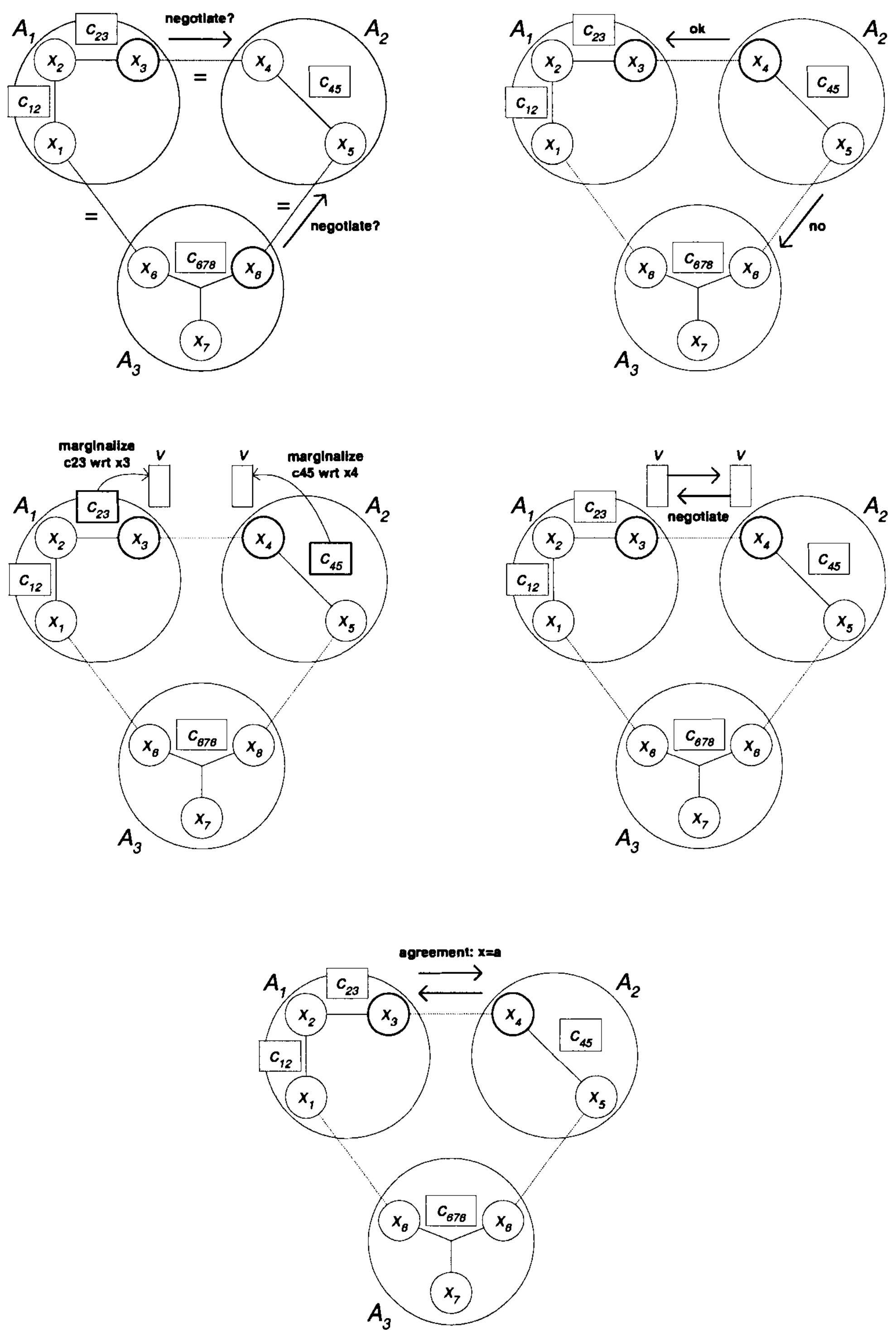


Figure 3.4: Example of algorithm execution

performance of this algorithm will depend on these two functions.

We know that using Nash bargaining solution as the negotiation strategy will result in a fair and efficient outcome. It also allows for incomparable agent utility values. Hence, the Nash bargaining solution can be fruitfully applied to our problem. The challenge then is to define a marginalization function that will lead such negotiations to not only any solution, but good solutions. As mentioned previously, there are two issues involved in solving a constraint optimization problem: feasibility and preference. A marginalization that emphasizes on feasibility of the problem will not be able to explore infeasible regions which may lead to a better solution. On the other hand, a marginalization that emphasizes on preference may never find a solution. We argue that therefore a marginalization function must consider both feasibility and preference simultaneously in order to reach good solutions. Unfortunately, existing marginalization functions only consider the feasible tuples of the constraint when producing the valuation. Therefore, they are inadequate for this purpose since they can only provide preference values for feasible domain elements. Furthermore, in local search methods, all the variables are assigned some values and the marginalization will simply return the preference value of this fully specified tuple. That is, no real marginalization is performed (unlike one described in figure 2.5 on page 19). The reason for this behavior is because marginalization has only been used in constructive search, where at least some variables are unbound[2]. Figure 3.5 illustrates this behavior by showing the result of applying different common marginalization functions to a constraint. We used the previous example, assuming that agent A_1 is marginalizing the constraint c_{23} with respect to x_3 (i.e., $c_{23} \Downarrow_{x_3}$). The first part shows the result for constructive search assuming x_2 is unbound (not assigned any value), and the second part shows the result for local search assuming the current value of x_2 is 2. An infeasible tuple is denoted with a dash.

[2]It should be noted that we are not interested in static marginalization, where the marginalization does not consider the current states of other variables. Such marginalization may provide static indication of preferences for each domain elements, but since it ignores the current states of other variables, it cannot actively guide the search.

x_2 is unbound

x_2	x_3	u
1	1	8
1	2	7
1	3	6
2	1	5
2	2	4
2	3	-

$\Rightarrow$

D_{x_3}	$f = min$	$f = max$	$f = avg$
1	$min(8,5) = 5$	$max(8,5) = 8$	$avg(8,5) = 6.5$
2	$min(7,4) = 4$	$max(7,4) = 7$	$avg(7,4) = 5.5$
3	$min(6) = 6$	$max(6) = 6$	$avg(6) = 6$

$x_2 = 2$

x_2	x_3	u
~~1~~	~~1~~	~~8~~
~~1~~	~~2~~	~~7~~
~~1~~	~~3~~	~~6~~
2	1	5
2	2	4
2	3	-

$\Rightarrow$

D_{x_3}	$f = min$	$f = max$	$f = avg$
1	$min(5) = 5$	$max(5) = 5$	$avg(5) = 5$
2	$min(4) = 4$	$max(4) = 4$	$avg(4) = 4$
3	$min(-) = -$	$max(-) = -$	$avg(-) = -$

Figure 3.5: Constraint marginalization example

Therefore, the traditional (constructive search) marginalization when applied to local search can only distinguish between feasible states and the valuation provided is the tuple preference. In addition, if a non-binary constraint exists in the problem and the current states of some variables already violate the constraint, these marginalization are helpless. We will describe and illustrate this shortcoming using a sport scheduling example prior to proposing a solution. In the following section, we first define the multi agent sport scheduling problem which we will use throughout this thesis.

3.2 Multi Agent Sport Scheduling

Previous work on solving sport scheduling problems has focused on centralized approach with global objective function. In transforming the problem into a distributed CSP with preferences, we have adopted a slightly different formulation. The following characterizes our multi agent sport scheduling problem (MASSP):

- The tournament is a dense single round robin.
- Each team is represented by an agent.

