Constraint-Directed Negotiation of Resource Reallocations

Arvind Sathi and Mark S. Fox

CMU-RI-TR-89-12

Center for Integrated Manufacturing Decision Systems
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

March 1989

Table of Contents

List of Figures

Abstract

The resource reallocation problem requires multi-agent choices under multiple-criteria, most of which are based on qualitative attributes. The conditional specification of a reallocation request (e.g., requiring a swap for one workstation with another) results in chains of reallocation transactions, which increase in complexity with the number of resources and agents. Also, the initial intentions for transactions may differ from the final transaction due to give-or-take on resource components. This work is motivated by the human negotiation procedures, such as logrolling, bridging, and unlinking. We view the process of reallocation negotiations as being constraint based. Constraints can be used both for evaluation of existing alternatives as well as for creating new ones. We define a set of qualitative evaluation and relaxation (alternative generation) techniques based on human negotiation problem solving. The search uses several aspects of constraints, such as constraint importance, looseness, utility and threshold levels. We evolve a mixed problem solving approach in which agents search individually in the early stages and as a group in the later stages. The constraint-directed negotiation approach is validated for the quality of solution in comparison to expert human negotiators on a variety of negotiation problems using a partial factorial design. The final version of the problem solver performs marginally better than the expert on experimental problems.

1. Introduction

Consider an engineering organization that is divided into a set of groups (agents) each possessing computing resources. As the projects each group works on change, the initial allocation of resources has to be adjusted to reflect any new requirements. Adjustment is performed through the buying and selling of resources, primarily between groups. The buying and selling process is often complex due to conditions placed upon them. For example, a group currently owning an Explorer and requiring a MicroVAX may wish to give up its explorer only if it can get a MicroVAX. As a result, a typical reallocation transaction specifies one or more buys and sells of resources among groups. A *simple transaction* involves selling of a resource from one group to another. A *trade* involves a two way exchange of workstations between two groups. A *cascade* involves an open or closed chain of buys and sells among more than two groups. Size and number of transaction chains increase rapidly with the number of resources and groups. In a typical resource negotiation, it is not uncommon to find cascade chains of 5 to 10 transactions. As each transaction involves at least two groups, it can be administered only if agreed upon by every one involved. Finally, the initial intentions for transactions may differ from the final agreed upon transactions due to give-or-take on portions of resources to make the best out of partial matches.

There are a number of constraints and objectives faced by the groups as they buy or sell workstations. For example:

- Workstations carry maintenance cost. Unused workstations reduce profit margins.
- Workstations are required for program development. Absence of the workstation may imply reduced revenues.
- Maintenance and depreciation cost allocation for workstations ranges between $1,000 to $3,200. Groups prefer the most inexpensive workstations as long as they meet the other requirements.
- Workstations differ in development and debug environment. Engineers' productivity is higher for the workstations equipped with better development and debugging tools. Such workstations are also the most expensive. Often a trade-off is struck between cost and

development environment.

- Different projects require different workstations due to the delivery software requirements which may specify the type of workstation, display mode (color or black & white), memory or disk space, etc.
- Workstations may be located anywhere in the building. They can be relocated, if needed, to bring them closer to the engineer's office. There is a cost associated with the relocation.
- Workstations can sometimes be shared by two groups. The accounting department recognizes (in terms of monthly allocation of maintenance) equal sharing of a workstation between two groups. In such a case, each group is charged for half of the share. Depending on the need, the workstations can be moved from wholly owned to shared status and vice-versa.
- Exchanges of resources may be conditional. Often the groups seek exchanges of less desirable workstations for more desirable ones. This is achieved by floating conditional bids where the sell bid is conditional on the buy bid. In the presence of conditional and non-conditional bids, sometimes the conditions can be unlinked from one bid and linked to another.
- Unnecessary changes to existing allocations are avoided. Any change involves unproductive engineering time in moving files and development environment from one workstation to another.

The goal for each group is to acquire the right mix of workstations to meet current workstation requirements while keeping the negotiation time to a minimum. Every time a change is sought by a group, fresh allocation (i.e., allocation from scratch without any regard to current allocations) can not be used as a problem solving technique. In most reallocation situations, a small percentage of resource allocations are changed. The groups would prefer not considering the rest of the workstations that are already allocated. At the same time, in otherwise unsolvable situations, some existing allocations could sometimes be changed.

We call this problem the *resource reallocation* problem. It is a sub-problem of the more general distributed planning problem in that

- there are multiple agents each possessing resources at the outset.
- each agent has one or more activities to perform in parallel, where each activity requires resources.
- each agent may require either additional or fewer resources.
- activities do not have to be selected nor sequenced.

Though conceptually simpler, the resource reallocation problem is combinatorially complex, due to the number and size of cascades that need to be performed in order to maximize the number of satisfied bids. Secondly, there are significant information requirements due to what has to be known in order to construct cascades.

Negotiation is required to solve the reallocation problem. Negotiation is composed of two phases: a *communication* phase where information relevant to the negotiation is communicated to participating agents, and a *bargaining* phase where "deals" are made between individuals or in a group (also called "social choice" [Arrow 51]. In resource reallocation, information about available bids has to be communicated minimally, while agents may individually or as a group make tradeoff decisions about how to satisfy requirements.

Prior research in negotiation has focused on protocols for supporting price system contracting [Davis & Smith 83]. What to communicate about an agent's position has been investigated by [Rosenschein & Genesereth 86]. In this case, the focus was on the communication of the payoff matrix for a single decision problem with an a priori enumeration of decisions. Bargaining was explored by [Sycara 88] where multiple agents alter their positions on multiple issues during multiple encounters. Case-based reasoning and multi-attribute utility analysis were used to alter an agent's position. Relative to this research, we are concerned less with the protocols of negotiation and more with what is to be communicated about an agent's bargaining position and how their positions are to be changed over time.

In contrast to these approaches, our methodology builds upon the concept of constraint-directed negotiation as outlined by Sathi [Sathi 86]. Within this framework agents' objectives are represented via sets of constraints together with their utilities. When a conflict occurs, agents negotiate by modifying either the current solutions or the constraints until a compromise is reached . Thus, **joint solutions are generated through a process of negotiation, which configures or reconfigures individual offerings**. In this approach, the agents do not need to know the other agents' utilities for each possible outcome of the interaction, nor do they even need to be aware of all the possible outcomes. Rather they iteratively exchange offers until a compromise is found. Beliefs about the other agents' utilities are used merely to speed up the convergence to a mutually acceptable solution.

Negotiators use different strategies for converging to a solution depending on the search space and the topology of the constraint space. For example, in an extreme situation in which constraints affect only individual offerings (as in Game Theory), a distributed negotiation in the form of a market is sufficient. However, **if most of the constraints are conditional upon multiple offerings, mediator-driven cooperative negotiation is more appropriate**. Though much literature is available on the former [Marschak & Radner 72, Baiman 82], the latter situations are frequent and not so well understood. In this paper, we report the results of research that has focused on a subset of this problem. In particular, the investigation of constraint satisfaction and relaxation processes in the reallocation of resources among agents in an engineering organization.

In the rest of this paper we describe the perspective humans use in performing negotiation: a constraint directed process. We then describe the representation of constraint knowledge and an agent's "bargaining" position. Finally, three negotiation algorithms are described. Each algorithm defines a different mode of communication and bargaining.

2. Approach

The negotiation process can be viewed as constraint directed search in a problem space. A state is defined by a set of transactions and cascades that are formed by pairing buy and sell bids. States are evaluated using constraints. Constraint-directed negotiation would simply reduce to heuristic search, if constraints were used only for evaluating negotiation positions. In constraint-directed negotiation we view constraints as fundamental in the generation of new negotiation positions. More complex transactions can be formed by the process of constraint satisfaction and relaxation. Our approach to constraint-directed negotiation defines a set of qualitative satisfaction and relaxation operators based on human negotiation problem solving [Pruitt 82]. The three operators are

- **Composition**: A grouping of buy and sell bids or transactions in order to satisfy a complex constraint. Composition takes as input a group of transactions, each of which is unacceptable on conditional constraints (see section 3.2 for definition), and creates a

cascade (a new state) that is acceptable on the conditional constraints.

- **Reconfiguration**: A change in resource attribute value in order to meet the requirements of a buyer. Workstation reconfiguration takes as input a set of workstation attribute definitions that are unacceptable on a set of requirement constraints and reconfigures them to create a new workstation (and thus a new state) that is acceptable on the requirement constraints.
- **Relaxation**: Selective constraint violation on less important constraints in order to accept a transaction that is acceptable on more important constraints. Relaxation takes as input a set of preferences. If an alternative (transaction or cascade) is acceptable or dominant on more important constraints, it removes the low importance preferences if they prefer other alternatives.

