University of Cincinnati

Date: 4/26/2013

I, Divya Paliwal, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:

A Negotiation Protocol for Optimal Decision Making by Collaborating Agents

Student's name: **Divya Paliwal**

This work and its defense approved by:

Committee chair: Raj Bhatnagar, Ph.D.

Committee member: Prabir Bhattacharya, Ph.D.

Committee member: Ali Minai, Ph.D.

3573

University of Cincinnati

Date: 4/26/2013

I, Divya Paliwal, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:

A Negotiation Protocol for Optimal Decision Making by Collaborating Agents

Student's name: **Divya Paliwal**

This work and its defense approved by:

Committee chair: Raj Bhatnagar, Ph.D.

Committee member: Prabir Bhattacharya, Ph.D.

Committee member: Ali Minai, Ph.D.

3573

A Negotiation Protocol for Optimal Decision Making by Collaborating Agents

A thesis submitted to the
Graduate School
Of the University of Cincinnati
In partial fulfillment of the
Requirements for the degree of

Master of Science

In the School of Computing Sciences and Informatics
Of the College of Engineering and Applied Sciences
By

Divya Paliwal

B.E. Rajasthan University, India,

April 2013

Thesis Advisor and Committee Chair: Dr. Raj Bhatnagar

Abstract

In modern world, where need for scaling and distributed computing is getting more crucial, distributed systems are becoming integral part of technology. An important issue in distributed environment is to make common decisions that optimize the requirements of all components in the system. Through our thesis we provide a negotiation model for decision making in a multi-agent system, where agents have incomplete set of information about each other. Our algorithm implements the idea of equal profit sharing and direct communication among agents. Agents in our algorithm collaborate with other agents for optimizing the final solution while preserving their individual privacies. Therefore an agent prefers existing proposals from other agents to generate its proposals instead of considering proposal solely based on its personal profit. We eliminate the need of a centralizing agent and maintain that need of information flow is minimal among agents while reaching toward an optimal solution. Ultimate goal of our algorithm is to provide a solution that maximizes individual and common benefits of all agents. We validate our approach through our experimental results for the problem we have chosen. Presented thesis contains our problem, algorithm description and our results with conclusion and future possibilities of our work.

Acknowledgement

I am so thankful to almighty for creating circumstances that led me to the decision of pursuing this degree. I feel really grateful and happy after completing my research work as through this process I got to broaden my horizon for the field of computer science. I am sincerely thankful to Dr Bhatnagar for providing continuous guidance, encouragement and keeping me on track. I would also like to thank Dr Minai and Dr Bhattacharya to serve in my committee and providing their valuable feedback for future work.

Thanks to my husband Yagesh for all his support and having a great understanding with me. Thanks to my dear friends who were always there whenever I needed them for any help or suggestion related to my work. Finally, special thanks to my parents for their unconditional love, support and teachings for life.

Contents

1. Introduction	1
1.1. Multi Agent Negotiation	1
1.2. Problem Statement	3
1.3. Existing Approaches	4
1.4. Our Approach	6
1.5. Results Received	8
2. Related Work	9
3. Negotiation Framework and Protocol	15
3.1. Negotiation Framework	15
3.2. Profit Calculation	22
3.2.1. Profit Calculation for Sender	22
3.2.2. Profit Calculation for Receiver	23
3.3. Algorithms and properties associated with an agent	24
3.3.1. Properties	24
3.3.2. Algorithms	25
3.4. Complexity Analysis	33
3.5. Proof of Completeness and Convergence	34
4. Analysis of Results	35
4.1. Negotiation Conditions	35
4.1.1. Condition 1	36

4.1.2. Condition 2	36
4.1.3. Condition 3	37
4.1.4. Condition 4	38
4.1.5. Condition 5	39
4.2. Results Sets	40
4.2.1. Cost effect on Results	41
4.2.1.1. Dataset 1	41
4.2.1.2. Dataset 2	48
4.2.2. Distance effect on Results	55
4.2.2.1. Dataset 1	55
4.2.2.2. Dataset 2	63
5. Conclusion and Future Work	70
5.1. Conclusion	70
5.2. Future Work	71
Bibliography	72

List of Figures:

Fig 3.1.1 Storage Mechanism	15
Fig 3.1.2 Negotiation Cycle	18
Fig 3.1.3 Negotiation Process Cycle	19
Fig 4.1.1 Breakups for Incoming Proposal's Consideration	35
Fig 4.2.1.1 Cost Effect Dataset 1 Condition 1	43
Fig 4.2.1.2 Cost Effect Dataset 1 Condition 2	44
Fig 4.2.1.3 Cost Effect Dataset 1 Condition 3	45
Fig 4.2.1.4 Cost Effect Dataset 1 Condition 4	46
Fig 4.2.1.5 Cost Effect Dataset 1 Condition 5	47
Fig 4.2.1.6 Cost Effect Dataset 2 Condition 1	50
Fig 4.2.1.7 Cost Effect Dataset 2 Condition 2	51
Fig 4.2.1.8 Cost Effect Dataset 2 Condition 3	52
Fig 4.2.1.9 Cost Effect Dataset 2 Condition 4	53
Fig 4.2.1.10 Cost Effect Dataset 2 Condition 5	54
Fig 4.2.2.1 Distance Effect Dataset 1 Condition 1	58
Fig 4.2.2.2 Distance Effect Dataset 1 Condition 2	59
Fig 4.2.2.3 Distance Effect Dataset 1 Condition 3	60
Fig 4.2.2.4 Distance Effect Dataset 1 Condition 4	61
Fig 4.2.2.5 Distance Effect Dataset 1 Condition 5	62
Fig 4.2.2.6 Distance Effect Dataset 2 Condition 1	65

Fig 4.2.2.7 Distance Effect Dataset 2 Condition 2	66
Fig 4.2.2.8 Distance Effect Dataset 2 Condition 3	67
Fig 4.2.2.9 Distance Effect Dataset 2 Condition 4	68
Fig 4.2.2.10 Distance Effect Dataset 2 Condition 5	69

Chapter 1:

Introduction:

1.1 Multi Agent Negotiation:

With increasing usability of distributed systems and autonomous agents, finding optimal solutions that meet the set of distributed constraints is becoming of significant interest. One way to design solutions for these problems is through a combination of local computations and sufficient communication among the agents. This combination of local computations and communication has been studied well for the design of distributed algorithms. Problems where local objectives and data must be kept private and only limited communication among agents is permitted have been addressed in the field of game theory. Agents in a multi-agent system have partly common and partly competing interests. Thus, game theory addresses two types of problems, cooperative and non-cooperative sets of agents. The broad area of game theory based on bargaining strategies, is studied under the topic of multi-agent negotiation. Since multi agent negotiation involves interaction, coordination and bargaining to reach an agreement, agents with different preferences and constraints can achieve a common goal while pursuing their individual interests.