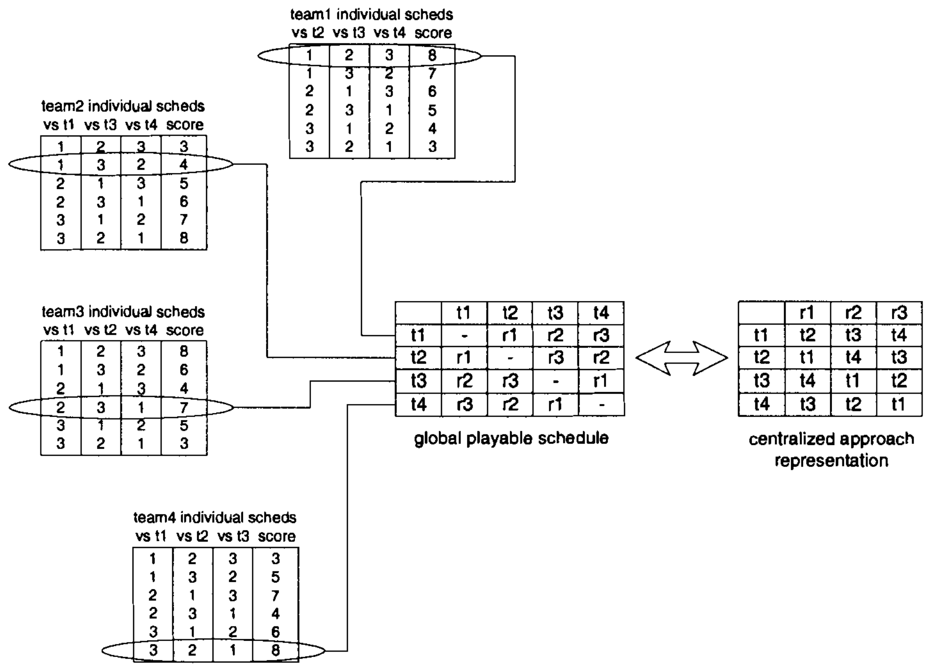
vs t2	vs t3	vs t4	score
rd 1	rd 2	rd 3	8
rd 1	rd 3	rd 2	7
rd 2	rd 1	rd 3	6
rd 2	rd 3	rd 1	5
rd 3	rd 1	rd 2	4
rd 3	rd 2	rd 1	3

Figure 3.6: The set of individual schedules for team 1 for $n = 4$

- Every agent/team possesses a set of individual schedules I. An individual schedule consists of rounds in which the games with other teams are to be played and an associated user-assigned preference. Figure 3.6 shows an example of a set of individual schedules.
- The objective is to find a compatible global schedule that maximizes the combination of individual preferences with respect to the Nash bargaining solution.

For example, the preferences can be the travelling cost and we want to minimize the total cost. However, the preferences can be much more complex and are not necessarily the same for each team. One team may want to schedule games to accommodate its fans, others may want to play easier opponents earlier and tougher ones later, and so on. In MASSP, we assumed that a user will assign a more preferred schedule with a larger preference value, and will assign infeasible schedules with values of zeros.

It is important to realize that the team set of individual schedules is different from the global playable schedule. The set of individual schedules lists the different possible schedules for one particular team and their corresponding values to the team. The global playable schedule consists of a compatible individual schedule from each involved team. In order to model the tournament as a multi agent problem, we have used a representation that is different from that used in a centralized approach. This representation facilitates the negotiation model between teams, that is, it allows for agents to negotiate the round (or week) in which their games should be played. Figure 3.7 presents an example of our representation and contrasts it with the equivalent centralized representation.



team1 individual scheds

vs t2	vs t3	vs t4	score
1	2	3	8
1	3	2	7
2	1	3	6
2	3	1	5
3	1	2	4
3	2	1	3

team2 individual scheds

vs t1	vs t3	vs t4	score
1	2	3	3
1	3	2	4
2	1	3	5
2	3	1	6
3	1	2	7
3	2	1	8

team3 individual scheds

vs t1	vs t2	vs t4	score
1	2	3	8
1	3	2	6
2	1	3	4
2	3	1	7
3	1	2	5
3	2	1	3

team4 individual scheds

vs t1	vs t2	vs t3	score
1	2	3	3
1	3	2	5
2	1	3	7
2	3	1	4
3	1	2	6
3	2	1	8

	t1	t2	t3	t4
t1	-	r1	r2	r3
t2	r1	-	r3	r2
t3	r2	r3	-	r1
t4	r3	r2	r1	-

global playable schedule

	r1	r2	r3
t1	t2	t3	t4
t2	t1	t4	t3
t3	t4	t1	t2
t4	t3	t2	t1

centralized approach representation

Figure 3.7: MASSP problem representation

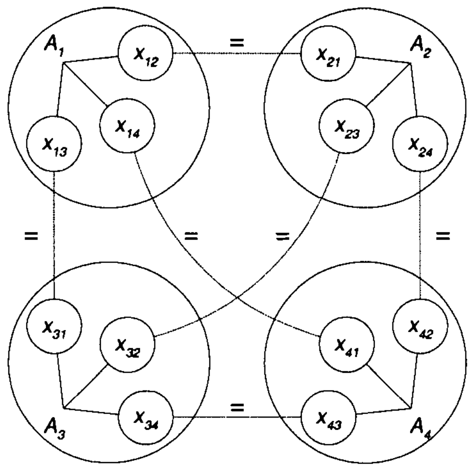


Figure 3.8: Multi agent network for 4 teams

We can now represent the MASSP in terms of our multi agent model. Figure 3.8 illustrates the multi agent network for 4 teams. Shown in dashed lines are the communication lines between agents which also denote the shared variables. The variable x_{ij} represents the games between team i and team j, and its domain is the playable rounds. Hence the set of individual schedules is simply a non-binary constraint (in this case an all-different constraint) with preferences and each agent can be seen as one valued constraint[3]. Each row or individual schedule is just a tuple of this constraint. We can view the set of individual schedules as an enumeration of the solutions to the all-different constraint with some associated preferences.

In considering the MASSP as a bargaining problem, an ideal solution would be to compute the bargaining set for the n teams involved and find the maximum Nash product. Since the preference or utility value is dependent on the complete individual schedule (a row in the table is one agreement), we have a multi-item bargaining involving all the teams. We will need to find or enumerate the compatible schedules for all teams in order for them to evaluate their payoffs. In other

[3]In terms of the equivalent centralized problem, for n teams, there will be $\frac{n(n-1)}{2}$ variables, each with domain size $n-1$, resulting in search space of $O(n^{n^2})$. For 8 teams it is 7^{28} or 4.6×10^{23}.

words, we need to first list all feasible solutions and then take the Nash product to decide which solution is the best. This is equivalent to taking the cross-product of n individual schedules first, which is in the order of $|I|^n$, exponential in size. This is NP-hard so an alternative bargaining model is required.

An obvious approach to approximating the bargaining process is to restrict it to a single item. Therefore the teams bargain for which round a particular game should be played. Since a game only involves two teams, each bargain will have only two participants, and there will be many of such negotiation. But, as alluded to previously, a team typically can only evaluate the utility of a complete schedule, not individual games. For example, it is easy to calculate the travelling cost given the entire tour, but the same is not true if all we know is that the game against team 2 is in round 3. Therefore the alternative is to marginalize the preferences of the complete schedule before the negotiation can take place. However, as we shall see, obtaining a reasonably accurate marginalization is not straightforward[4]. Let us consider an example of MASSP for 4 teams. Consider that we have the following constraint table (or individual schedules) for $t1$:

vs $t2(x_{12})$	vs $t3(x_{13})$	vs $t4(x_{14})$	score
1	2	3	8
1	3	2	7
2	1	3	6
2	3	1	5
3	1	2	4
3	2	1	3

Using this table, say $t1$ wants to negotiate the game between $t2$ and itself (i.e., x_{12}) and currently $x_{13} = 2$ and $x_{14} = 3$, then the projection[5] of $t1$ to x_{12} is the following table[6]:

[4]We can also think of the marginalization of a non-binary constraint as putting a value on an item in a bundle. Even if we know the value of a bundle, it may not be easy to assign values to the individual items (e.g., computing the individual item prices for combinatorial auctions problem).

[5]Recall that with local search, the traditional marginalization or projection simply returns the tuple preference, regardless of the operator.

[6]Value of zero means the domain element is not feasible.

domain	value
1	8
2	0
3	0

Hence x_{12} would prefer to have value of 1. However, say that currently $x_{13} = 2$ and $x_{14} = 2$, then the projection of $t1$ to x_{12} is the following table:

domain	value
1	0
2	0
3	0

Which value should x_{12} take? This behavior is the result of marginalizing a non-binary constraint where the values of the other variables (variables that are not currently considered to be repaired) taken together always are infeasible. That is, we cannot repair x_{12} and get to a feasible state. We are faced with the problem that we are unable to distinguish between infeasible states. Intuitively, since we are working with the all-different constraint, x_{12} should be either 1 or 3, but not 2.