In a typical scenario, a good solution cannot be found by means of simple pair-wise exchanges. It is only by composing and reconfiguring bids and transactions can the maximum number of bids be satisfied. The difficulty is the combinatorics of composition and reconfiguration.

Now that the problem space has been defined, including the operators for generating new states via composition, reconfiguration and relaxation, the problem remains to search the space in an efficient manner. In the rest of the paper, we describe the representation of knowledge in more detail, followed by the constraint-directed negotiation strategies.

3. Representation

This section addresses the constraint representation. It elaborates the contents of a constraint, their classification hierarchy and their evaluation process. In the resource reallocation problem, most of the constraints are qualitative. That is, their utilities and importances can not be expressed using numbers. In our definitions and evaluations, we describe how the qualitative utilities are combined without the use of arithmetic operations.

3.1. Environment

Negotiation is performed among a set of **agents**. Each agent owns a set of **resources** employed by the agent to fulfill resource **requirements**. If there is a difference between the resource requirement and the resources owned by an agent, changes in resource ownership are solicited through **buy** and **sell** bids. A **buy bid** expresses an intention to acquire a resource for a specified requirement. Similarly, a **sell bid** expresses an intention to sell a specified resource. Resources have a number of **attributes**. Each resource is described as a set of **attribute-value** pairs. A requirement contains a set of restrictions on the attribute values. A **transaction** is a pair of bids, one of which is a buy bid while the other is a sell bid. Together they express a joint intention between a buyer and a seller to change the ownership of resources from the seller to the buyer. A **cascade** is a set of two or more transactions that can be executed only together.

3.2. Constraints

Constraints are used to evaluate transactions and cascades[1]. They specify the cost or the restrictions in choosing a transaction or a cascade by an agent or a group of agents for an attribute or for the

[1]The representation is an extension of the constraint representation defined in [Fox 83]

transaction as a whole. The constraint specifies preference for an alternative in the form of **utility**, minimally accepted or a **threshold** for a utility and a constraint **importance** relative to other constraints. Constraints, as well as their importance, utilities and thresholds may differ from one agent to another.

A **requirement constraint** measures the extent of match between a resource specified in a sell bid and a requirement specified in a buy bid. It takes at least two possible utilities: "acceptable" if the match is found and "unacceptable" if the match is not found. For example, a *workstation-type* constraint is a requirement constraint. It matches the value of the *workstation-type* attribute for the workstation with the value of the *workstation-type* attribute for the requirement. If the workstation value is a subset of the requirement value, the utility is acceptable, otherwise it is not. Thus, Symbolics 3640 is acceptable for a requirement for Symbolics (as Symbolics 3640 is a subset of Symbolics models), while MicroVAX is not.

In a **fixed requirement constraint**, the resource attribute value can not be changed and the requirement for the attribute value can not be waived. A **reconfigurable requirement constraint** measures the cost of changing an attribute value for a resource. For example, a workstation can be relocated from one location to another.

A **conditional constraint** measures the utility of a set of transactions as a group. It specifies whether the respective sell and buy bids should be executed conjunctively (AND) or disjunctively (OR). For example, an agent may own a Symbolics and may require an Explorer. He may specify the buy and sell bids as conditional, whereby the sell bid for the Symbolics workstation cannot be executed unless his buy bid for the explorer workstation is satisfied.

Each constraint specifies a **utility** that measures relative preference, limitation or cost for the chosen value compared to the other values and the constraint threshold. The utility itself is an attribute that can take either an ordinal or cardinal form depending on the chosen level of specification.

The constraint **threshold** measures the acceptable limit of the utility below which the constraint is considered violated. Threshold can be specified as an ordinal or cardinal attribute.

A **preference** specifies a utility without any threshold satisfaction level. Any transaction would score a satisfactory utility for a preference, though one transaction may be preferred over another. Such constraints are useful as tie-breakers among competing transactions or cascades that have equal utilities for the other constraints.

The utilities are measured as follows:

- For cardinal attributes (e.g., maintenance cost), the utility is specified as a direction (positive or negative) between the utility attribute and the attribute constrained. If the direction is positive, the objective is better satisfied with higher values of the attribute. For example, the direction is negative for the attribute maintenance cost because a lower value in maintenance cost is preferred over a high value. In addition, a transformation function can be specified that maps the attribute values to utility values. By default, the transformation function is identity (i.e., the utility value is the same as the attribute value).
- For ordinal attributes (e.g., workstation development and debugging environment), the utility is specified as a direction. A positive direction implies that the utility is in the same direction as the ordinal rank while a negative direction implies the utility opposes the direction for ordinal rank. For example, the utility direction is positive for workstation development environment as programmers prefer better development and debugging facilities. Utility specification for ordinal attributes can at best be ordinal.

- For nominal attributes (e.g., hardware platform requirement), the utility is specified for each value in comparison to the other values and the threshold. The preference is specified in the form of a "predicate" that either matches a value or does not match the value. The utility is specified for both "match" and "non-match" between the requirement and the chosen value. There may be as many relaxation specs as the possible values for the nominal attribute[2]. The match and non-match utilities specify the rank order among the utility of match, non-match and threshold. For example, if the requirement demands a "Symbolics" workstation (a fixed requirement), the predicate would match a workstation of the type "Symbolics" and would not match for any other type of work-station. The match-utility is greater than threshold while the non-match utility is less-than threshold (as the constraint is non-flexible).

The constraint **importance** specifies the priority order among the constraints for the agent. Importance can be specified using either ordinal or cardinal attributes. In our discussions, we will restrict ourselves to an ordinal definition of importance.

3.3. Negotiation Position

A **position matrix** is a set of transactions or cascades evaluated on a set of constraints. A row represents a single transaction and its utilities for each constraint. A position constraint is the aggregate utility for a transaction or cascade on all the constraints specified in the position matrix.

	Constraint			
Constraint	C_1	C_2	C_3	C_4
Importance	I_1	I_2	I_3	I_4
Threshold	Th_1	Th_2	Th_3	Th_4
Transaction/ Cascade				
T_1	U_{11}	U_{12}	U_{13}	U_{14}
T_2	U_{21}	U_{22}	U_{23}	U_{24}
T_3	U_{31}	U_{32}	U_{33}	U_{34}

Figure 3-1: Negotiation Position

For example, figure-3-1 shows a position-matrix. T_1, T_2 and T_3 are transactions (i.e., sell/buy combinations). C_1, C_2, C_3 and C_4 are constraints with thresholds Th_i and importances I_i and U_{ij} is the utility of constraint C_i for transaction T_j.

Constraint utilities can be compared across alternatives transactions or cascades. The result of evaluation places the alternative in either of the following three categories:

- Unacceptable (the alternative is unacceptable on the constraint)
- Acceptable (the alternative is acceptable on the constraints)

[2]The partial matches are currently ignored

- Dominant (the alternative is dominant over other alternatives on the constraint)

Having generated an evaluation on each constraint, how do we build an overall evaluation for the alternative that combines the evaluations on individual constraints? A number of combinational strategies are available through the researchers of behavioral decision theory [Tversky 1969, Payne 1976, Svenson 1979, Johnson & Payne 1985]. Some of the well-known strategies are as follows:

- Additive utility: Given a set of utilities for constraints, the overall utility is the sum total of all the individual utilities. In order to execute this strategy, the utilities must be cardinal as it requires an arithmetic operation.
- Additive difference: This can be used to compare two transactions. To compute additive difference, first we compute the difference between utilities for the two alternatives on each constraint. These differences are then added. If the sum is positive, then the first alternative is better. Otherwise, the second alternative is better. Like additive utility, this method also requires cardinal utilities.
- Elimination by aspects: Given a threshold utility for the constraint, the utility for each constraint for a transaction/cascade is compared with the corresponding threshold or the utilities on other constraints. Transactions with low utilities on any constraint are eliminated from the consideration. This process continues till only one transaction remains. Unlike the above combination rules, elimination by aspect can be used if the utilities are ordinal as it requires no arithmetic operation. It can also be used for identifying an acceptable subset of transactions or cascades (by using constraint threshold as the comparison point).
- Lexicographic semi-order: It is similar to elimination by aspects. It examines the transactions on each constraint and eliminates those which have values less than the dominant alternative. The rule is applied first by using the most important constraint and then by using the second most important constraint, and so on. Like elimination by aspects, this rule requires only ordinal utilities and importances. It can be used for identifying the dominant set.

In a comparative study, Johnson and Payne found a remarkable reduction in the comparison effort for elimination by aspects and lexicographic semi-order. The accuracy of these rules depends on the distribution of importance and the emphasis placed on the non-compensatory nature of the domain. Thus, if the importances are too far apart, a combination of elimination by aspect and lexicographic semi-order are sufficient to identify the acceptable set and to select from within the acceptable set. These strategies can be used by individual agents to identify their favorite alternatives. They are not usable in consolidating the preferences across agents.