Different variants of the agent negotiation strategy have been proven useful in addressing the needs of competitive and/or cooperative environments. Different protocols of negotiation in multi-agent systems have been proposed for solving different types of

distributed problems, such as vehicle routing, networking, resource allocation, and task assignments. Since a negotiation pattern changes based on the environment and problem's requirements, we need many variants of multi-agent negotiation protocols. A number of approaches [6, 7, 8, 9, 10, 16, 17] for multi-agent negotiation have been presented in published research. Contracting, auction or bidding, coalition-formation, direct-negotiation, and their many variants, are some of the variants of these negotiations. The following paragraphs provide a brief outline of some popular existing protocols for multi agent negotiation.

Usually contracting applies to situations where one agent wants some work to be performed by other agents but wants to monitor and guide the execution. A very basic pattern of contracting is to assign the job to another agent by offering the agent some incentives. Many applications that work in a cooperative environment make good use of this strategy and strive for efficient assignment of resources or jobs among the agents.

Auction strategy is proven to be quite useful for resolving conflicts among agents [7, 8]. In a competitive environment, self-motivated agents submit their bids to a centralized auctioneer and final decision is made in favor of auction winner. An extensive use of this mechanism can be seen in competitive fields such as online trading, advertising, resource allocations and network routing protocols. Most popular auction strategies are Vickery auction, Dutch auction and English auction [11].

In many cases when a centralized broker is not necessarily needed, direct negotiation among agents is also applied in a multi agent environment. These kinds of strategies will be our focus in rest of the discussion as our algorithm implements direct negotiation among all agents involved.

Apart from trying individually, sometimes agents form coalitions in order to increase benefits. Once goals through coalition are achieved, agents in these groups act as individuals for further stages of negotiation. Various strategies for practicing coalition formation focus on different aspects of problem such as, parameters for forming coalitions, different stages for forming and splitting coalitions etc. Different variants of coalition forming can be useful in both competitive and cooperative environment. However, since our algorithm does not use any form of coalition, we won't provide much consideration to this strategy during further discussion.

1.2 Problem Statement:

Our algorithm is designed to solve a problem that consists of n agents, which are situated at different locations in a two dimensional plane. These agents are required to meet at some common point. We call this meeting point as the destination for each agent. All agents have different costs to reach the destination. Cost of an agent depends on two factors i.e. total time spent to reach the destination and the fuel consumed to drive to the destination. All agents have different relative weights assigned to their cost attributes. Based on these respective weights, cost attributes contribute in calculating the total cost to an agent for any proposal. Agents have no information about other agents' costs and relative weights. Our primary goal is to reduce the individual cost of every agent and also minimize the overall cost incurred by all of the agents together. Final travelling cost cannot be higher than the original cost of travel to the arbitrarily chosen meeting point for any agent. In other words, if a solution increases individual cost for any agent then it is

discarded, regardless of how much total saving it provides overall. Through our negotiation protocol, we try to find the most cost effective meeting point for everyone, using distributed negotiation.

The algorithm should keep striving as long as it can find a better solution than existing ones. Moreover, the algorithm should be able to exit at a suitable point rather than searching indefinitely when quality of successive solutions is not improving with further iterations.

1.3 Existing Approaches:

Different variants of multi-agent negotiation have been presented in literature. Many of them [4, 9, 10, 12], include a centralized agent to manage the negotiation process. Other agents involved in negotiation have no role apart from submitting bids or making offers. Most of the negotiation protocols that use auction or contracting strategies follow this pattern. These approaches have been proven useful in a number of complex distributed problems like network routing [7], task distribution [13, 17] and vehicle routing [10]. One common goal in all of these problems is to effectively allocate the resources to agents in a multi-agent system with multiple objects. Where a resource can be anything from transportation paths to tasks available. However, there are different types of problems in a distributed multi agent environment. Our problem described above is about finding one solution in a multi agent environment that benefits everyone to the maximum extent possible, with a negotiation algorithm.

While many researchers [2, 12] have proposed strategies that work in absence of a centralized broker and use agent-to-agent direct negotiation, there is an extra overhead of learning about other agents. An agent uses his learning to maximize his own benefits by changing the negotiation pattern based on other party's preferences and constraints. Even with incomplete amount of information about other agents, a negotiating agent forms his beliefs with time and generates offers accordingly. Especially, with scenarios of self-interested agents or buyer-seller arrangements, negotiation designs are more strategic and less cooperative. Since learning in non-cooperative world can lead to dishonest proposals, algorithms such as [8] have been presented to prevent fraudulent negotiations. Further in this thesis, we are going to discuss an assumption-free approach where an agent does not have any pre-formed beliefs. Instead, with a new proposal, it solely offers to share the rewards that it would get through execution of the proposal.

Many researchers have proposed coalition formation strategies that work for cooperative as well as self-interested agents. In a multi-agent-system, there can be multiple coalitions acting as individual agents. Once coalitions are formed, act of negotiation can be different from or just like any other existing protocol. Complexity of algorithm lies under designing interaction protocol to form coalitions. Besides, distribution of payoff among agents in a coalition group needs another algorithm design once negotiation is successful. Among other approaches that have been using distributed negotiation as underlying protocol, various factors like time constraints, communication cost, computation complexity and cost have been playing key roles. In those cases, finding a complete solution is less of a concern. Rather, emphasis has been put on meeting available constraints while proposing a solution. Through our algorithm, we propose a design for

problems where primary concern is finding the best possible solution with incomplete information available. Computations for individual agents are quite simplified and no learning and its consequent complexity is involved.

1.4 Our Approach

Our model does not include any time constraint and keeps working as long as quality of solutions improves. As stated before, our algorithm deals with incomplete knowledge transfer among agents and no provision of self-learning during negotiation. Moreover, there are direct communications among agents instead of channeling through a centralized agent. An agent that creates and sends the proposal acts itself as the broker. It forwards the proposal to all other agents directly and processes their responses before sending them the final verdict of proposal. Entire process of negotiation includes iterative exchanges of proposals from every agent to all other agents. Although, agents are self-interested in our problem but they work cooperatively as overall increased profit is eventually an increase in individual profits too.

One important condition of the negotiation process is to have a tentative initial solution. For our problem, there has to be a meeting point before starting the negotiation. All agents start their negotiations on the initially decided meeting point and keep looking for better meeting points with consecutive iterations. In our implementation, we decided the initial point to be the one that is at equal distance from all agents. Since our experiment consists of three agents, the circumcenter of their coordinate locations will be the one to have equal distance from each of them. However cost to reach this point is not

necessarily the same for every agent. In-fact, there is a high probability that due to their different costs per unit-distance, every agent will end up expending different costs to reach the destination. In order to save its resources, every agent wants the meeting point to be closer to itself. Therefore, it tries to convince other agents to change the meeting point. The strategy used for convincing other agents is to provide them some incentives for travelling to a new destination instead of the current one. Since an agent is not aware of other agents' costs, it offers a share of its savings through proposals sent to every other agent.

Once an agent receives a proposal from another agent, it compares its incentive with the extra cost it needs to pay for reaching the new destination point. If proposal provides enough profit, that is higher or equal to receiver's minimum acceptable criteria, it accepts the proposal. Else, if proposal does not meet the criteria but is somewhere close to it, agent does not entirely discard it. Rather, it considers and modifies the proposal and forwards it further as its own new proposal for negotiation. If a proposal is too low compared to its acceptable limit, the agent rejects the proposal.