In the bargaining context, the projection represents the utility values for different agreements. In both cases, we have scores of zeros which means that those domain values will not be in the bargaining set. The reason for this is that existing projection operators only consider feasible agreements. In addition, projection operations are usually employed in constructive algorithms where feasibility is always maintained. If both teams only consider feasible agreements, the bargaining set will often be an empty set, and there will be nothing to negotiate. Thus we need an alternative to this projection, one that is able assign preference to all domain values and is more suitable for local search.

3.3 Combined Marginalization

In developing our marginalization strategy, we have identified four desirable properties that such technique should satisfy:

1. Ability to distinguish between feasible states.

2. Ability to distinguish between infeasible states.
3. Ability to choose state with higher preference at the cost of more conflict.
4. Contains the element of both preference and feasibility.

Property 1 is an obvious requirement and property 2 is to handle cases illustrated in the previous example. The reason for property 3 is that since we are performing repeated negotiation, we can insist on what we really prefer and hope that other agents may move to their preferred states and eventually arrive at a good feasible solution. It is somewhat a greedy approach to the optimization problem. This is similar to both Tabu Search and Simulated Annealing and therefore allows for exploration of possibly better solution space. Property 3 also borrows from the breakout concept [19], where the state with more conflicts is chosen, but instead of trying to escape from local maxima, we are trying to get to highly preferred solutions. Property 4 represents a novel method in solving constraint optimization problems. Instead of the typical approach of first finding feasible solutions and then improving it (e.g., branch and bound search), we want to explore the search space while considering both elements of preference and feasibility simultaneously. We called this strategy *combined marginalization*[7]. Note that, while the following sections describe specific functions that possess the above properties, one can employ alternate functions having the same characteristics.

3.4 Valued Conflict Projection

We first present a simplified version of MASSP. Here each agent has one all-different constraint representing its set of individual schedules and we assume

[7]We can think of this technique as combining the solution quality landscape with the feasibility landscape. The solution quality landscape will have many large flat areas with scattered peaks of differing heights. The higher the peak the better the solution. However, it is difficult to navigate through this space since the solution quality may not be available unless the current state is feasible. On the other hand, the feasibility landscape will be more contoured and with equal height highest peaks. These peaks represent the feasible states. It is easier to navigate the feasibility landscape but the quality of solution achieved may not be great. Hence combining the two landscapes produces a new landscape that is more easily navigated and has contours that lead to good solutions.

rd	vs t2 (u_{12})	vs t3 (u_{13})	vs t4 (u_{14})
1	9	3	2
2	2	7	6
3	3	4	9

vs t2	vs t3	vs t4	score
1	2	3	$9+7+9=25$
1	3	2	$9+4+6=19$
2	1	3	$2+3+9=14$
2	3	1	$2+4+2=8$
3	1	2	$3+3+6=12$
3	2	1	$3+7+2=12$

Figure 3.9: Schedule preference from individual games preferences for $t1$

that the preference of the individual schedules is simply a function of the individual games, as shown in equation 3.1.

$$u(x_1 = a_1, ..., x_n = a_n) = \sum_{i=1..n} u_{x_i}(a_i) \tag{3.1}$$

In other words, the user is assigning preference to individual games, instead of to complete schedules, and the utility value of the complete schedule is a function of these preferences. While this is in contrast to what we have emphasized previously, where a team is actually able to accurately evaluate individual games, it provides a convenient starting point for our marginalization. We will remove this restriction in the next section. An example is given in figure 3.9.

Given such individual games preferences, we can extend the min-conflict heuristic to marginalize the all-different constraint using the following:

$$f_{x_i}(a) = u_{x_i}(a) + \sum_{x_j \in C, j \neq i, x_j \neq a} u_{x_j}(b_j) \tag{3.2}$$

Equation 3.2 is simply summing up the individual game preferences when values between the target variable x_i and the other variables x_j are pairwise different (b_j is the current value of x_j). This function essentially penalizes the domain values that will lead to more conflicts, or more accurately, values that will lead to higher degree of violation of the all-different constraint. This function satisfies all 4 desirable properties mentioned above, except the first one since it does not apply to all-different constraint. That is, there can be no more than one feasible state if we only consider the value of one variable. An example using values from figure 3.9 is given to illustrate this marginalization.

> **Example** Let $x_{13} = 1$ and $x_{14} = 1$. The marginalization for x_{12} is
>
> $$f(1) = 9 + 0 + 0 = 9$$
> $$f(2) = 2 + 3 + 2 = 7$$
> $$f(3) = 3 + 3 + 2 = 8$$
>
> Hence the value of 1 will be preferred even at the cost of more conflicts. Furthermore, while all three states are infeasible, we are able to differentiate them.

Equation 3.2 can be seen as an approximation to any possible states, feasible or infeasible. It serves as a greedy indication to the direction of the search and the computational cost of this operation is linear in the number of involved variables. Note that if we have no preferences between individual games (e.g., all $u_{x_i} = 1$), the marginalization simply represents the degree of satisfaction of the all-different constraint. That is, a value that is most pairwise different from the other variables' current values is preferred (i.e., min-conflict behavior).

3.5 Table Based Projection

In this section we generalize our method further. First, we remove the assumption that the individual game preference is available. Second, we use our method on random binary CSPs.

3.5.1 General Non-Binary Constraints

Here we consider the MASSP where each agent possesses complete individual schedules[8] with preferences, such as the one listed in figure 3.10. Although for the MASSP case the constraint is all-different, the following method is applicable

[8]The number of possible individual schedules is the permutation without repeated elements, i.e., $(n-1)!$ for each agent.

to problems where the constraint is expressed as a table with associated preferences. We propose the following marginalization function:

$$f_{x_i}(a) = \sum_{x_j \in C, j \neq i} \sum_{t \in C} u(x_i = a, x_j = b_j) \tag{3.3}$$

Equation 3.3 considers each of the non-target variables separately. $u(x_i = a, x_j = b_j)$ returns the preference value from the table where $x_i = a$ and $x_j = b_j$ ignoring the rest of the other variables. The inner summation[9] gives the indication of support for the current pair, while the outer summation iterates through each involved variables. This equation also satisfies the desirable properties mentioned previously. Intuitively, this marginalization function transforms the single k-ary constraint[10] into $k-1$ binary constraints. Each binary constraint involves the target variable and one other variable, and simply takes the valuation of the k-ary tuple as its own. The result of the marginalization is the sum of these binary valuations. Figure 3.10 presents an example of this marginalization for x_{12} when $x_{13} = 2$ and $x_{14} = 2$. Note that this is the same example given in the beginning of the chapter, in section 3.2. Recall that when using traditional marginalization, we were not able to distinguish between infeasible states. In this example, the value 1 will be preferred although it is infeasible.

From this example, we observe that the marginalization function is not overly greedy. It will only consider values that will not require changing every other existing agreements. In the example, the value $x_{12} = 2$, will not be part of the bargaining set, since in order to achieve a solution with such assignment, both x_{13} and x_{14} have to be changed. While such exclusion will occur less frequently with larger arity constraint, it nevertheless prevents the exploration of certain search space. Figure 3.11 presents another example of the marginalization for x_{12}, but with $x_{13} = 2$ and $x_{14} = 1$. In this case, the infeasible value 1 is preferred, instead of the feasible value 3, since it is deemed more promising.

Note that in both examples, there is only one occurrence for each pair (e.g., $\langle x_{12} = 1, x_{13} = 2\rangle$) because of the size of the teams. For larger sized teams, the inner

[9]This summation is the marginalization operator in the traditional sense, which can also be replaced with max.