Now, how do we apply elimination-by-aspect and lexicographic semi-order to position evaluation by an agent? The positions themselves are viewed as constraints. They carry composite utility, which is an ordinal measure with the following categories:

1. No bid: There exists no bid in the position matrix.
2. No transactions: There exist only buy and sell bids but no transactions. Buy and sell bids alone are insufficient to make a transaction.
3. Unacceptable: The transaction or cascade evaluates to less than threshold for a constraint.
4. Acceptable: The transaction/cascade is above threshold on every constraint but does not dominant every other transaction/cascade.
5. Dominant: The transaction/cascade dominates every other transaction/cascade.

The above classification is also a ranking of alternatives (in the ascending order).

The position constraints can be classified into the following categories:

1. Fixed requirement position (FP) measures the overall evaluation for a transaction on requirement constraints. Their utility is derived from the utility for the individual requirements using the following algebra:

   "Unacceptable-bid" + "Unacceptable-bid" = "Unacceptable-bid"

   "Unacceptable-bid" + "Acceptable-bid" = "Unacceptable-bid"

   "Acceptable-bid" + "Acceptable-bid" = "Acceptable-bid"

   In other words, if "unacceptable-bid" were assigned a rank of 1 and "Acceptable-bid" were assigned a rank of 2, the composite match takes the lowest of a set of utilities.

2. Transaction position (TP) measures the overall utility for a transaction. It follows the same algebra as described above.

3. Cascade position (CP) measures the overall utility on constraints applicable to cascades such as conditional constraints.

4. Agent position (AP) measures the overall utility of all of an individual agent's constraints.

5. Group position (GP) measures the overall satisfaction of all the negotiating parties. It combines all the agent positions.

Using elimination by aspects and dominance analysis, we can evaluate each of the above positions for a set of transactions into any of categories defined above (no bids, no transactions, unacceptable, acceptable and dominant). Using lexicographic semi-order and the preference constraints, we can further search among acceptable transaction for the dominant ones.

4. Negotiation Operators

Using the insight of human negotiations, we define, in this section, three constraint satisfaction and relaxation operators - composition, reconfiguration and relaxation - that can be used for modifying transactions or cascades, for creating new cascades, or for changing the evaluation of transactions. The human negotiation literature refers to these strategies as bridging, unlinking and logrolling, respectively [Pruitt 82].

4.1. Composition

Composition satisfies a constraint by means of composition. Composition occurs between two negotiators when a new option is developed by combining together two existing alternatives which satisfy both parties' most significant constraints. In the context of resource reallocation, composition is the act of grouping or combining buy and sell bids, transactions or cascades in order to meet conditional constraints. Suppose there exist two transactions $T_{i1\ j1}$ and $T_{i2\ j2}$ each of which are acceptable to the respective buyers on the non-relaxable resource requirements. Also, suppose one of the negotiating parties specifies a conditional constraint ($S_{i2}\ B_{j1}$) that prohibits selling workstation in the sell bid S_{i2} without buying for requirement specified in B_{j1}. A cascade formed by combining the two transactions satisfies the conditional constraint.

Composition is typically used by resource reallocation negotiators as a strategy for building cascade solutions. Each agent has a set of conditional bids which need to be executed together. By bringing two

transactions together, agents build cascade solutions that bridge on their most important needs.

4.2. Reconfiguration

Cooperative negotiations, in which both parties get all they were seeking, occasionally occur as a result of the discovery of a fortunate composition formula. But such agreements are not always available. It is usually necessary for one or both parties to make selective changes in the offerings to derive a mutually agreeable alternative. Reconfiguration is a process of regrouping the bundle of negotiated goods. Suppose there exist two buy bids for .5 workstation each and a single sell bid for a workstation. The two halves of the workstation can be unlinked from each other to provide viable sell alternatives for each buy bid. Regrouping may occur to define a new workstation configuration.

As an example of reconfiguration, agent m requires a Symbolics running the KC-3-2 tool suite. Agent n is offering a Symbolics running KC-3-1. The Symbolics offered by agent n can be reconfigured. Thus, the current tool suite will be removed from its disk and a new one loaded. The end result is what agent m desires. This requires several hours of an engineer's time to change the workstation configuration.

There are many reconfiguration possibilities in the workstation negotiations:

- Committed allocations of resources to requirements are reconfigurable. It is possible for an agent to enter new bids into the market place by breaking existing resource and requirements pairs, generating sell bids for the resources and the buy bids for the requirements. Thus, in a negotiation situation where the existing buy and sell bids are not matching with each other, additional bids can be introduced through reconfiguration, thereby increasing the number of options for each bid.
- Conditional bids can be reconfigured. Given a conditional bid (S_{k1}, B_{k2}) and an unconditional sell bid S_{k3} from the same agent, B_{k2} can be unlinked from the conditional bid and reconfigured with S_{k3}.
- Resource definitions are a way of packaging workstation components into a single package. Depending on the needs, resources can be reconfigured. The most obvious example is that of breaking the ownership of a single workstation into a shared ownership where the use of workstation in the morning is unlinked from the afternoon use. Similarly, two agents may swap memory boards, disk drives or displays separately from the workstation, or may move to the workstation from one location to another.

The focus of reconfiguration is a conceptual group G = $\{m_i\}$ where m_i are the group members. Similar to Simplex iterations on decision variables, reconfiguration searches for a group in which the value of holding the members together in the group is less than the value of breaking the group members apart and re-organizing the groups. The groups in the above example are existing resource allocations, conditional bids and resource configuration definitions. Our focus of attention in reconfiguration are those changes that result in strict or weak increase in utilities without trade-offs. Thus, the utility on each constraint should either increase or remain the same.

4.3. Relaxation

Relaxation is the process of ignoring constraints in case a chosen alternative is unacceptable on a specific constraint. When bargaining concerns a set of issues, it may be possible to arrange an exchange of concessions. One party concedes on issues A, B and C, and the other on issues D, E, and F. This exchange will be successful in reconciling interests to the extent that the parties have differing priorities across the set of issues - concessions on A,B and C being minimally costly to the first party but providing

considerable benefit to the second, and concessions on D, E, F being minimally costly to the second party yet providing considerable benefit to the first. This process is called logrolling in human negotiations [Pruitt 81].

Given a set of preferences for transactions, the agents need to find a common preference. The common preference is not easy to find. First of all, a single agent may find it difficult to aggregate preferences on different constraints. In addition, there is no easy mechanism for voting on preferences across agents. Relaxation provide an approximate technique for selecting transactions or cascades that perform the best on the most important constraints for each individual.

Much as voting or weighting techniques can be used to identify only near dominance, but not marginal dominance, the relaxation technique suffers from the same weakness. In a decision between two alternatives, if one is almost dominant, then it can be spotted. If there are close runners, then one may use random assignment or more exact methods for finding a solution. Relaxation does not always result in unambiguous conflict resolution but sometimes it can be used for breaking ties. Under the assumption of ordinal information for the importances and utilities, this may be the best we could do.

5. The Negotiation Process and Experimental Results

		Complexity		
		Small	Medium	Large
Size	Small	No of Constraints Small	No of Constraints Medium	No of Constraints Large
	Medium	No of Constraints Large	No of Constraints Small	No of Constraints Medium
	Large	No of Constraints Medium	No of Constraints Large	No of Constraints Small

Figure 5-1: Latin Square Design

	Cell								
	1-1	1-2	1-3	2-1	2-2	2-3	3-1	3-2	3-3
Conditions									
Size	Low	Low	Low	Med	Med	Med	High	High	High
Complexity	Low	Med	High	Low	Med	High	Low	Med	High
No of constraints	Low	Med	High	Med	High	Low	High	Low	Med
Managers	3	3	3	6	6	6	9	9	9
Workstations	9	8	7	36	25	27	61	81	75
requirements	9	9	9	36	28	21	61	72	59
sell-bids	1	3	2	7	8	14	10	30	36
buy-bids	1	4	4	7	11	8	10	21	20
Fixed reqs	1	7	6	14	28	8	30	15	20
Relaxable attrb	1	5	4	14	22	4	30	15	19
Total trans	1	12	8	49	88	112	100	630	720
Conditionals	0	1	1	0	1	5	5	15	17

Figure 5-2: Cell Statistics

Three search strategies have been investigated for the efficient generation of negotiated reallocations. To test their performance, a double-blind experiment[3] was performed over a latin square of nine classes of problems (see figures 5-1 and 5-2). The latin square rows differentiate the test cases on size. Thus, each cell in the first row has three agents and on an average nine workstations-requirements pairs. Each cell in the second row has six agents and up to thirty-six workstations-requirement pairs. Each cell in the third row has nine agents and up to eighty-one workstations-requirement pairs. The columns differentiate on the complexity. Thus, each cell in the first column has no workstation/requirements imbalance, has each requirement fully specified using workstation parameters, and only 20% of the population has reallocation bids. Each cell in the second column has an average of 10% imbalance, only 75% of the workstation attributes are specified in the requirements and 30% of the population is being reallocated. Each cell in the third column has 20% imbalance, only 50% of the workstation attributes are specified for requirements and 40% of the population is being reallocated. The superimposed third dimension differentiates in terms of number of constraints (3, 6 or 9).