In order to reach the equilibrium, all agents in our system use the following strategy.

A proposal accepted by all is considered 'Accepted Proposal' and is stored in every agent's list. A proposal rejected by any of them is referred as 'Rejected Proposal' and discarded altogether. However, there could be some proposals that are not rejected by anyone, but only some agents find them acceptable while others consider some need of modification in them. Any proposal that falls in that category is referred to as 'Modifiable Proposal'. Once a proposal is declared modifiable, agents that considered it modifiable

save it in their database for further modifications and resending. Those agents that accepted it ignore the proposal. Instead, they wait for a modified version of the proposal that works for all to arrive in upcoming iterations. A solution consists of an ‘Accepted Proposal’ and all information associated i.e individual profit of agents and total profit earned through the proposal. Total profit determines quality of a solution. As long as quality of solution keeps getting improved, negotiation process continues. When no improvement in quality of solutions gets observed for continuous x negotiation cycles, negotiation process stops. An upcoming proposal, that generates more total profit than best existing solution in database is considered an improvement in quality of solutions. For given algorithm, every agent maintains its personal database to store proposals.

1.5 Results Obtained:

We expected our algorithm to decrease the final costs for all agents. Agents with higher costs would save more than other agents for equally reduced travelling distance. Thus, those agents were also capable of providing better incentives during negotiations. Hence, meeting point was expected to shift closer to agents with higher travelling costs. Our solutions complied with our assumptions. While final meeting points were closer to the agents with higher costs, agents with lesser costs travelled more in exchange of rewards. As a result in general, final cost to meet at that destination point was lesser than original cost for every agent after including rewards. Nevertheless, relation between individual costs and distances were not quite straightforward. Negotiation results got affected by number of other aspects too such as, Coordinate positions of agents relative to each other,

differences among their costs. Any change in these elements could change the equations of extra travelling costs with rewards for agents, which could change the final solutions. For different values of costs and distances among agents, we applied multiple negotiation strategies of bargaining for agents. It was observed that no strategy was a clear winner in all cases. For instances, with lesser differences in agents' costs or their distances, better results were achieved when agents had modest criteria for accepting proposals. Nevertheless, time consumption was always higher to find solutions using these strategies. For higher distances or cost differences, results were clearly better and faster if agents had stricter benchmarks to consider the proposals for acceptance or modification.

Chapter 2:

Related Work:

Many protocols using contracting, auction, coalition formation or one-to-one negotiation have been designed considering one or more desirable goals. Very common of them are maximizing the overall output, expediting the negotiation process and establishing the equilibrium. We will discuss few existing algorithms that have been targeting one or the other of these goals. Because of commonality of goals and underlying strategies, our algorithm shares and contrasts various features of different algorithms described below. Therefore, along with introduction of algorithms, comparisons in context of the problem, goals and design are made between our approach and discussed algorithms.

Smith [9] introduced Contract Net Protocol for distributed problem solving where he applied this approach on network routing problem. According to the given approach for the problem, manager node announces the task with eligibility and bid specification along with task abstraction and expiry time. In response, all available nodes submit their bids to manager node along with node abstraction. Manager picks the one that seems most suitable to accomplish the task. Afterwards, manager node sends the award to winning node along with task specification. Nodes are selected based on the criterion of time consumption for routing. Contract net Protocol has served as a crucial platform for many other successful strategies to develop and improvise for distributed problem solving. Some of those strategies are discussed further in this paper and compared with the approach we present here. Basic Contract Net Protocol requires a managing agent to

make final decisions for all negotiations. In our strategy, we aim to eliminate the need of a centralized agent and make the negotiation process distributed among all agents.

Sandholm [10] extended Contract Net Protocol and applied it on vehicle routing problem. According to his approach, award announcing and bidding decisions are based on local marginal cost calculation for a contractee and contractor. Unlike conventional CNP, in place of negotiating tasks once, repetitive interaction approach is used here. Negotiation for a task is reconsidered by estimating its cost along with current cluster of tasks that agent is completing. This approach works for cooperative and competitive agents both. Our approach is comparable to Sandholm's extension of CNP as we used repetitive negotiation process including existing negotiations to achieve or reach as close as possible to most optimal solution to the problem. For both approaches, an agent can play role of contractor and contractee both. Moreover, such as Sandholm's approach, agents in our approach are not assumed to know about other agents' cost. However, while Sandholm's approach uses one to one interaction for task allocation, negotiations in our approach are one to all. One agent is supposed to convince all others in order to get the proposal accepted, where in Sandholm's approach, two agents can mutually decide over an agreement without any interference from a third agent. Besides, repetitive negotiations in our approach extend Sandholm's approach further. In addition to successful solutions, we use some unsuccessful solutions too for renegotiations.

Luo et al [13] proposed a multi agent negotiation approach based on Markov Decision Process for distributed task allocation. Their model consists of two agent roles in the system, process manager and resource manager. A resource manager is associated with one resource and process manager decides to map different tasks to the resource manager.

Process manager receives tasks from environment or resource managers. A resource manager may or may not be associated with different process managers based on its personal decision. When a new task arrives, process manager needs to decide whether or not to send the task to corresponding resource manager. When it comes to assign a task to a resource, task goes through some of these 9 stages; arrival, waiting, rejected, scheduling, tentative commitment, final commitment, de-commitment, end, and execution. Total 11 actions can be performed by an agent are: reject the task, schedule the task, hold the task into scheduling pool, wait, schedule the scheduling pool under some constraints, make tentative commitment, make final commitment, de commit, cancel tentative commitment, execute task and stop scheduling. Resource managers perform all of these actions. Once corresponding resource manager gets the task, it decides whether to accept or not accept it based on its own priority and policy. In our approach, a proposal goes through multiple stages although not as many as their approach. Since our approach presents a rather simplified model for distributed problem solving, there is no process manager role for an agent. An agent receives every proposal directly, sent by other agents. Functionality of the agent mentioned in our approach, is close to the resource manager. As per requirement of our problem, 1 of the 3 actions is performed on an arrived proposal i.e. rejection, hold for modification and acceptance.

Sarit Kraus [11] presented an approach to solve data allocation problem in multiple server environment. She suggested a framework where all servers make an offer one by one. During negotiation process every server has liberty to reject or accept the offer or opting out from negotiation process. Once all servers accept an offer, process stops there only. Our algorithm extends the approach on two fronts. First, we added one more

response type for modifiable proposals. This addition helps an agent to generate new proposals from existing failed but nearly successful proposals. Our experiment shows that the likeliness of a proposal getting accepted improves, if it's based on some existing proposal that was close to get accepted. Second, when it comes to exiting from the process. Rather than, going for the first accepted solution we try to look for better solution and algorithm stops when it can't find any better solution for a while. While algorithm in [12] used time-based approach for negotiation turn for servers, agents in our algorithm take their turns by order regardless of time. In our algorithm, when an agent creates an offer, it also notifies all other agents about its final status at last. All agents are aware about sender of the offer as well.