[10]A non-binary constraint involving k variables.

vs $t2(x_{12})$	vs $t3(x_{13})$	vs $t4(x_{14})$	score
1	2	3	8
1	3	2	7
2	1	3	6
2	3	1	5
3	1	2	4
3	2	1	3

$x_{13} = 2$ and $x_{14} = 2$

$D_{x_{12}}$	value
1	$u(x_{12} = 1, x_{13} = 2) + u(x_{12} = 1, x_{14} = 2) = 8 + 7 = 15$
2	$u(x_{12} = 2, x_{13} = 2) + u(x_{12} = 2, x_{14} = 2) = 0 + 0 = 0$
3	$u(x_{12} = 3, x_{13} = 2) + u(x_{12} = 3, x_{14} = 2) = 3 + 4 = 7$

Figure 3.10: Table based combined marginalization example

$x_{13} = 2$ and $x_{14} = 1$

$D_{x_{12}}$	value
1	$u(x_{12} = 1, x_{13} = 2) + u(x_{12} = 1, x_{14} = 1) = 8 + 0 = 8$
2	$u(x_{12} = 2, x_{13} = 2) + u(x_{12} = 2, x_{14} = 1) = 0 + 5 = 5$
3	$u(x_{12} = 3, x_{13} = 2) + u(x_{12} = 3, x_{14} = 1) = 3 + 3 = 6$

Figure 3.11: Table based combined marginalization example

summation will provide support indication. For example, consider the schedule for five teams[11] as shown in figure 3.12. The inner summation of the first term will result in $u(1, 2, 3, 4) + u(1, 2, 4, 3)$ for domain element 1. That is, when we only consider $x_{13} = 2$, we see that there are 2 rows that support the domain value 1, and only 1 row that supports the domain value 3. The cost of this marginalization operation is dependent on the size of constraint table and the number of variables involved in the constraint[12]. However, as long as the preferences do not change, the marginalization need only to be calculated once and can be cached.

[11]For simplicity, we have excluded the dummy team required to make the number of teams even.

[12]For an all-different constraint considering all possible permutation, the cost would be $O(n!)$, where n is the number of teams.

vs $t2(x_{12})$	vs $t3(x_{13})$	vs $t4(x_{14})$	vs $t5(x_{15})$	score
1	2	3	4	8
1	2	4	3	7
1	3	2	4	6
1	3	4	2	5
		⋮		
3	2	1	4	9
3	2	4	1	0
		⋮		

$x_{13} = 2$

$D_{x_{12}}$	value
1	$(8+7)+\cdots$
	⋮
3	$(9)+\cdots$
	⋮

Figure 3.12: Table based combined marginalization example

3.5.2 Random Binary CSP

In this thesis we used flawless random binary CSP constructed using the method recommended by Gent et al [6][13]. However, it is necessary to transform the CSP into an optimization problem. We augmented each resulting constraint with a valuation structure, that is when a tuple is allowed, a preference score is assigned. This is similar to the individual schedule discussed in section 3.2, and serves to differentiate which tuples in the constraint are preferred. Evaluation on random CSPs provides an additional validation of the results and further illustrates the generalization of our method.

Here, we are given a table representing allowed and disallowed tuples for each constraint, and for each allowed tuple, there is a preference value assigned to it. If we simply use equation 3.3 for the marginalization, the possible agreement points will be the feasible states differentiated by their preferences. In other words, we are considering feasibility first and then break any ties using the preferences. Note

[13]The author would like to thank Michael Horsch for providing the code for the generation of random CSPs.

that this behavior is due to the fact that we are now dealing with binary constraint instead of non-binary one. Equation 3.3 degrades to the problematic traditional marginalization when applied to binary constraint, since the inner summation represents traditional marginalization and there is only one term for the outer summation. Thus this function will not allow us to explore infeasible states, no matter how attractive they are. We propose the following function to address this issue:

$$f_{x_i}(a) = max(u(x_i = a)) + u(x_i = a, x_j = b_j) \tag{3.4}$$

The first term of equation 3.4 is simply a static value preference and allows for consideration of infeasible states. The second term is the same with equation 3.3, but without the summations since we are considering a binary constraint, and there will be only one tuple with the values $(x_i = a, x_j = b_j)$. This function therefore provides the ability to prefer states with higher preference at the cost of more conflict. Figure 3.13 illustrates the binary constraint marginalization example of x_1 using both equations 3.3 and 3.4. In this example, we let $D_{x_1} = D_{x_2} = \{1, 2, 3\}$, $x_2 = 1$ and we have omitted the disallowed tuples from the constraint table. As we can see, equation 3.4 results in preferring $x_1 = 3$ even if that value is infeasible, due to the high preference value.

x_1	x_2	u
1	1	1
1	2	2
2	1	2
3	2	5

D_{x_1}	value(eqn 3.3)	value(eqn 3.4)
1	$u(x_1 = 1, x_2 = 1) = 1$	$max(u(x_1 = 1)) + u(x_1 = 1, x_2 = 1) = 2 + 1 = 3$
2	$u(x_1 = 2, x_2 = 1) = 2$	$max(u(x_1 = 2)) + u(x_1 = 2, x_2 = 1) = 2 + 2 = 4$
3	$u(x_1 = 3, x_2 = 1) = 0$	$max(u(x_1 = 3)) + u(x_1 = 3, x_2 = 1) = 5 + 0 = 5$

Figure 3.13: Random binary CSP marginalization example

3.6 Refinement

The same desirable properties that can guide the search to good solutions can also doom the search altogether. There will be cases where the preferences are so strong that the marginalization will always choose these high values for some variables even at the cost of more conflicts, and hence never arrive at a solution. In order to avoid such pitfall, we incorporated a preference skewing mechanism. If after a certain number of restarts and no solution is found, the preference values are skewed such that the high preference values do not overwhelm the low preference values. We propose the skewing procedure given in figure 3.14.

```
01 procedure skew_preferences(constraint $c$, max_pref $max$, increment $inc$)
02     for each allowed tuple $t \in c$
03         if $u(t) \neq max$ then $u(t) = u(t) + inc$
```

Figure 3.14: Preference skewing procedure

This procedure increases all the preference values that are not the maximum possible by the specified increment value. This has the effect of reducing the gap between the preference values and lessens the domination of high values to low values[14]. Figure 3.15 continues the previous example given in figure 3.13, showing the constraint preference values and the marginalization of x_1 after one skewing procedure.

In this example, the infeasible state is no longer preferred because its preference value is not as attractive anymore. The skewing process attempts to find a balance point between preference and feasibility, allowing the marginalization to be as greedy as possible and still be able to find solutions.

Another advantage of the skewing is the ability to allow the marginalization to degrade gracefully. For very hard problems, repeated skewing will eventually

[14]In our experiment, we have used the integer values from 1 to 10 for the preference values. Incrementing the low values allows us to express the smallest possible value as 9 and the largest as 10. This translates to the highest preference being about 10% better than the lowest one. If decrementing the high values is performed instead, we will have values 1 and 2, which translates to 100% better. Thus decrementing allows for a finer control of the difference between preference values.

x_1	x_2	u
1	1	2
1	2	3
2	1	3
3	2	5

D_{x_1}	value(eqn 3.4)
1	$max(u(x_1 = 1)) + u(x_1 = 1, x_2 = 1) = 3 + 2 = 5$
2	$max(u(x_1 = 2)) + u(x_1 = 2, x_2 = 1) = 3 + 3 = 6$
3	$max(u(x_1 = 3)) + u(x_1 = 3, x_2 = 1) = 5 + 0 = 5$

Figure 3.15: Random binary CSP marginalization after skewing

lead to all the allowed tuples having the same preference values and hence the marginalization operation will simply degrade to min-conflict.

Chapter 4

Experimental Study

4.1 Algorithm

We evaluated the combined marginalization technique for approximating the bargaining set used in local negotiations and compared the solution found with one found by simply using min-conflict local search. In both cases, we terminate the search after finding the first feasible solution. Whenever possible, the optimal solution is also obtained for comparison. The actual optimal solution is found using an exhaustive backtrack search method. The solution quality is obtained by taking the product of the scores of each team individual schedule in case of MASSP or the preference value of the solution tuples in each constraint in case of random CSP.