[3]The same 9 problems were given to the program and to an expert. Their performance were rated by another expert without knowledge of who generated the solutions.

5.1. CDN I

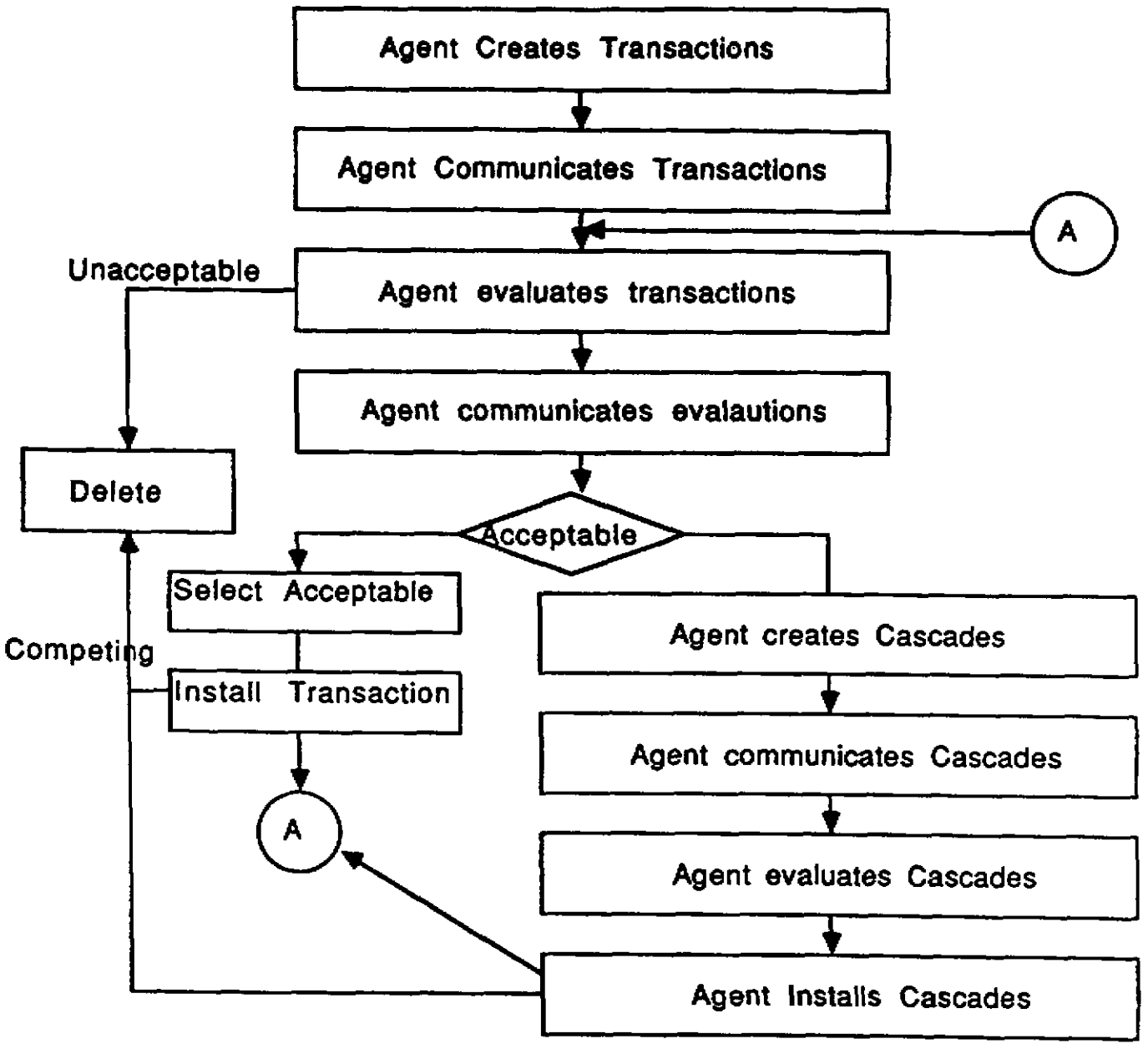
The CDN I algorithm investigated the use of the composition operator in a completely distributed, non-negotiated, greedy approach to reallocating resources. At the outset, each agent possesses a set of resources, buy bids, sell bids, and constraints on both bids and resources. In the first step, each agent communicates its sell bids to all other agents. Consequently, each agent has only a partial model of the potential set of transactions. Each agent creates a position matrix containing its buy bids and all known sell bids in order to identify potential transactions. Each buy-sell pair is combined into a potential transaction if it satisfies the agent's own fixed requirement constraints. Potential transactions along with the position constraints (i.e., aggregate evaluation of the known constraints for each potential transaction) are communicated to all agents. If an agent's buy bid participates in more than one potential transaction, a deadlock constraint, specifying that the installation of a transaction cannot be performed without agreement of the other participating agent, is created and communicated to the other agent. If any transaction is acceptable as it is (i.e., without any relaxations on the conditional constraints, or deadlock constraints), then it is installed by the buying agent. Installation of a transaction by an agent is communicated immediately to other agents so that it can be removed from further consideration.

Once all the transactions are exhausted, the focus moves to the creation of cascades by using the composition operator to construct conditionally constrained transactions. If the resulting cascades are acceptable on conditional requirement constraints, they are installed and communicated. Competing cascades generated by a single agent are differentiated among using preference constraints. If different agents have competing cascades, preference is given to the agent which generated its cascade first. Figure 5-3 flow charts the decision process.

Experiment 1 evaluated the performance of CDN I over 90 randomly generated situations. The performance of CDN I was poor:

- All the bids with acceptable transactions on fixed requirements were not installed. In the most complex transactions (size-high complexity-medium and size-medium complexity-high), the system failed to install transactions though they were identified during relaxation of fixed requirements This is because many acceptable transactions compete with each other at both transaction and cascade levels. Choice of one of the competing cascade results in the removal of competing transactions and cascades. If the corresponding buy or sell bid has no other transactions, the bid remains unresolved. See figure-5-4.
- The CPU time increased rapidly as the cases became more complex. To focus on the causes, we plotted the CPU time/bid as the number of bids were also increasing rapidly (which increases the complexity of the problem exponentially). Both the number of bids and the CPU time/transaction increases non-linearly with the increase in the size. A careful examination revealed that most of the increase in CPU time comes from the need for maintaining consistency among local knowledge-bases (one for each agent). In the version of algorithm used for this experiment, no thought was given on whether a piece of information should be kept in a local or shared memory. Thus, there are trade-offs to be considered. Keeping information in common memory would slow down access and require task synchronization. Keeping information in local memory increases number of messages in order to maintain consistency of data. See figure-5-5.
- The CPU time per transaction increased with the number of constraints from low to medium but then it decreased from medium to high. This was an interesting behavior. Initially the CPU time increases as the effort increases, but then after a point adding more constraints builds tighter and fewer matches, thereby reducing the total time for comparisons. See figure-5-6.