Fatima et al [2] proposed agenda based multi issue negotiation, where different agents were trying to reach a solution with common interest but everyone had different preferences. Every agent had a personal choice about order of issues over which it preferred to bargain. Initially, an agent is chosen randomly to start the negotiation process. Other agents will generate a counter offer if current agent is not able to present an agreeable offer within the given time limit. Negotiation framework is using buyer seller situation i.e one agent bargains with another agent in one to one manner rather than many agents at one time. Presented algorithm is similar to their approach in sense that agents are bargaining with incomplete set of information about other agents. In their algorithm, an agent gets to decide endogenously about order of issues over which it wants to bargain. We took the similar approach for considering personal interests with little modification. Rather than deciding the order, agent decides to assign weights to all constraints for it-self and combines all of them in a cost function. However, unlike the

former approach, our protocol is fixed and doesn't get affected by number of issues. Since different issues present in the bargaining system ultimately contribute to one cost factor, final protocol for bargaining remains same once agents decide their personal cost based on different constraints.

Apart from finding the solutions, various strategies have been generated and modified in order to deal with ethical issues of a negotiation process such as dishonesty during negotiation and contract breaching. For example, Akinine et al [4] introduced extended multi agent negotiation protocol, an extension of contract net protocol. They introduced concepts of prebidding and reassignments with definitive bidding and assignment respectively. The prime concern behind this concept was to avoid the situations of decommitment by a contractor and to prevent penalization for contractor if still a commitment is broken. In order to provide dominant solution in multiple buyer and multiple seller environment Mccaffe [6] introduced double auction strategy, also called PMD protocol. Underlying strategy for PMD is to figure out efficient number of trade k , using number of buyers and sellers. Further, Yokoo et al [8] introduced TPD protocol, to prevent fake bidding by some sellers in PMD protocol by introducing a threshold value for buyers and sellers. Xueyuan et al [7] introduced policy in Vickery auction. They applied their strategy on multi-Path Multi-Hop routing and addition of policy into protocol was useful for controlling dishonest bidding. Under the policy introduced by authors for traffic allocation, Nash equilibrium was achievable for an agent only if he made an honest bid.

Protocols discussed in last paragraph work on the assumption that agents know or keep learning during negotiation about each other. Moreover, agents in their algorithms are

negotiating in strictly competitive environments. Our approach is designed mainly for environment where overall benefit also increases with personal goals. Moreover, our strategy works in a similar fashion as stock trading in free market works. Decisions of profit sharing and earning are solely personal preferences. Considering those aspects of negotiation, our strategy works on distributing the profits among agents based on their choices. Thus, no additional regulations are needed to ensure proper following of protocol. However, we shortly discussed those mechanisms to introduce the various parameters to be considered for design of a negotiation protocol and to reason for the way our protocol is designed.

Chapter 3:

Negotiation Framework and Protocol:

3.1 Negotiation Framework:

According to our approach, the negotiation framework involves two parties:

Sender: Agent who creates a proposal and sends to others

Receiver: Each agent other than the sender

As mentioned during introduction, there is no centralized broker among agents for negotiation. For this reason, a software agents has its own storage and decision making mechanism. During the negotiation process, every agent acts as a sender on its turn for generating proposal. Rest of the time he is one of the receivers for the proposal sent by the sender agent. Sender agent uses his personal database to generate new proposal and also to save proposals. As shown in figure 3.1, Agent's personal database consists of two priority queues and one single element data structure.

Fig 3.1.1: Storage Mechanism

Accepted Proposals: Agent uses this queue to store all those proposals that are accepted by all agents. If agent is a sender, it gets this confirmation by evaluating every receiver agent's response before storing the proposal. Otherwise, if agent is a receiver, it stores the proposal after getting acceptance confirmation from sender. The queue is maintained in a sorted order based on total profit earned by all agents enclosed with the proposal. Since an accepted proposal goes in every agent's storage and all agents have equal value for total profit, this particular queue is exactly same size and order in all agents' databases. Before starting of negotiation process, queue has one element that is original solution.

Modifiable Proposals: Agents uses queue for storing those proposals, which, it thinks, can be modified for more benefits and no other agent has rejected them either. A sender agent never stores his own proposal as modifiable as it accepts that proposal by default while generating it. A receiver agent, after getting the 'modifiable' notification from sender stores the proposal only if it considered that proposal as modifiable one at first place. The queue is maintained in a sorted order based on total profit earned by the particular agents for given proposal. Those agents, who accepted the proposal but got the final notification of proposal's modifiable status just ignores the proposal and don't store it anywhere.

Last Proposal Generated: This data structure can contain only one proposal and is used to hold the last proposal generated by agent. Purpose of this storage is to help in generating a new proposal, in case if Modifiable Proposals queue is empty. As depicted

in figure 3.1.1, if there are no modifiable proposals available then agent modifies its own last proposal to create a new one. Every time the agent generates a proposal this field gets updated. Before agent starts any negotiation, original solution is stored as last generated proposal.

As mentioned during introduction, negotiation starts with an initial meeting point as original solution. At initial stage, each agent considers the original meeting point as current proposal and tries to modify it. Original profit for each agent is zero and original cost is simply calculated using cost function. As the process makes progress, all agents try to modify latest findings of meeting points for their benefits. Once it is concluded that the algorithm is generating no better solutions to the problem any more, process stops there. At this stage the best among the available solutions is considered to be the final proposal and meeting point and transactions among agents are decided according to the proposal. In case, if no agent comes up with any better solution at all that can be accepted by every one, original meeting point is considered as final proposal.

As shown in figure 3.1.2, each agent broadcasts its proposal to all other agents one-by-one. The entire process continues in multiple cycles until it comes to a stage when there are no further improvements via negotiation.

Fig 3.1.2: Negotiation Cycle

Here, negotiation is a two-way communication between a sender and a receiver but one receiver has no interaction with other receiver during negotiation process. Hence, there is no external influence on an agent's decision to accept or reject a proposal. Neither sender's negotiation strategy is influenced by external factors. Rather, its negotiation is based on relative profit it makes with the proposal.

As shown in figure 3.1.1, there are two ways an agent can generate a new proposal.

- (1) Takes the best of the existing proposals by other agents that he considers as modifiable ones. It tries to pull the meeting point given in that proposal, closer to its original location.
- (2) In case if there are no proposals as mentioned in (1), agent considers its own last proposal and create a better proposal out of it by doing the same as above.

Depending on every agent's acceptance toward the new proposal, it is stored back in the databases of agents or ignored by them. Figure 3.1.3 summarizes the final decision at receiver's end about a proposal at the end of the negotiation process.

Final decision is taken after the series of conversations between sender and all receivers individually. Loosely one negotiation can be divided in four steps:

- I. Sender sends the proposal to all receivers
- II. Receivers send their responses
- III. Sender receives responses and sends the final verdict of proposal to all receivers
- IV. Receivers store the proposal at their individual databases or discard the proposal based on the notification from sender

I. Once an existing proposal is selected using one of the two ways mentioned before, following actions are taken while generating a new proposal:

- Sender treats the current proposal and its negotiation outcomes as the original deal before modifying it.
- Sender changes the meeting point, updates its own profit and adds some incentives for all other agents.
- Incentives from sender depend on the amount of its saving/profit from the new proposal after modifying the current proposal.
- According to our approach, incentives from sender to all other agents are equal shares of its saving/profit.