We have simulated the multi agent system in a single machine using a system agent. The system agent is responsible for selecting which shared variable should be negotiated[1], detecting local maxima or solution, and maintaining a counter between preference skewing. In our experiment, the system agent instantiates a "broker" agent that actually performs the bargaining process. Depending on which game/variable is currently being considered, the broker agent obtains the preferences from either teams/constraints, calculates the Nash product and returns the maximum to the respective sides. Ties are broken randomly by this

[1]In our experiment, it is simply performed in lexicographic order.

broker agent. Figure 4.1 presents the algorithm[2] for the system agent.

```
01 proc system_agent_run()
02     while ($skewing\_counter < max\_skewing$)
03         $restart\_counter = 0$
04         while ($restart\_counter < max\_restart$)
05             while (not in local maxima)
06                 for all games $g_{ij}$ { $game\_agent$.negotiate($t_i, t_j$) }
07                 $global\_score = 1$
08                 for all teams $t_i$ { $global\_score = global\_score * t_i$.score() }
09                 if ($global\_score <> 0$) { return $global\_score$ }
10             end
11             for all teams $t_i$ { $t_i$.reinitialize() }
12             $restart\_counter++$
13         end
14         for all teams $t_i$ { $t_i$.skew_preferences() }
15         $skewing\_counter++$
16     end
17 return 0
```

Figure 4.1: System agent procedure

There are three nested loops which control the algorithm execution. The innermost loop (lines 05 to 10) performs the local search process by negotiating for each games. Line 09 checks whether a solution is found, and if so the search is terminated. Otherwise, the rounds of negotiation is repeated until a local maximum is detected. The middle loop performs the random restart procedure until a maximum number is reached. If we still did not find any solution, the outermost loop initiates the preference skewing as shown in line 14. Finally, if after a maximum number of skewing is reached and no solution is found, the system agent exits.

In the experiment, the preference values are between 1 and 10, and *max_skewing* is set to 10 (after 9 skewing, the resulting preference will all be 10). *Max_restart*

[2]The algorithm described is for the MASSP. For random CSP, we simply replace the teams with constraints and the games with variables. Note that while each game is connected to exactly two teams, a variable in random CSP can be connected to many constraints. However, the single item n-player Nash product can be computed accordingly.

is set to 500, so if no solution is found after 500 random restarts, we assume that the preference values are too strong and they constantly lead the search to infeasible states thus skewing is necessary.

In evaluating the solution, the global score is obtained using the actual preferences before any skewing is performed. The reason is because the skewing is done as part of our search method and the resulting skewed preferences do not reflect the true valuation of the individual schedule. Furthermore, the true valuation is necessary in order to compare the solution found with ones found by other search methods.

4.2 Results and Discussion

4.2.1 Multi Agent Sport Scheduling

Figure 4.2 shows the performance of our combined marginalization compared to min-conflict average on MASSP. Both the valued conflict projection (comb-vc-*) using equation 3.2 and table based projection (comb-tb-*) using equation 3.3 are evaluated. The table based projection was limited to ten teams due to the almost exponential size (i.e., $n!$) of the individual schedules. In creating the tables or individual schedules for table based projection, we simply enumerate all possible schedules and use the individual game preferences to obtain the individual schedule preferences just as shown in figure 3.9 on page 45. The individual game preferences are assigned values between 1 and 10 leading to schedule preferences values between $n-1$ to $10(n-1)$. Furthermore, any skewing on the MASSP is actually performed on the game preferences. The schedule preferences are then re-computed to reflect the new values.

For each team size, 25 different sets of preference values[3] are used, and for each set of preference values, 25 runs are executed. Hence each point in figure 4.2 displays the average of 625 runs. For each set of preferences, we record both the maximum and average scores obtained. The averages of these are represented with *-max and *-ave respectively.

From the figure, our marginalization outperforms min-conflict exponentially

[3]Values are randomly generated using Java Random Object in JDK 1.4.

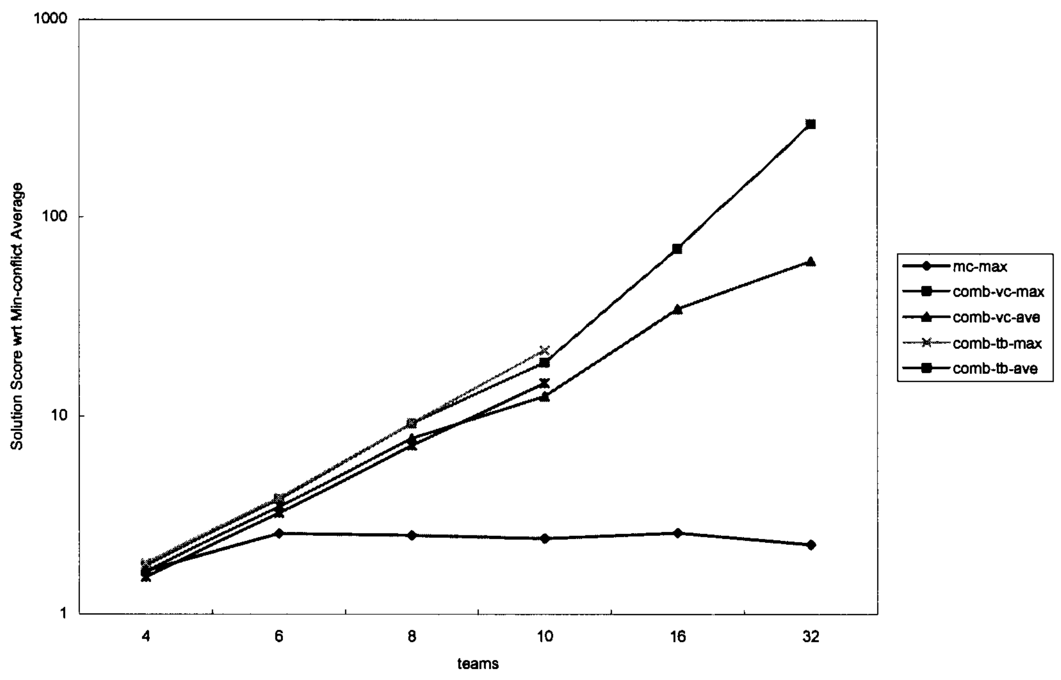


Figure 4.2: Combined marginalization performance on MASSP with respect to min-conflict

as the number of teams increases. We believe this is due to exponential increase in the number of solutions[4]. Min-conflict on average will find average quality solution while our marginalization will find solution close to the optimal. As the number of solutions increases the range between optimal and average solutions also increases, thus resulting in significantly better performance of our technique.

In comparing the results found by valued conflict projection and table based projection, we observed that former performed slightly better. This can be attributed to the fact that valued conflict projection has a better approximation to the actual preference simply because it has access to the preference function (see equation 3.1). That is, valued conflict projection is using the game preferences for its marginalization, while table based projection can only use the individual schedules preferences that are derived from these game preferences. In general, problems may not have such explicit function, and the valuation has to be estimated from the user-given preferences of the complete tuples. This is exactly what table based projection is performing, and results show that its approximation is quite good. However, it is also possible that the good performance obtained by the table based projection is due to the underlying linearity of the schedule preferences. We investigated this effect at the end of this section.

Due to the hyper exponential size of search space for MASSP, we are only able to find the optimal solutions for six teams within a reasonable amount of time. The performance with respect to the optimal is shown in figure 4.3. This figure further supports our argument with regards to the significant advantage of our marginalization. Here we see that our technique is able to achieve solutions within 80-90% of the optimal.

Since the quality of the solution is measured by taking the product of team scores, the absolute difference in score between one global schedule to the next better one can vary greatly. It is possible that the assigned preferences are such that all the global solutions scores do not have a normal distribution. For example, say that a problem has 10 solutions, where 1 is scored 100 and the rest

[4]The number of one-factorizations for a complete graph gives an indication as to the number of solutions. Results from graph theory shows that there are 6240 one-factorizations of K_8, 1,255,566,720 of K_{10}, and over 2×10^{17} of K_{12} [33].