Figure 5-3: CDNI Algorithm



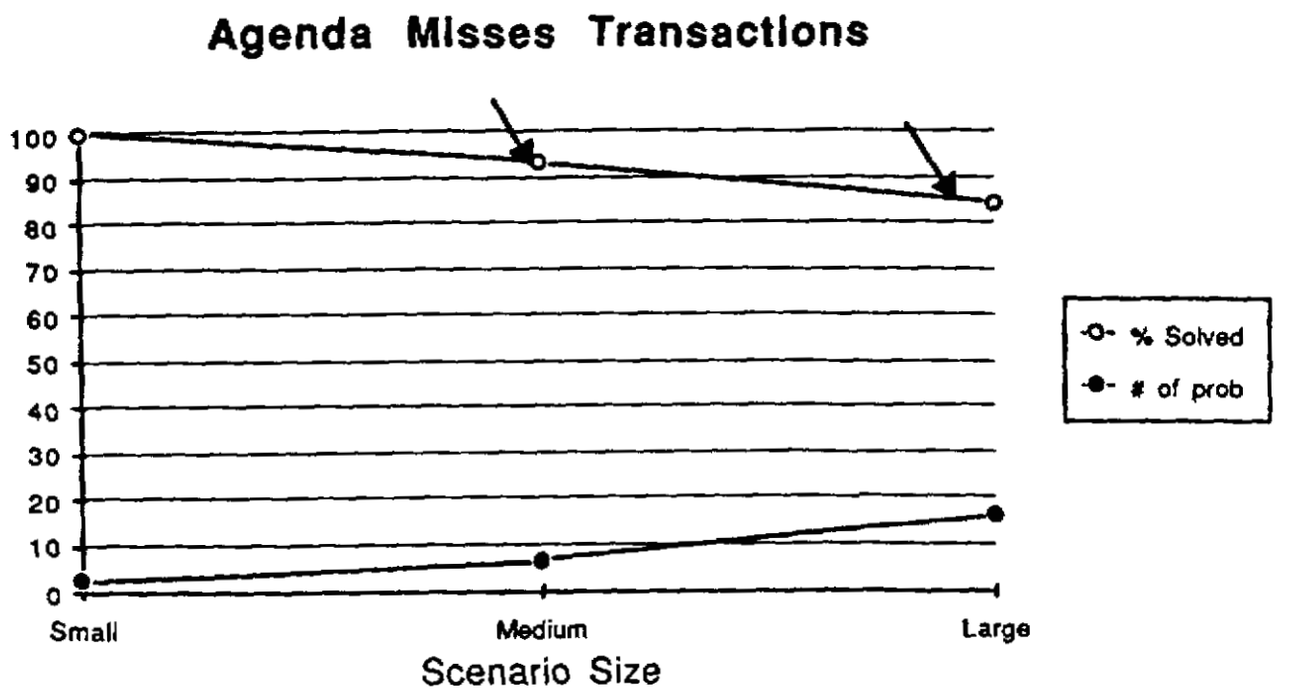


Figure 5-4: CDN I performance - missing transactions

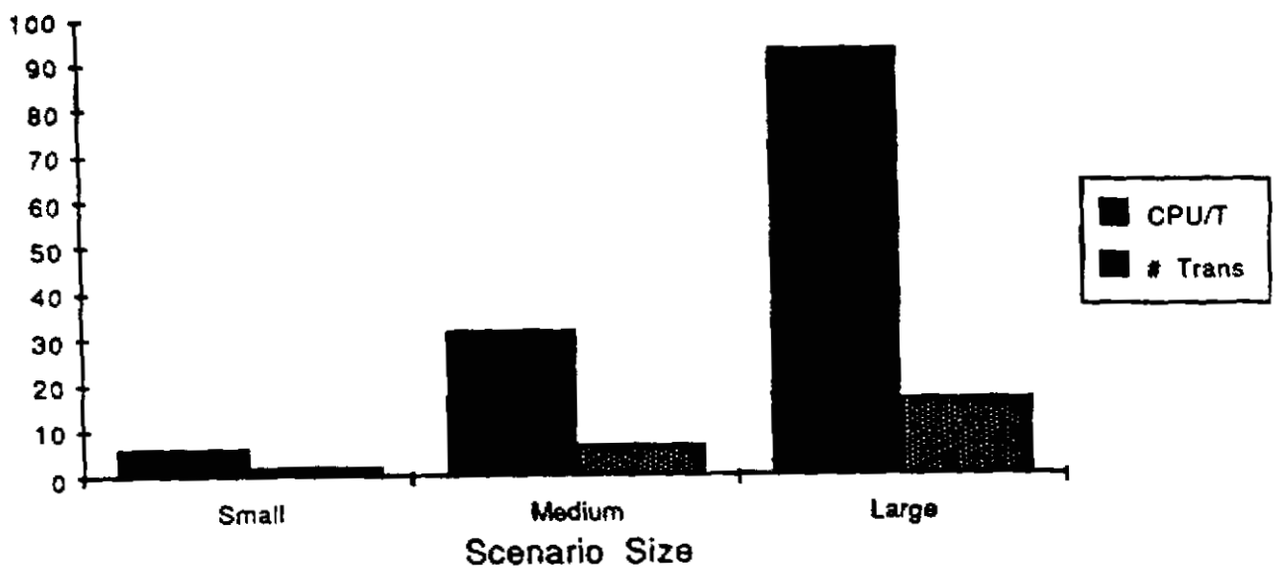


Figure 5-5: CDN I performance - CPU Time by Size

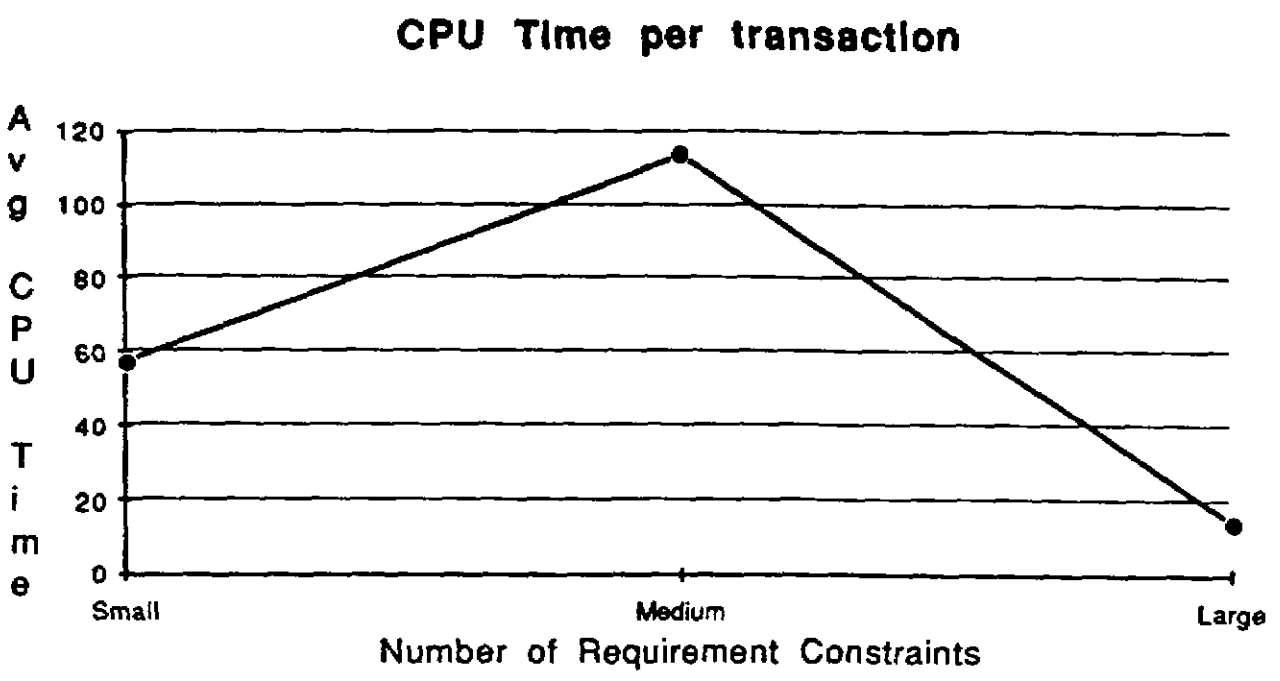


Figure 5-6: CDN I performance - CPU Time by Number of Constraints

5.2. CDN II

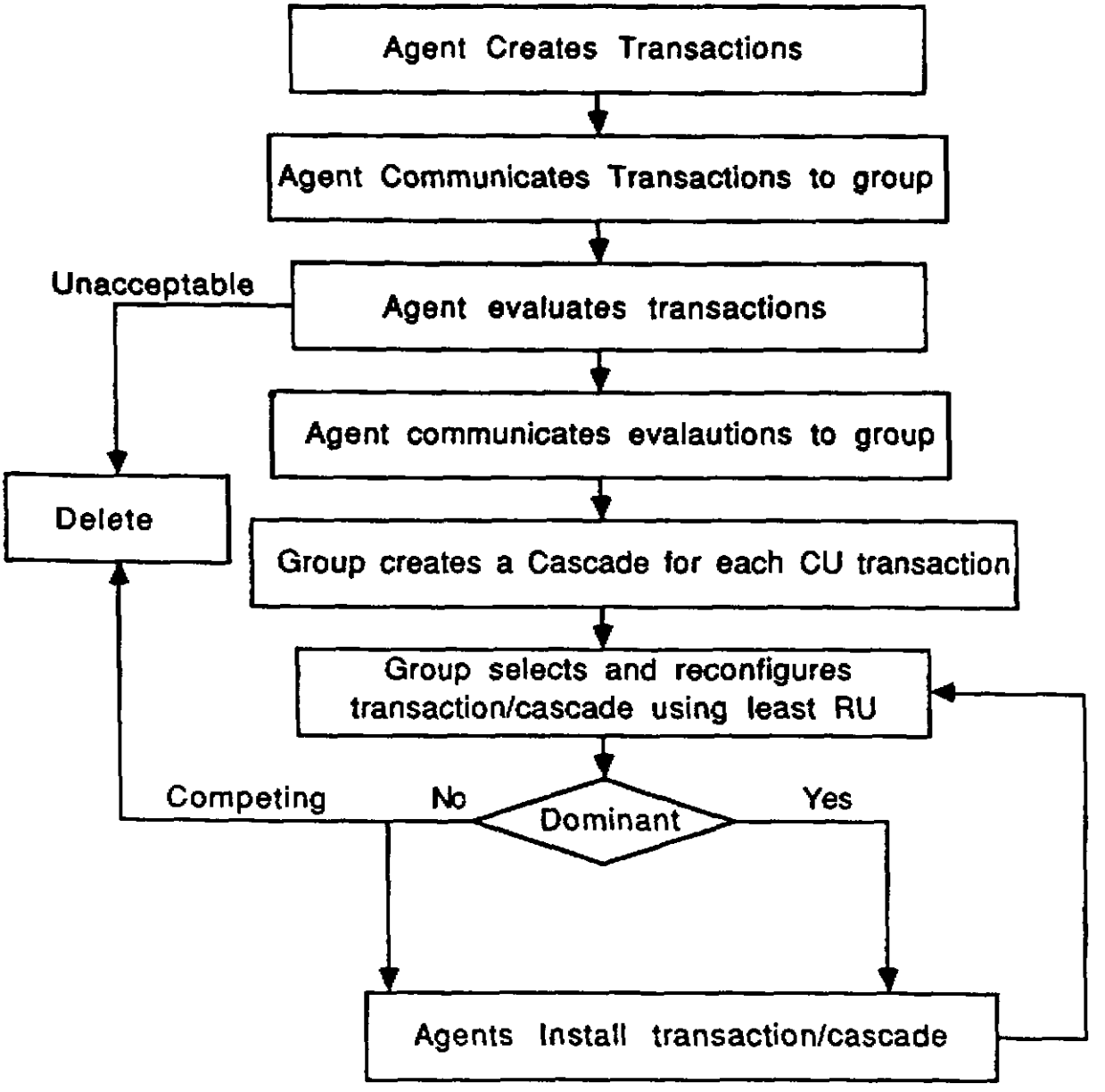
The problem of not be able to satisfy all bids in the CDN I algorithm was due largely to the combination of each agent's myopic view in constructing compositions and being too opportunistic in its selection of transactions to install. Since each agent knew only of their own buy bids, they installed immediately transactions that were of only local value. If an agent had delayed in installing transactions and waited to find out about the transactions constructed by other agents, then it could have (1) constructed cascades that satisfied the conditional constraints and (2) chosen alternative transactions to maximize the remaining bids. Accordingly, we refined the algorithm into CDN II that delays the selection of transactions until after cascades have been formed.

To obtain a more global view, each agent must communicate all potential transactions, position evaluations, and cascades. An agent's constraints are not communicated. Rather than have each agent construct its own transactions and cascades, CDN II algorithm uses a central mediator to perform this task. The purpose of the mediator is to compose cascades that maximize the number of bids satisfied. All transaction and cascade evaluations are done individually by each agent, while composition, reconfiguration, and relaxation are performed by the mediator.

In CDN II, the reconfiguration and relaxation operators are introduced. The evaluation of reconfigurable requirements results in either acceptable or unacceptable utility. If a transaction is unacceptable on reconfigurable requirements, then it can still be selected, though incurring a cost of reconfiguration. This selection minimizes the number of reconfigurations[4].

[4]Note that a count of reconfigurable requirement violations is still an ordinal operation on the utilities.

Figure 5-7: CDNII Algorithm



The CDN II reallocation process is described as follows (also see figure 5-7):

1. For each buy and sell bid pair, the group create a transaction in common memory.
2. Each agent evaluates the transactions associated with his buy bids on requirement constraints.
3. Each agent deletes all the transactions which are unacceptable on fixed requirement constraints. All the remaining transactions are acceptable to the respective buyers.
4. The group uses transaction composition to create an acceptable cascade for each acceptable transaction. If the transaction carries no conditional bids, it is acceptable, otherwise the transactions are grouped with other transactions using the conditional constraint evaluations located in individual agents' memory and position evaluations stored in the group memory.
5. If there exists an acceptable cascade that does not require reconfigurations and does not compete with any other cascade (i.e., does not have any buy or sell bids in common with any other cascade), then the group selects it and installs the changes proposed by the buy and sell bids. If there are no transactions or cascades, then the algorithm stops.
6. If there exist a set of acceptable cascades that do not require reconfigurations and compete with each other on buy or sell bids, the group chooses a cascade with the maximum number of bids (using relaxation operator to break ties). Each affected agent installs the chosen cascade and deletes all the competing cascades. Steps 5 and 6 are repeated for each acceptable cascade.
7. If there exists a set of acceptable cascades that require reconfigurations, then the group selects one with the least number of reconfigurations (using relaxation to break ties) and applies the reconfiguration operator to derive the acceptable cascade. Each agent installs the chosen cascade and deletes all the competing cascades. Steps 5 to 7 are repeated for each cascade requiring reconfiguration.