II. Once the message is sent to a receiver, following sequences are performed on every receiver's part:

- Receiver evaluates the proposal by calculating the total profit that it would earn by accepting the proposal.

- The receiver agent uses the benchmark it has set to decide over the amount of profit.
- Based on its evaluation, receiver has to make one of three decisions for proposal. Accept, reject or consider for modification.
- Receiver sends the response to sender along with the proposal. If its response is ‘accepted’, or ‘considering for modification’, it updates the proposal by updating the overall profit earned after this proposal for itself.

III. Once sender receives the responses from all receivers, it evaluates all responses and sends the final notification to all as described below.

- If any of the receivers sent the ‘reject’ response, sender broadcasts ‘rejected’ notification to all receivers.
- Else, if all agents sent ‘accept’ responses, sender agent calculates total profit using everyone’s personal profit after the proposal. It updates the proposal with agents’ personal profit, total profit and replaces current meeting point with new meeting point. Finally, it sends ‘accepted’ notification along with the proposal to all. In addition to that, sender too saves the proposal in ‘Accepted Proposals’ queue.
- Else, if any agent sent ‘modifiable’ response, proposal is updated the same way as for ‘accept’ response. After updating the proposal, agent sends ‘modifiable’ notification and the updated proposal to all receivers.

IV. Once a receiver gets the final notification from the sender, he stores the proposal in appropriate priority queue if the enclosed message is ‘accepted’ or ‘modifiable’. Otherwise if final notification from sender is ‘rejected’, no action is taken from receiver’s side and process moves further to another agent as shown in figure 3.1.2.

3.2 Profit Calculation:

3.2.1 Profit Calculation for Sender:

Let's say

w_1 = weight assigned to total fuel spent in cost calculation for an individual agent

w_2 = weight assigned to total time spent in cost calculation for an individual agent

s = sender's position

m = Current meeting point

x = Meeting point to be proposed by sender

d_{a-b} = distance from point a to point b

$cost_{a-b}$ = cost of walking from point a to b

ps_k = Profit earned by sender for walking to point k instead of current meeting point

n = no of agents

i = Incentive from sender for each agent including himself

s_{id} = Sender's Id

Using continued iterations, sender keeps looking for a new meeting point between s and m , until he finds a point x where following conditions are holding true.

$$d_{s-m} = d_{s-x} + d_{x-m} \quad \text{----- (3.2.1.1)}$$

$$cost_{s-m} - cost_{s-x} > 0 \quad \text{----- (3.2.1.2)}$$

$$ps_x > 0 \quad \text{----- (3.2.1.3)}$$

Where, ps_x is calculated using following parameters

$$cost_{s-x} = w_1 * d_{s-x} / mpg + w_2 * d_{s-x} / mph \quad \text{----- (3.2.1.4)}$$

$$i = (cost_{s-m} - cost_{s-x}) / n \quad \text{----- (3.2.1.5)}$$

$$ps_x = ps_m + i \quad \text{----- (3.2.1.6)}$$

Here criterion (1) implies that point x lies in straight line between s and m . Criteria (2) and (3) indicate that sender itself saves some cost by walking to new point rather than current point, with and without including his current profit. Finally, agent generates the proposal with following format and sends to all other agents.

$$Proposal = \langle m, x, i, s_{id} \rangle \quad \text{----- (3.2.1.7)}$$

3.2.2 Profit Calculation for Receiver:

Let's say

r = receiver's position

i = Incentive from sender to receiver

m = Current meeting point enclosed in proposal

x = New Meeting point enclosed in proposal

d_{a-b} = distance from point a to point b

$cost_{a-b}$ = cost of walking from point a to b

pr_k = Total profit earned by receiver for walking to point k from its original location

Receiver evaluates the proposal using following parameters in order to decide about rejecting or accepting it.

$$pr_x = i - (cost_{r-x} - cost_{r-m}) + pr_m$$

$$i.e. pr_x = i + cost_{r-m} - cost_{r-x} + pr_m \quad \text{---- (3.2.2.1)}$$

3.3 Algorithms and properties associated with an agent:

3.3.1 Properties:

Modifiable_offers_list: A priority queue, to store those offers, which are neither accepted nor rejected by the agent. Instead, agent considers modifying those offers in order to generate new offers with more profit. Proposals are stored in a sorted order using personal profit value for agent enclosed in the proposal.

Accepted_offers_list: A priority queue, to store all offers that every agent has accepted. Proposals are stored in a sorted order using total profit value for all agents enclosed in the proposal.

Initial_Position: coordinate information of agent's original location. This information is given by user and fixed throughout the program execution. Algorithm uses this information to decide the initial meeting point and distance that agent has to travel to reach the destination.

mpg: information about 'miles per gallon' for agent, given by user.

mph: information about agent's speed, given by user.

Weights: Weights W1, W2 are constant factors (provided by user) associated with agent's fuel consumption (MPG) and time taken (MPH) respectively. Program uses these inputs to calculate the total cost of arriving at the meeting point.

Last_deal: This variable stores a copy of last offer made by the agent.

Response: Stores current response of an agent for an offer

****Here an offer is a one dimensional array that encapsulates meeting point, each agent's profit, overall profit and net amount to be paid by each agent****

3.3.2 Algorithms:

Algorithm 3.3.2.1: Put_modifiable_offer:

Agent uses this function to update *Modifiable_offers_list* by storing an incoming offer in a sorted order. Offer gets ranked based on agent's personal profit for that offer.

Algorithm:

Input: *offer*

Output: *Modifiable_offers_list* /*updated */

1. *Modifiable_offers_list.insert (offer)*
2. *PersonalProfit = personal profit for corresponding agent from offer*
3. *Modifiable_offers_list = Modifiable_offers_list. ascendingSort (personalProfit)*
4. *Return Modifiable_offers_list*

Algorithm 3.3.2.2: Accept_the_offer:

Agent uses this function to store an incoming offer in *Accepted_offers_list* in a sorted order. Offer gets ranked based on overall profit determined by fitness function for that offer.

Algorithm:

Input: <i>offer</i>
Output: <i>Accepted_offers_list</i> /* Updated */
<ol style="list-style-type: none"> 1. <i>Accepted_offers_list.insert (offer)</i> 2. <i>totalProfit = total profit value from offer</i> 4. <i>Accepted_offers_list = Accepted_offers_list. ascendingSort (totalProfit)</i> 5. <i>Return Accepted_offers_list</i>

Algorithm 3.3.2.3: Pull_best_modifiable:

This function is used to fetch a modifiable offer, which is highest in hierarchy in the list, in order to generate an offer. Once an offer is pulled, it's removed from the list.