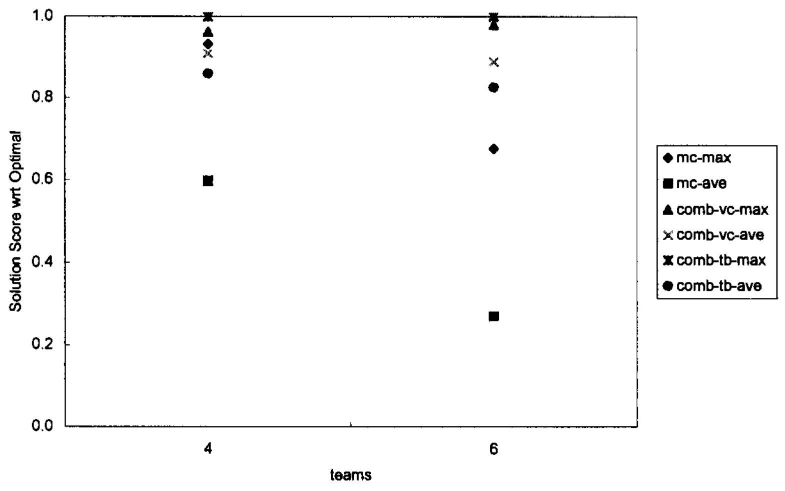


Figure 4.3: Combined marginalization performance on MASSP with respect to optimal

are scored between 10 and 20. Unless we found the optimum, any solutions obtained will be at best 20% of optimal in terms of score. Of course, the reverse scenario is also possible. In such cases, the measure of percentage with respect to optimal score may be misleading. Another measure that we use is the solution ranking. We exhaustively search for all global solutions and rank them according to their scores. The rank of any solution found using our marginalization can then be determined by comparing the solution score to this complete list of solutions. Figure 4.4 plots the number of solutions achieving certain top percentile by rank. From the figure, for 6 teams, about 95% of the 625 solutions found using our marginalization are within the top 10th percentile[5] with respect to solution ranking. Therefore, the combined marginalization is able to find close to optimal solutions.

Among the benefits of Nash bargaining solution is that the agreement is fair to all the involved parties. We evaluated the "fairness" of the solution by calculating

[5]Given the complete feasible enumeration for the all-different constraint, for 6 teams, there are 720 global solutions, and for 8 teams, there are 31,449,600 global solutions.

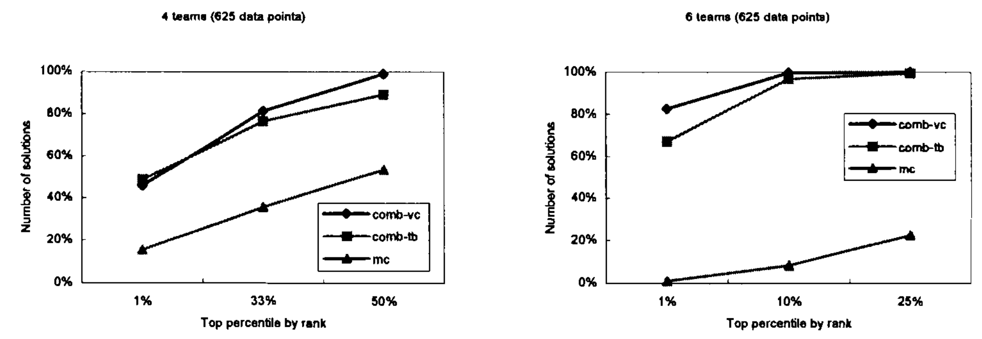


Figure 4.4: Combined marginalization performance on MASSP with respect to solution ranking

the percentage of highest scored team and lowest scored team with respect to the geometric average. That is, once the global solution is found, all the individual teams scores (the individual schedules preferences that make up the global solution) are compared[6] against each other. The best and worst scores are then normalized with respect to the geometric average of the global solution. We used geometric average since the solution quality is measured by taking the product of teams preferences. The closer the scores of these teams are to the geometric average, the fairer the bargain. The result is plotted in figure 4.5. As expected, the solution found by valued conflict projection exhibits a narrower range of high and low scores when compared to min-conflict, leading to a fair solution.

Non-Linear Schedule Preferences

We further evaluated the performance of the table based projection on a problem where schedule preferences are not derived from game preferences. Instead, the schedule preferences are assigned random values between 1 and 100. We compared the solution quality found against both min-conflict and optimal for 6 teams. Figure 4.6 shows the results with respect to score and rank. In terms of

[6]Although the Nash bargaining solution allows for incomparable agents utilities, in the experiment, we have used the same preference scales for all agents.

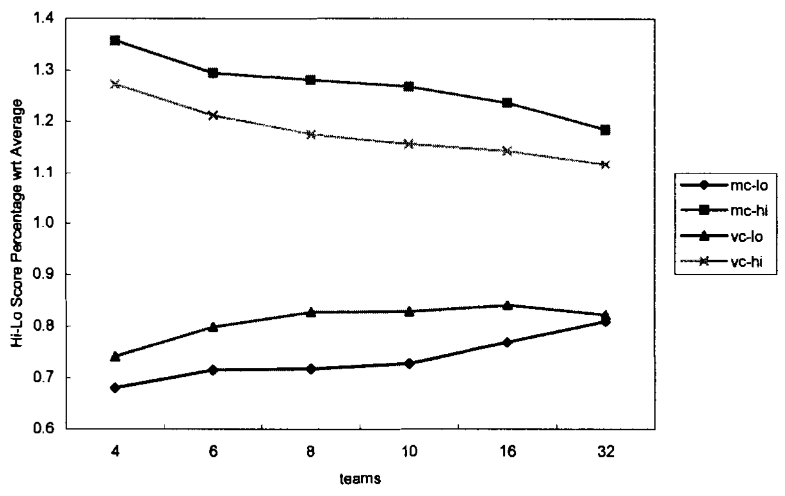


Figure 4.5: Solution "fairness"

solution score, there is a substantial decrease in performance compared to linear preferences (45% vs 80% of optimal). However, we believe that a significant amount of the decrease is due to the higher degree of abnormality in the distribution of solution scores. This is because we see a similar decrease in min-conflict (5% vs 25%), which does not take into account any preferences. This is further supported with the performance in terms of rank. There is only a slight decrease of the number of solutions (80% vs 95%) that are within the top 10th percentile. Also, the solution rankings obtained by min-conflict are identical for both linear and non-linear preferences, which is as expected. These results show that the table based projection can find good solutions regardless of the linearity of the preferences.

4.2.2 Random CSP

In case of random CSP (RCSP), we have used the following parameters: $n = 10, m = 20, d = 0.4$, and $t = 0.40 - 0.49$. The t value close to 0.49 has been identified as the hard region or *mushy* region where the expected number of solution is 1 [31].

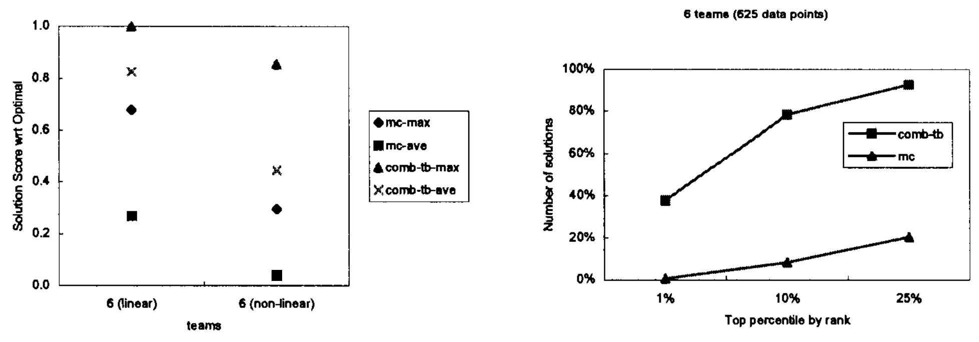


Figure 4.6: Combined marginalization performance on MASSP with non-linear preferences

These parameters lead to $\hat{p}_{2crit} = 0.454$ and $\kappa = 1.11$ (see section 2.6). Since we are interested in optimization problems, we did not go further than $t = 0.49$. Values higher than this will generate problems with almost zero solutions. Furthermore, we only tested on problem instances having solutions. The binary constraint preference values are between 1 and 10 the for allowed tuples and 0 for disallowed tuples. Figure 4.7 plots the quality of the first solution found by our technique using the equation 3.4 (comb-*) and by simple min-conflict (mc-*) with respect to the optimal solution. As a further comparison, we also plot the result using marginalization where feasibility is considered first and then any ties are broken using the preferences (i.e., actual marginalized preference) as shown in equation 4.1. The first term constant of the equation is to guarantee that the valuation is non-zero since we are taking the Nash product of the valuations.

$$f_{x_i}(a) = 1 + u(x_i = a, x_j = b_j) \tag{4.1}$$

In our experiment, for each t value, 50 random seeds[7] are used, and for each seed, 50 runs are repeated. Again, from these 50 runs, both the maximum and average are computed. The average of these values are plotted as *-max and *-ave respectively. In addition, we also plotted the average optimal score labelled

[7]The seeds are used by Java Random Object to generate both the CSP and preference values.