The above algorithm essentially carries each transaction through a number of levels shown in figure 5-8. No commitment is made unless all the transactions pass through levels 1, 2, 3, and 5[5]. Level 1 carries all the new buy and sell bids. At this level, buy and sell bids are paired to form transactions. Each such transaction is moved to level 2 where it is evaluated around requirement constraints. All the unacceptable transactions are dropped to level -1 where they get deleted; the rest are moved to the cascade evaluation level (level 2) where they are evaluated around the conditional constraints and moved to level 3. At level 3, all the evaluations from individual agents are combined to build group evaluations. On the basis of the group evaluation, each transaction is moved to any of the above four levels. All the transactions that are already dominant (i.e., they do not compete with other transactions on any bid and are acceptable on all the fixed and conditional constraints) are moved to level 4, where they are selected for installation. All the transactions that require composition as they currently violate conditional constraints are moved to level 5, where transaction composition is applied to generate new cascades that no longer violate conditional constraints. The acceptable transactions are moved to level 6 where relaxation is applied to select among competing transactions. Level 7 gets all the transactions that currently violate reconfigurable requirement constraints and unlinks the transactions to create new definitions for workstations and sell bids. Levels are executed in the order, smallest first. Thus, if there are any dominant transactions, then they are executed before those waiting to be bridged.

The selection criterion for level 5 is acceptability on reconfigurable requirement constraints. Level 6

[5]The only exceptions are the unique dominant bids. That is, those without deadlock constraints.

selects transaction or cascade with the largest number of bids first and breaks ties using preferences. Level 7 selects the transaction or cascade with the largest number of bids first and breaks ties using the number of reconfiguration changes.

```
-----------------------------------------
Level 7: Transactions requiring Unlinking
-----------------------------------------
Level 6: Transactions requiring Relaxation
-----------------------------------------
Level 5: Transactions requiring Composition
-----------------------------------------
Level 4: Dominant Transactions
-----------------------------------------
Level 3: Group Evaluation Level
-----------------------------------------
Level 2: Conditional Evaluation Level
-----------------------------------------
Level 1: Transaction Evaluation Level
-----------------------------------------
Level 0: New Bids
-----------------------------------------
Level -1: Unacceptable Transactions
-----------------------------------------
```

Figure 5-8: Search Levels in CDN II

The algorithm produced good results and was placed in competition with the expert (experiment 2) using the Latin Square design described earlier. The results are summarized in figure 5-10. The expert outperformed CDN II for both very small and very large problems, whereas differences were only marginal for the rest of the problems. The results are somewhat contrary to expectation, especially in the large cases. The computer program considers many more alternatives than the expert does, but still loses on the quality of results. How did the expert outperform the computer program, specially in large cases? Verbal traces of the expert's problem solving process show the following patterns of search:

- Emphasis on conditionals: In each experiment, the expert started the problem solving by grouping all the bids into three categories - single buys, single sells, and conditionals. The expert worked on the conditionals first and used the conditionals themselves for forming the chain as far as possible. The computer program did resolve the conditionals first but did not give any extra weight to forming chains within the conditionals only. As a result, the first cascade from the expert is typically bigger than the first cascade from the computer program. The average chain length is 2.3 for the first chain made by the expert while that of the computer program is a mere 1.1.
- Emphasis on rare bids: The expert used opportunistic selection when encountering a less populous workstation type or a requirement with too many attributes. No such intelligence was built into the selection mechanism of the computer program.

5.3. CDN III

There were two major differences between CDN II and the expert. The expert gave higher emphasis to the conditional bids in the early stages of search, thereby generating cascades with bigger chains. She also used her knowledge of the less populous workstations to select rarer buy bids opportunistically. This extra knowledge was enough to offset the extra effort in identifying larger number of cascades by the computer program. Our first reaction was to explore the possibility of searching for bigger chains during

composition.