Algorithm:

Input: <i>void</i>
Output: <i>BestOffer</i>
<ol style="list-style-type: none"> 1. <i>bestOfferOrder = sizeOf (Modifiable_offers_list)</i> 2. <i>result = Modifiable_offers_list.fetchRow (bestOfferOrder)</i> 3. <i>BestOffer = result</i> 4. <i>Modifiable_offers_list.remove (result)</i> 5. <i>Return BestOffer</i>

Algorithm 3.3.2.4: Find_cost:

Through this function, agent calculates its cost to reach a point by cost function and distance to the destination. While cost function attributes are constant, user provides distance in input.

Algorithm:

Input: <i>distance_pt</i> /*distance to destination point pt*/
Output: <i>cost</i>
<ol style="list-style-type: none">1. Calculate: $cw_1 = Weights [1] * mpg / distance_pt$2. Calculate: $cw_2 = Weights [2] * mph / distance_pt$3. Calculate: $cost = cw_1 + cw_2$4. Return <i>cost</i>

Algorithm 3.3.2.5: Make_an_offer:

Agent generates an offer by retrieving and improvising the best offer available in the Modifiable_offers_list using Pull_best_modifiable method. If that list is empty then agent uses its own last proposal and improvises it to generate a new offer.

Algorithm:

Input: <i>listOfAgents</i>
Output: <i>generatedOffer</i>
<pre> 1. If size (modifiable offer's list) > 0 OfferIn = Pull_best_modifiable Else OfferIn = Last_deal End 2. currentPt = meeting point coordinates from offerIn 3. newPt = currentPt ± x Where newPt lies on straight line from agent's location to currentPt 4. noOfAgents = size (listOfAgents) /*total number of agents*/ 5. Calculate savedCost = find_cost (distance_currentPt) – find_cost (distance_newPt) 6. Calculate incentive = savedCost / noOfAgents /* share for each agent */ 7. generatedOffer = [newPt, incentive, senderId] 8. Return generatedOffer </pre>

Algorithm 3.3.2.6: Receive_offer_and_respond:

A receiver agent receives the offer, evaluates it, generates the response (rejects, accepts or wants to modify) and sends it to the agent who made the offer to him. At this stage agent does not do anything with the proposed offer and just wait for sender's final notification about the message

Algorithm:**Input:** *offerIn, newPt, incentive***Output:** *offerOut* /* updated offer and with response */

```
1. currentPt = current destination point from offerIn
2. currentProfit = profit that receiver is earning currently, enclosed with offerIn
3. offeredProfit = find_cost (distance_currentPt) - find_cost (distance_newPt) +
incentive
4. profit = offeredProfit + currentProfit
5. modifiableRange = find_cost (distance_currentPt) * x /* x, common fraction value*/
6. if incentive > 0
    response = accepted
    else if incentive <= 0 and incentive >= modifiableRange
        response = considered_Modifiable
    else
response = rejected
end
7. offerOut = [generatedOffer, profit]
8. sender = senderId from generatedOffer
9. return offerOut to sender
```


Algorithm 3.3.2.7: Process_responses_send_final_notification_to_all:

Sender agent uses this method after receiving responses from all agents on receiving side. Agent evaluates if any one has rejected the offer or if everyone has accepted it. Based on the responses, agent himself either discards the message (do nothing) or accepts the message and send final message about offer to all agents. If offer is not to be discarded, he updates the offer with total profit and send it back with to all agents with appropriate notification.

Algorithm:

Input: <i>generatedOffer, newPt, listOfAgents</i>
Output: <i>[message, offerOut]</i>
<i>1. for each agent receiverN</i> <i> if responses(receiverN) = reject</i> <i> (i) message = discarded_proposal</i> <i> (ii) offerOut = Null</i> <i> (iii) exit from loop</i> <i> else if responses(receiverN) = accept</i> <i> message = accepted_proposal</i> <i> else</i> <i> (i) message = modifiable_proposal</i>

(ii) exit from loop

end

end

2. If message = accepted_proposal

(i) profitN = personal profit for receiverN from generatedOffer

(ii) totalProfit = $\sum$ profitN

(iii) offerOut = [generatedOffer, newPt, totalprofit]

(iv) Accept_the_offer(offerOut)

else If message = modifiable_proposal

(i) profitN = personal profit for receiverN from generatedOffer

(ii) totalProfit = $\sum$ profitN

(iii) offerOut = [generatedOffer, newPt, totalprofit]

end

3. Returns [message, offerOut]

Algorithm 3.3.2.8: Finalize_the_offer:

When an agent get final notification about offer's verdict from sender, it acts accordingly and either discards the offer or stores it in appropriate queue in its database using this function.

Algorithm:

Input: <i>generatedOffer, message</i>
Output: <i>void</i>
<pre>1. if message = accepted_offer (i) Accept_offer(generatedOffer) (ii) Exit end 2. if message = modifiable_offer (i) If response = consideredForModification Put_modifiable_offer (generatedOffer) End (ii) Exit End 3. If message = rejected_offer Exit End</pre>

Algorithm 3.3.2.9: Main Algorithm:

All agents generate their offers consecutively by taking turns. The cycle continues as long as incoming accepted solutions are better than existing ones. For a period, when there is no improvement in quality of solutions before next n number of iterations, algorithm halts the negotiation cycle. In our implementation, we decided $n=20$, considering empirical results of different n values.

Algorithm:

Input: <i>listOfAgents</i>
Output: <i>final_offer</i>
<i>1. PositionList = [Agent. Initial_Position] for all agents in listOfAgents</i> <i>2. Calculate meeting_point = circum_center(PositionList)</i> <i>3. for each agent, Agent(i) in listOfAgent:</i> <div style="padding-left: 40px;"><i>Agent(i).profit = 0</i></div> <div style="padding-left: 40px;"><i>End</i></div> <i>4. Calculate total_profit = 0</i> <i>5. Calculate initial_offer = [meeting_point, {Agent(i).profit}, total_profit]</i> <i>6. Set initially,</i> <div style="padding-left: 20px;"><i>6.1 last_deal = Initial_offer;</i></div> <div style="padding-left: 20px;"><i>6.2 for each agent, AgentN in listOfAgent,</i> <i>AgentN.Modifiable_offers_list(0) = Initial_offer</i></div> <div style="padding-left: 20px;"><i>6.3 for each agent, AgentN in listOfAgent,</i></div>

```

    AgentN.Accepted_offers_list(0) = Initial_offer

7. While (0) //loop continues until termination condition depicted in step 7.2 meets//

7.1 For each agent AgentN in listOfAgents

    7.1.1. If size(AgentN modifiable_offers_list) > 0

        offer = Agent(i).Make_an_offer (Pull_best_modifiable)

    Else

        offer = Agent(i).Make_an_offer (last_deal)

    7.1.2. For each agent, Agent(j) where  $j \neq i$ ,

        Messages[j] = Agent(j). Receive_offer_and_respond (offer)

        // Where messages is a collection, storing messages of all agents at
        receiving side

    7.1.3. Agent(i).Process_responses_send_final_notification_to_all (Messages)

    7.1.4. For each agent, Agent(j) where  $j \neq i$ ,

        Agent(j). Finalize_the_offer (offer, notification_message)

End loop 7.1

7.2 Check any agent's accepted offers' list.

    If best accepted offer is same since last n iterations,

        Exit from while loop

    Else continue.

End loop 7.

8. final_offer = Accepted_offers_list(0);    /*upper most solution in priority queue*/

9. Return final_offer

```

3.4 Complexity Analysis:

A negotiation from one agent to others is a three-way interaction. It includes sending an offer, receiving the response and sending notification. Since, during one negotiation cycle every agent negotiates with every other agent, complexity of one negotiation cycle will be $O(n^2)$, given that n is number of agents. Yet, total time spent by algorithm is highly affected by total number of negotiation cycles. Negotiation cycles are byproduct of various external factors such as cost differences among agents, agents' positions relative to each other, bargaining criteria and number of agents. Thus, complexity analysis of presented algorithm is far from straightforward calculation.