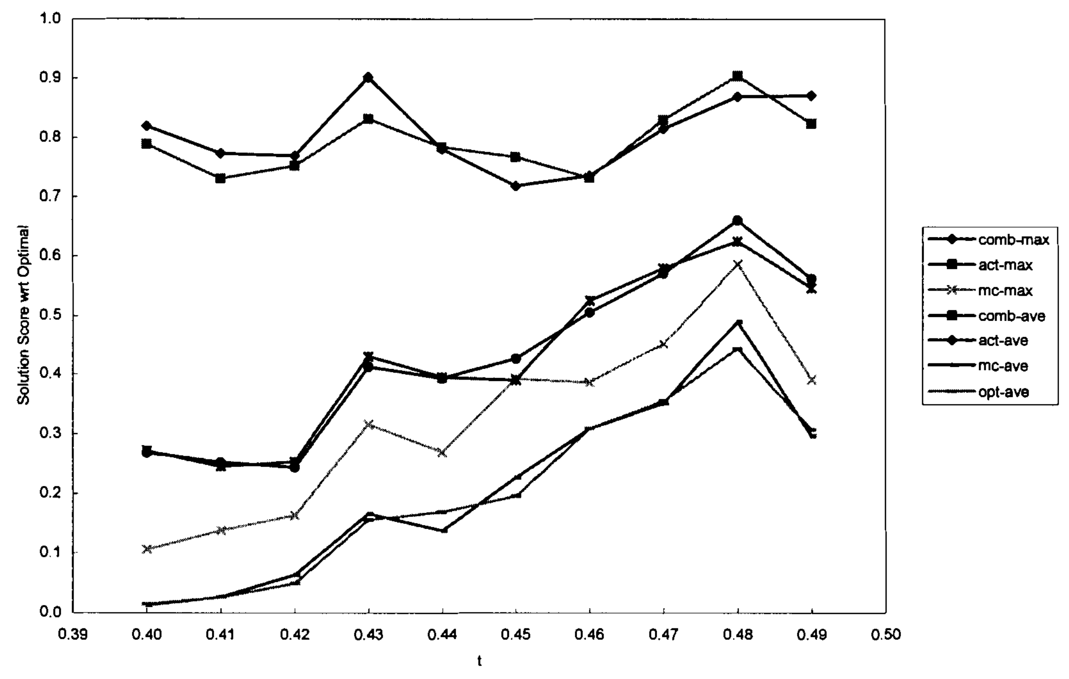


Figure 4.7: Combined marginalization performance on RCSP by score

opt-ave.

From the graph, our method (comb-*) clearly outperforms simple min-conflict. In addition, while the average solution score (comb-ave) is around 20-70% of the optimal, the average maximum (comb-max) is between 70-90%. This shows that our technique is capable of finding close to optimal solutions (note that average maximum of min-conflict (mc-max) is only at 10-60%, and the average score found by min-conflict (mc-ave) follows the actual average of the problem). However, the advantage of our method over actual marginalization (act-*) is minimal. In fact, for a few t values, actual marginalization performs better than our method. This shows that the first term of equation 3.4, has little effect on the solution quality.

We plotted the success rate of the different marginalization since a local search is not guaranteed to find a solution. Figure 4.8 shows the average success rate

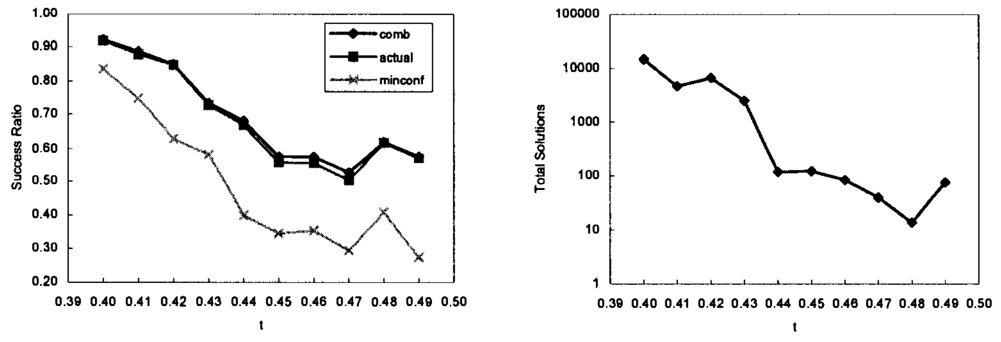


Figure 4.8: RCSP success ratio and total solutions

for both combined marginalization and min-conflict. It is interesting to note that min-conflict has a lower success ratio compared to others, since it does not have the breakout [19] behavior that the other methods exhibit due to the preference or weighting of tuples. This result shows that hill-climbing through the combined preference and feasibility landscape has a higher chance of finding solution. Figure 4.8 also shows the average number of total solutions for each t value.

We also rank the solution found to give some indication as to the distribution of the solution scores. This is plotted in figure 4.9 and it shows that on average the solution found is within 70-90% of the optimal in terms of ranking. The average maximum is at 90-100%. Note that this figure shows that while the average solution obtained by our technique is relatively poor in terms of absolute score, it is actually quite good in terms of solution rank. This is due to the RCSP not having a normal distribution of solution scores.

Finally, for both MASSP and RCSP, we calculated the average number of preference skewing performed in order to find a solution. This is given in figure 4.10. In general, the harder the problem, the more likely strong preferences to mislead the search, thus more skewing is required to lessen its effect. While the skewing allows for degradation to min-conflict in the worst case, we see that even for harder problems (32 teams and $t = 0.48$), our technique is still able to utilize the heavily skewed preferences and perform better than min-conflict.

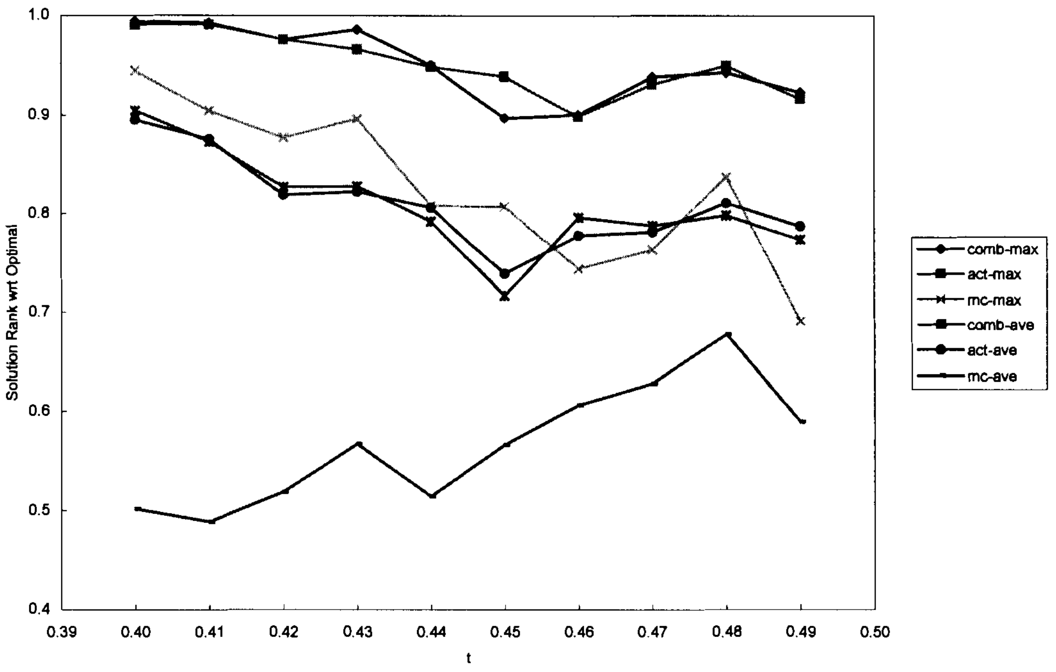


Figure 4.9: Combined marginalization performance on RCSP by rank

problem	skews	problem	skews
$teams = 4 - 6$	0.00	$t = 0.40$	6.08
$teams = 8$	0.06	$t = 0.42$	6.67
$teams = 10$	1.72	$t = 0.44$	7.22
$teams = 16$	4.54	$t = 0.46$	7.46
$teams = 32$	7.42	$t = 0.48$	7.72