Unfortunately, search for bigger chains during composition was not feasible under the current algorithm. CDN II resolves all composition alternatives before moving to commitments. Any extra search during composition produces exponential growth in the CPU time. When we tried to apply a refined version of CDN II that searched for bigger chains, the system showed limits to CPU time (for experiment cell 3-2, the algorithm ran overnight just generating cascade chains).

One solution is to use domain-specific information for selecting transactions for composition. The search can easily be modified using the findings of the verbal protocols by specifying domain-specific heuristics (e.g., "Search for a Sun workstation"), but the intelligence incorporated will not work if the environment changes. It is important to study the root cause for the difference in performance and build measures that use the structure of problem space rather than the surface-level domain-specific heuristics. This section analyzes the structure of the problem space and builds a modified algorithm that takes advantage of the structure in selecting candidates during relaxation.

In an earlier section, we discussed a number of attributes related to constraints. The first is constraint importance. Fixed requirements are considered more important than the conditional constraints. We also discussed the relaxation utility. Given that the utility for preference is higher than the threshold even if the preference is not met, the relaxation operator associated with the preferences is applied last. None of these attributes captured the structure of the problem space. Often in job shop scheduling, a bottleneck resource is given preference in the early stages of the search [Ow 84]. Presumably, the problem space is smaller around the bottleneck (less in the number of alternatives) and hence can be opportunistically attacked first. The question is how to map the problem space and identify areas that are "narrower" than others?

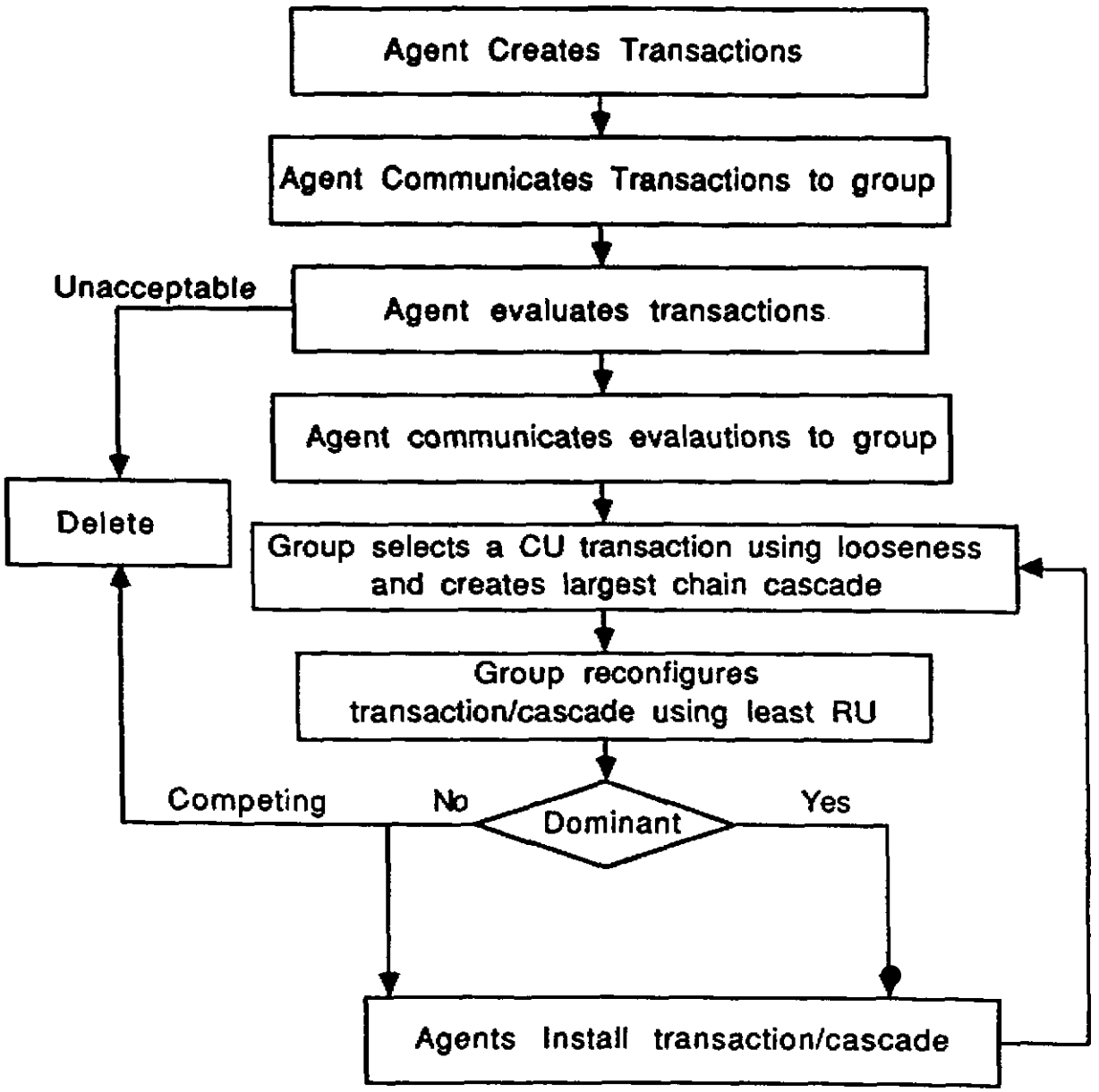
The above insight can be formalized into the concept of **looseness**. The looseness of a constraint measures the number of relaxations (alternatives) per constraint. Thus, for requirement constraints, looseness is the number of transactions that can meet a requirement. Since, each transaction has only one buy bid, looseness for the transaction is the same as looseness for its buy bid. An inverse function can be used to measure how many constraints are affected by a relaxation.

Looseness can be used for prioritizing the relaxation process. Thus, looseness for requirement constraint can be used for prioritizing composition candidates. Similarly, looseness in conditional constraints can be used for prioritizing reconfiguration, and so on. As discussed above, if we look for larger chains during composition, we need to have good starting points. Such a starting point would have a low value for looseness on requirement constraints. While the CDN II algorithm tried to generate a better solution through more exhaustive search, the CDN III algorithm uses a more intelligent selection during composition.

The modified algorithm is as follows (also see figure 5-9) and refers to the solution levels in figure 5-8.

1. For each buy and sell bid pair, create a transaction and move the transaction to level 1.
2. Evaluate the transaction on fixed (non-relaxable) requirement constraints.
3. Move all the transactions which are unacceptable on fixed requirement constraints to level -1, move the rest to level 2.
4. Evaluate transactions on conditional constraints in level 2 and move to level 3.

Figure 5-9: CDNIII Algorithm



	Latin Square Cell								
	1-1	1-2	1-3	2-1	2-2	2-3	3-1	3-2	3-3
Conditions									
Size	Low	Low	Low	Med	Med	Med	High	High	High
Complexity	Low	Med	High	Low	Med	High	Low	Med	High
No of constraints	Low	Med	High	Med	High	Low	High	Low	med
Scores									
Expert	2	2	1	2	1	1	1	3	0
CDNIII	2	2	3	2	3	3	3	1	4
Constraint violations									
Unresolved buy									
Expert	0	1	2	0	3	0	2	0	0
CDNII	0	1	2	0	3	0	0	3	6
CDNIII	0	1	2	0	3	0	1	0	0
Unresolved sell									
Expert	0	0	0	0	0	6	2	9	16
CDNII	0	0	0	0	0	6	0	12	22
CDNIII	0	0	0	0	0	6	1	9	16
Reconfigurations									
Expert	0	0	0	0	1	2	8	6	8
CDNII	0	1	1	0	0	2	0	4	7
CDNIII	0	0	0	0	0	1	7	9	3
Broken conditionals									
Expert	0	0	1	0	0	0	0	1	4
CDNII	0	0	0	0	0	1	0	2	4
CDNIII	0	0	0	0	0	0	0	2	0
Total constraint violations									
Expert	0	1	3	0	4	8	12	16	28
CDNII	0	2	3	0	3	9	0	21	39
CDNIII	0	1	2	0	3	7	9	20	19

Variation in number of constraint violations

	Variation		
	Low	Med	High
Size			
Expert	4	12	56
CDNII	5	12	60
CDNIII	3	10	48
Complexity			
Expert	12	21	39
CDNII	0	26	51
CDNIII	9	24	28
Constraints			
Expert	24	29	19
CDNII	30	41	6
CDNIII	27	20	14

Figure 5-10: Experiment 2, 3 Results

5. At level 3, group transaction evaluations from agents and move the transaction to one of level 4, 5, 6, or 7.
6. At level 4, if there exists an acceptable cascade that does not require reconfigurations and does not compete with any other cascade (i.e., does not have any buy or sell bids in common with any other cascade), select it and install the changes proposed by the buy and sell bids. If there are no acceptable cascades, stop.
7. At level 5, use looseness values to sort the transactions requiring composition. Start with the smallest value of looseness and find a cascade that is acceptable on conditional constraints. If there is more than one choice, select the one with the largest chain. Move to level 6,7, or 4 depending on whether competing cascades are found and evaluation on reconfigurable requirement constraints.
8. At level 6, if there exist a set of acceptable cascades that does not require reconfigurations and compete with each other on buy or sell bids, then choose a cascade with the maximum number of bids (use relaxation operator to break ties). Install the chosen cascade and delete all the competing cascades.
9. At level 7, if there exists a set of acceptable cascades which require reconfigurations, select one with the least number of reconfigurations (use relaxation operator to break ties) and apply unlink algorithm (see [Sathi 88]) to derive the acceptable cascade. Install the chosen cascade and delete all the competing cascades.