Besides, based on user's preference, number of negotiation cycles or conditions to exit from algorithm can be decided explicitly. Proposed algorithm operates in a greedy hill climbing fashion. All agents extend the negotiation process by improvising the solution based on personal preferences. Since every accepted solution is better than original, exiting the algorithm at any stage will provide a solution that is better or at least as good as original.

Based on results we received from our experiments in chapter 4.2.2, it was observed that number of negotiation cycles were directly proportional to distances among agents. The underlying reason is that agents have more meeting points to bargain over with increased distance. In other words, increasing distance among agents causes the increase in solution space. Increased solution space causes more number of negotiations to obtain the optimal solution.

Other observation was made from results in chapter 4.2.1. When agents had cost values with higher differences, number of negotiations was lesser than when agents had their costs close to each other. With high differences among agents costs, agents with Costlier travelling rate benefitted easily by providing incentives in exchange of travelling. Likewise, other agents benefitted easily by accepting incentives because of significantly lower travelling rates. Therefore, most of the proposals were approved directly without needing modifications from multiple agents, which gave optimal results with lesser number of negotiation cycles.

Above observations can be formulated for estimating the time consumption approximately as described below.

l_n = length of negotiation cycle

n_l = number of negotiation cycles

Δc = total difference in agents costs *versus* total costs of agents

d_s = Size of solutions space or total distance among agents for our problem

t_n = total time consumption in negotiation

$$l_n \propto n^2 \quad \text{----- (3.4.1)}$$

$$n_l \propto d_s \quad \text{----- (3.4.2)}$$

$$n_l \propto 1/\Delta c \quad \text{----- (3.4.3)}$$

Since,

$$t_n = l_n \cdot n_l \quad \text{----- (3.4.4)}$$

Using equations (3.4.1), (3.4.2) and (3.4.3) and (3.4.4)

$$t_n \propto n^2 \cdot d_s / \Delta c \quad \text{----- (3.4.5)}$$

3.5 Proof of Completeness and Convergence:

Our approach ensures that negotiation process will never get stuck in an endless loop and will end eventually after finding the best solution it can reach.

Our algorithm stays in process as long as it keeps finding improvement in quality of solutions. In other words, incoming proposals should be more beneficial overall than existing ones to keep the algorithm running. Hence, only way that our algorithm can go in endless loop is, when algorithm keeps increasing overall benefits infinitely.

In order to generate successful negotiations, there have to be some agents that can provide profitable rewards to other agents while saving themselves some travel cost.

As mentioned in chapter 3.1.2, criterion (3.2.1.1) permits an agent to generate a proposal only by shifting the meeting point toward its origin location, i.e.

$$d_{s-m} = d_{s-x} + d_{x-m}$$

According to condition (3.2.1.1), for any agent destination point cannot shift farther than its original location from meeting point. In addition, criteria (3.2.1.2) and (3.2.1.3) ensure that profit graph through any new proposal should only move upward for an agent that created the proposal. If an agent proposes any other meeting point after proposing its original location, it will violate criteria (3.2.1.2) and (3.2.1.3). Since, original location as destination results in zero travelling cost for an agent, shifting anywhere else from there will just increase agent's travelling cost. Therefore, if an agent keeps generating overall rewarding proposals, its last proposal will be the one with meeting point at its original location.

Above statement implies that in no case any agent is capable of endlessly proposing improvised solutions. Negotiations from any agent will come to an end once destination point coincides with its original location.

The other case when destination point wouldn't shift to an agent's original location would occur only when that agent could not offer enough profit for all agents involved in its negotiations. In that case, process will terminate eventually after observing no improvement in quality of solutions for a while.

Clearly, algorithm will eventually terminate in any case. Either the agents will stop generating proposals after sharing their maximum profit; or, a series of unaccepted or less beneficial proposals will prompt the algorithm to exit.

In either of the cases, algorithm will keep running as long as there is a scope of better proposals. That says, algorithm will reach to an optimal or close to an optimal solution. However, sometimes algorithm might get stuck in local optima if waiting time by algorithm for looking for a better proposal before quitting is too short. In order to avoid this issue, waiting time could be kept large enough to pass through local optima and reach global optimum.

Chapter 4:

Results and Analysis:

4.1 Negotiation Conditions:

Proposed negotiation protocol could create totally different results under different circumstances. Different aspects related to agents such as benchmarks for accepting proposals, position relative to each other and mutual cost differences play crucial roles in affecting negotiation results. Through our experiments, we applied different conditions of negotiation by changing the parameters S1 and S2 shown in figure 4.1.1.

Fig 4.1.1 Breakups for Incoming Proposal's Consideration

According to figure 4.1.1, S1 and S2 represent the range of accepted, modifiable and rejected proposals. Range of modifiable solutions can be reduced or expanded by moving S1 and S2 closer or distant from each other. Shifting S1 to right side increases the range of accepted proposals and lowers the upper limit for modifiable proposals. Similarly, shifting S2 to left increases the range of rejected proposals and heightens the lower limit of modifiable proposals. Our purpose here is to figure out strategies that can return least cost solutions possible with lesser consumption of time. Nevertheless, one strategy might have varying effects on results based on environmental and individual factors such as agents' location and cost to travel. Therefore, following conditions were

combined with different sets of costs and distances to examine their overall impact on quality of resulted solution and time consumption.

4.1.1 Condition-1:

An agent accepts a proposal only if it causes the agent to earn more profit than one-nth of its original cost. If agent does earn some profit but it's not as high as agent's fixed benchmark, it considers the proposal for modification in future. This particular condition is hardest to fulfill amongst all of the conditions. Any proposal accepted through this condition will be accepted in other conditions too.

Agent accepts the incoming proposal	Agent does not accepts but save the incoming proposal to modify and send to other agents in future	Agent rejects the incoming proposal
Net-Profit > original cost/n	Net-Profit <= original cost/n & Net-Profit >= 0, where n=3.	Net-Profit < 0

4.1.2 Condition-2:

An agent starts its strategy just as condition-1. But unlike **condition-1**, it doesn't keep its acceptance criterion same throughout the negotiation process. Instead, agent keeps reducing it to lower ratios such as one-(n+1)th, one-(n+2)th and so on. Any proposal causing any profit but less than expected is considered for modification else it is rejected. Hence, S1 remains at same place as condition-1 but S2 keeps coming closer to S1 with time. Underlying reason is to loosen up the constraints if present criterion is too high to

consider any proposal as a good one. If accepted, later on same solutions with improvisation might provide as good result as agent initially wanted.