Figure 4.10: Average number of preference skewing

Chapter 5

Conclusion

5.1 Summary

When solving a multi agent optimization problem, one is faced with the challenge of balancing feasibility and preference. Considering the two factors simultaneously may be more effective in guiding the search to find good solutions compared to only considering one factor at a time (e.g., branch-and-bound search). Moreover, within a multi agent setting, the preferences of every participants should be taken into account. When all agents are equal, good solutions also means ones that is fair and mutually beneficial to all. In such situations, local search techniques are preferred since they do not impose ordering of agents, and Nash bargaining solution can be employed because of its pareto optimality and independence of utility transformation. Hence, it is able to produce a fair agreement regardless of agents utility scales.

However, global Nash bargaining solution requires the computation of the global bargaining set. This is equivalent to taking the cross-product of all the constraints with preferences which is NP-hard. Instead, local negotiations can be used as an approximation. Although the true optimum is sacrificed, we found that repeated local negotiations can quickly find sub-optimal solutions.

Unfortunately, local search also suffers from not being able to distinguish between infeasible states when using traditional marginalization methods. Prior to negotiating for an agreement, an agent must first evaluate the values of all possible agreements. This is achieved by marginalizing the constraint with preferences. The result of this marginalization is simply the preference values of the constraint tuples (i.e., no real marginalization is performed) since all the variables are assigned in local search. If the tuple is infeasible, the domain element will not be included as a possible agreement. Thus, only feasible states can be considered in the bargaining. Therefore, existing marginalization methods are inadequate since they are typically only employed in constructive search, where feasibility is always maintained. In case of local search, where the current state is often infeasible, especially if there exists non-binary constraints, these marginalization are helpless.

We presented a marginalization technique which combines both feasibility and preference, and that is more suitable for local search. This approximation uses functions that can differentiate between feasible states, between infeasible states, and can prefer promising infeasible states over less attractive feasible states. It also allows for a more effective local search at the presence of non-binary constraints since it is able to distinguish between infeasible states. However, such functions could and did mislead the search to dead ends due to false promise of attractive infeasible states. In order to mitigate this, we introduced a preference skewing mechanism, where the preferences were updated so as to minimize the difference between high and low values. We combined local search with this policy and found that such repeated negotiations did lead to good solutions. Experimental results on sport scheduling and random CSP optimization problems showed that it performed better than min-conflict and did achieve close to optimal solution. On the above problems, the average maximums of solution quality obtained was about 90% of the optimal. The solutions found were also fair to all participants which showed that the Nash bargaining solution, combined with our marginalization technique, is a viable negotiation strategy for multi agent optimization.

5.2 Future Work

There are several areas in which this work can be further extended. These could include further evaluation of the performance on other optimization problems, e.g., travelling tournament problem [4], giving an indication of how our approach compared to the state-of-the-art. Instead of a simulated multi agent system, a true multi agent environment can be implemented. This will also entail a more detailed development of its coordination protocol, which could enable parallel processing and lead to a more efficient system. We have only given basic functions for the marginalization, and investigation of alternative functions could lead to better performance. Finally, instead of a random restart, the algorithm could incorporate a more sophisticated digression mechanism, such as a nogood cache, for escaping local maxima.

Bibliography

[1] Dimitris Achlioptas, Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Michael S. O. Molloy, and Yannis C. Stamatiou. Random constraint satisfaction: A more accurate picture. In G. Smolka, editor, *Proceedings of Third International Conference on Principles and Practice of Constraint Programming (CP97)*, pages 107–120, 1997.

[2] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and Valued CSPs: Basic Properties and Comparison. In M. Jampel, E. Freuder, and M. Maher, editors, *Over-Constrained Systems (Selected papers from the Workshop on Over-Constrained Systems at CP'95, reprints and background papers)*, volume 1106, pages 111–150. 1996.

[3] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.

[4] Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem description and benchmarks. *Lecture Notes in Computer Science*, 2239:580–589, 2001.

[5] Eugene C. Freuder. Partial Constraint Satisfaction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89, Detroit, Michigan, USA*, pages 278–283, 1989.

[6] I. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. Technical Report APES-08-1998, APES Research Group, 1998.

[7] Enrico H. Gerding, David D. B. van Bragt, and J. A. La Poutre. Scientific approaches and techniques for negotiation. A game theoretic and artificial intelligence perspective. Technical Report SEN-R0005, 29, 2000.

[8] Fred Glover and M. Laguna. *Tabu Search*. Blackwell Scientific Publishing, Oxford, England, 1993.

[9] Jean-Philippe Hamiez and Jin-Kao Hao. Solving the sports league scheduling problem with tabu search. *Lecture Notes in Computer Science*, 2148:24–36, 2001.

[10] Martin Henz. Constraint-based round robin tournament planning. In *International Conference on Logic Programming*, pages 545–557, 1999.

[11] Martin Henz, Tobias Mller, Sven Thiel, and Marleen van Brandenburg. Global constraints for round robin tournament scheduling. *European Journal for Operational Research*, 153(1):92–101, 2000.

[12] N. R. Jennings. Coordination techniques for distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley & Sons, 1996.

[13] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. In *AAAI/IAAI*, pages 169–174, 2000.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, 4598:671–680, 1983.

[15] S. Lauritzen and P. Shenoy. Computing marginals using local computation. *Working Paper No 267, School of Business, University of Kansas, Lawrence, KS*, 1996.

[16] Alan K. Mackworth. Consistency in network of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[17] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[18] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[19] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 40–45, 1993.

[20] J.F. Nash. The bargaining problem. *Econometrica*, 18:155–162, 1950.

[21] G. L. Nemhauser and M. A. Trick. Scheduling a major college basketball conference. *Operations Research*, 46:1–8, 1998.

[22] Russell Ovans. *A Multiagent Solution to the Venue Equalization Problem*. PhD thesis, Simon Fraser University, 2002.

[23] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of The Twelfth National Conference on Articial Intelligence*, pages 362–367, 1994.

[24] Jean-Charles Régin. The symmetric alldiff constraint. In *Proceedings of the International Joint Conference on Articial Intelligence*, volume 1, pages 420–425, 1999.

[25] J.S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers.* MIT Press, 1994.

[26] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, 1982.

[27] K.G. Russell. Balancing carry-over effects in round robin tournaments. *Biometrika*, 67(1):127–131, 1980.

[28] Robert A. Russell and Janny M. Y. Leung. Devising a cost effective schedule for a baseball league. *Operations Research*, 42(4):614–625, 1994.

[29] Tuomas W. Sandholm. Distributed rational decision making. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 201–258. The MIT Press, Cambridge, MA, USA, 1999.

[30] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.

[31] B. Smith. Phase transition and the mushy region in constraint satisfaction. In *Proceedings of the 11th ECAI*, pages 100–104, 1994.

[32] Gadi Solotorevsky and Ehud Gudes. Solving a real-life nurses time tabling and transportation problem using distributed csp techniques. In *Proceedings of CP '96 Workshop on Constraint Programming Applications*, pages 123–131, 1996.

[33] W.D. Wallis. *One-Factorizations.* Kluwer Academic Publishers, 1997.

[34] William E. Walsh and Michael P. Wellman. Marketsat: An extremely decentralized (but really slow) algorithm for propositional satisfiability. In *AAAI/IAAI*, pages 303–309, 2000.

[35] Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[36] Makoto Yokoo. *Distribution Constraint Satisfaction: Foundations of Cooperation in Multi-agent Systems.* Springer-Verlag: Berlin, Hiedelberg, New York, 2001.