The steps are repeated based on which levels are non-empty. Levels are prioritized - if a lower level has any accumulated bids or transactions, then it gets executed first. The only exception is after a cascade is found at level 5.

The quality of results generated by CDN III exceeds those of CDN II and the expert (see figure 5-10). The expert violated 72 constraints, CDN II violated 77 constraints and CDN III violated 61 constraints. CDN III consistently performed better than the expert except for one case involving medium complexity and small number of constraints where the expert performed marginally better.

There are many deficiencies yet to be resolved and offer potential for future research:

1. We discovered some interactions between constraint importance and looseness in our experiments. Search strategies need to take such interactions into account while deciding upon how to select states and operators. This is an area which requires further experiments to understand how these two constraint attributes interact with each other. For example, CDN II excels in experiment cell 3-1. This cell had the most uniform looseness and the highest number of constraints per bid.
2. We did not deal with situations which required a mix of qualitative and quantitative reasoning. In a number of situations, qualitative reasoning is used to identify one or two good alternatives and then quantitative reasoning is used to fine tune or select one of the alternatives. For example, in the workstation negotiations, quantitative reasoning can be used in the final stage of the negotiation to bargain the sharing of reconfiguration costs for selecting among the competing alternatives. Due to this deficiency, CDN algorithms cannot be used in situations involving bargaining.
3. We did not study the impact of knowledge sharing on the evaluation of constraint relaxation processes. In a distributed situation, not only the constraint evaluations but also alternatives or new ways of relaxation may be shared across the negotiators. This would form an interesting extension towards a truly distributed negotiation solution.
4. Real systems are often created in decision support mode. CDN can be extended into a decision support system where the problem solver can be used for most of the time consuming pattern matching and search processes and the human decision-maker can

create new forms of relaxation or decide which constraints to logroll. CDN has a lot more potential as a decision-support than as an automated problem solver. How the mixed mode problem solving will actually work is the subject for another research.

6. Conclusions

Our goal is to explore the concept of constraint-directed negotiation as a means to achieve plan synchronization and resource allocation. Our approach has been to solve a narrow version of this problem on real data in order to understand the complexity of the decision process and the associated search processes. A number of important insights have come out this research:

- In order to maximize the number of satisfied resource buys and sells, it is necessary to generate solutions via composition to satisfy conditional constraints, decomposition of configurations to satisfy reconfiguration constraints and relaxation of requirements to satisfy portions of constraints.
- Maximization can occur only when multiple agents cooperatively negotiate, thereby creating complex cascades of transactions.
- The choice of cascades can be viewed as an opportunistic process in which looseness is the metric of choice.
- The participation of more than two agents in the cascading process is best achieved in a mediated, group problem-solving mode.

The last point is somewhat surprising. In situations involving conditionality between the decisions of three or more agents, mediated solutions appear to be better than totally distributed negotiations, where agents conduct their own negotiations.

7. References

[Arrow 51]
Arrow, KJ.
Social Choice and Individual Values.
Wiley, New York, 1951.

[Baiman 82]
S. Baiman.
Agency Research in Managerial Accounting: A Survey.
Journal of Accounting Literature, 1: 154-213, 1982.

[Baykan & Fox 87]
C. Baykan & M. Fox
Opportunistic Constraint Directed Search in Space Planning
In Proceedings of the International Joint Conference
on Artificial Intelligence, Milano Italy, 1987.

[Cammarata 83]
S. Cammarata, D. McArthur, R. Steeb.
Strategies of Cooperation in Distributed Problem Solving.
Tech Report N-2031-ARPA, The Rand Corporation, 1983.

[Durfee 85]
E. Durfee, V. Lesser, and D. Corkill.
Coherent Cooperation Among Communicating Problem Solvers.

Technical Report, Department of Computer Science and Information Science, University of Massachusetts - Amherst, Massachusetts 01003, September, 1985.

[Fox 83]
M. Fox.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.
PH.D. thesis, Department of Computer Science, Carnegie-Mellon University, 1983.

[Fox 86]
M. Fox
Observations on the Role of Constraints in Problem Solving
In Proceedings of the Canadian Society for Computational Intelligence, 1986.

[Genesereth 86]
M. Genesereth, M. Ginsberg, and J. Rosenschein.
Cooperation Without Communication.
In Proceedings of the Fifth National Conference on Artificial Intelligence, pages 51-57. 1986.

[Johnson & Payne 85]
E.J. Johnson and J.W. Payne
Effort and Accuracy in Choice
Management Science 31(4):395-414, April 1985.

[Lesser 83]
Victor R. Lesser and Denial Corkill.
The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks.
The AI Magazine 4(3):15-33, Fall, 1983.

[Marschak & Radner 72]
J. Marschak and R. Radner.
Economic Theory of Games.
Cowles Foundation Monograph 22 (Yale Univ Press), 1972.

[Parunak 87]
H.Van Dyke Parunak.
Manufacturing Experience with the Contract Net.
In Michael N. Huhns (editor), Distributed Artificial Intelligence, chapter 10, pages 285-310. Pitman Publishing & Morgan Kaufmann Publishers, 1987.

[Payne 76]
J.W.Payne.
Task Complexity and Contingent Processing in Decision Making: An Information Search and Protocol Analysis.
Organization Behavior and Human Performance, 16:366-386, 1976.

[Pruitt 81]
Dean G. Pruitt.
Negotiation Behavior.
Academic Press, New York, 1981.

[Rosenschein 84]
J. Rosenschein and M. Genesereth.
Deals Among Rational Agents.
HPP- 84-44, Stanford Heuristic Programming Project, Computer Science Department, Stanford University, Stanford, CA 94305, December, 1984.

[Sadeh 88]
N. Sadeh, K. Sycara, M. Fox, J. Hynynen, and A. Wittmann.
Trends in Coarse-Grained Distributed AI.
Technical Report, Intelligent Systems Laboratory ,The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1988.
(working paper).

[Sathi 86]
Arvind Sathi, Thomas E. Morton, and Steven F. Roth.
Callisto: An Intelligent Project Management System.
The AI Magazine 7(5):34-52, Winter, 1986.

[Sathi 88]
Arvind Sathi.
Cooperation Through Constraint Directed Negotiation:
Study of Resource Reallocation Problems.
PH.D. thesis, Graduate School of Industrial Administration, Carnegie-Mellon University, 1988.

[Shaw 83]
Jeng-Ping Shaw, Andrew B. Whinston.
Distributed Planning in Cellular Flexible Manufacturing Systems.
Technical Report, Management Information Research Center, Krannert Graduate School of Management, Purdue University, West Lafayette, Indiana 47907, 1983.

[Smith 80]
Reid G. Smith.
The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver.
IEEE Transactions on Computers , 1980.

[Smith 83]
S. Smith.
Exploiting Temporal Knowledge to Organize Constraints.
CMU-RI-TR-83-12, Intelligent Systems Laboratory, The Robotics Institute, Carnegie Mellon University, 1983.

[Steeb 81]
R. Steeb, S. Cammarata, F. Hayes-Roth, P. Thorndyke,P. Wesson.
Distributed Intelligence for Air Fleet Control.
Tech Report R-2728-ARPA, The Rand Corporation, 1981.

[Svenson 79]

O. Svenson.
Process Descriptions of Decision Making.
Organization Behavior and Human Performance, 23:86-112, 1979.

[Sycara 87]
Ekaterini P. Sycara.
Resolving Adversarial Conflicts: An Approach Integrating
Case-Based and Analytical Methods.
PH.D. thesis, School of Information and Computer Science, 1987.
Georgia Institute of Technology.
Also appeared as Technical Report GIT-ICS-87/26.

[Tversky 69]
A. Tversky.
Intransitivity of Preferences.
Psychological Review 76: 31-48, 1969.