Agent accepts the incoming proposal	Agent does not accepts but save the incoming proposal to modify and send to other agents in future	Agent rejects the incoming proposal
Net-Profit > original cost/n, where n=3 initially. With each iteration, n = n+1.	Net-Profit <= original cost/n and Net-Profit >= 0, where n=3 initially. With each iteration, n = n+1.	Net-Profit < 0

4.1.3 Condition-3:

This strategy was very similar to **condition-1** except that instead of using original cost, agent used its current cost to judge a proposal. Current cost is described as the amount that agent is paying for current proposal, over which the new proposal has generated. If agent does earn some profit but it's not as high as agent's fixed benchmark, it considers the proposal as modifiable one in future. Underlying notion is that for a more effective solution lesser scope of improvement will be present. If an agent doesn't lower its acceptance benchmark for improved solutions, it might miss some improvements, which could have made a good solution even better. Cost for a beneficial solution is always lesser than original solution.

Agent accepts the incoming proposal	Agent does not accepts but save the incoming proposal to modify and send to other agents in future	Agent rejects the incoming proposal
Net-Profit > current	Net-Profit <= current cost/n & Net-Profit	Net-Profit < 0

cost/n	≥ 0 , where $n=3$	
--------	------------------------	--

4.1.4 Condition-4:

For condition-4 we combined models of **condition-2** and **condition-3**. Agent's current cost is used to evaluate acceptability of the proposal. Moreover, as time progresses, agent keeps reducing the fraction of current cost to compare with the profit suggested by proposal. Similar to condition-3, agent reduces the fraction when incoming proposal is not being accepted. With this approach, S1 remains at same position and S2 keeps getting closer to S1 with a faster rate than condition-2 and condition-3.

Agent accepts the incoming proposal	Agent does not accepts but save the incoming proposal to modify and send to other agents in future	Agent rejects the incoming proposal
Net-Profit > current cost/n, With each iteration, $n = n+1$.	Net-Profit $\leq$ current cost/n and Net-Profit ≥ 0 , With each iteration, $n = n+1$.	Net-Profit < 0

4.1.5 Condition-5:

According to this approach, agents started considering proposals for modifications, which were causing a little loss. Agent accepts a proposal that generates any profit for him over his original cost. Otherwise, if offer is causing some loss, agent checks if the loss is lesser than a fraction (6^{th} in this case) of original cost. If loss is still in that range, agent considers the proposal for modification in future otherwise rejects it.

For this approach S1 shifts to same distance as **condition-3** but S2 also shifted to right side. Some of the previously rejected solutions are in modifiable range now and accepted range is same as **condition -3**.

Agent accepts the incoming proposal	Agent does not accepts but save the incoming proposal to modify and send to other agents in future	Agent rejects the incoming proposal
Net-Profit > 0	Initial Net-Profit <= current cost/n1 and Net-Profit >= -(current cost/n2). Here n1 = 3 and n2 = 6.	Net-Profit < -(current cost/n2).

4.2 Result Sets:

We applied all of the conditions on four different datasets. Two of those datasets represented effect of cost and other two represented effect of distance on overall result.

Each dataset includes following observations:

- (1. Upper left) Total distance travelled by each agent:** This graph displays distance travelled by every agent individually for the final proposal for varying x-axis values.
- (2. Upper right) Absolute final payment made by each agent:** This graph displays net payment made by every agent individually for the final proposal for varying x-axis values.
- (3. Lower left) Number of iterations to find the best solution:** This graph displays total iterations the algorithm took before reaching the final proposal. In this context, iteration consists of a cycle where all agents generate and offer their proposals one by one.
- (4. Lower right) Total cost to travel, original and final:** This graph displays total cost incurred by all agents together for travelling to the original destination point and total cost incurred by all agents together for travelling to the point suggested through the final proposal.

4.2.1 Effect of Cost on Results:

We plotted results with two sets of data with different cost values. With every set, we reduced the cost of C by changing MPG value since MPG contributes in cost of an agent. We examined these results to observe the impact of cost value on negotiation process. Costs for agents A and B were kept same and we analyzed the effect of one agent's cost on every agent. We expected C to travel more with its cost increasing. Besides, we expected overall cost to reduce with one agent in the process getting more cost efficient.

4.2.1.1. Dataset 1:

AgentA: (MPG, MPH) =(40, 50) (w1, w2)=(1, 1) coordinates=(0,-20)

AgentB: (MPG, MPH)=(10, 35) (w1, w2)=(1, 1) coordinates=(-15,15)

AgentC: (MPG, MPH) =(x, 50) (w1, w2)=(1, 1) coordinates=(15,15)

where x = (2,4, .. 40)

and $\text{cost}(\text{Agent}) = \text{distance} / \text{MPG} * w1 + \text{distance} / \text{MPH} * w2$

Results were similar in nature for all conditions with little variation in numeric values. However, condition-1 stood out among all because of its criterion and we did not see much difference in results after the negotiation for that particular condition. Given below are general observations for different graphs for all conditions.

(1. Upper left) Total distance travelled by each agent: As cost of agent C decreases, distance travelled by him increases. This change also affects other agents' travelling distances and one or both of the other agents get to travel lesser distance. However, there

comes one phase in graph where distances travelled by all agents come closer for a while. During this period C's cost is close to B, Which reduces the overall negotiation exchange, hence differences in agents' distances have also reduced.

(2. Upper right) Absolute final payment made by each agent: With reduced costs, agent C became more receiving toward incoming proposals. This led to a decrease in its net payment and increase in expensive agents' net payment. For the period where B and C had closer costs, they faced a temporary minor drop in negotiation exchange. For this reason, net payment from agents also came a little closer for a while.

(3. Lower left) Number of iterations to find the best solution: With little bit of variations, number of iterations were almost same for finding the best solution for all different values of MPG for C. Number of iterations had sudden hike for the phase where B and C agents had closer cost values. Apart from that, there was no fixed relation of time with cost factor. For **condition-2** it kept decreasing with C's cost, where for **condition-5** results were reversed. For **condition-3** and **condition-4** it was almost constant except few bumps.

(4. Lower right) Total original cost to travel for all agents and total cost after finding the best proposal: With decrease in C's cost, total original cost for all agents kept reducing. Final total cost kept decreasing too but with a different rate than original cost. Difference between both total costs was minimum at points where C's cost was closer to both A and B's total cost.

Detailed descriptions of results for different negotiation criteria of agents are discussed further in this section.

Condition-1:

- As derived from **figure 4.2.1.1**, only during initial phase, where C's cost is too high, final destination is different than original.
- Only solution that got produced here was most time consuming among all as well.
- With very high criterion for accepting proposals, no proposal got accepted afterwards for given condition.
- Least effective solution saves no money and most effective solution saves around 60.83% with 166 iterations. Average saving is 2.84%, where average number of iterations is 4.20.

Fig 4.2.1.1 Cost Effect Dataset 1 Condition 1

Condition-2:

- As shown in **figure 4.2.1.2**, acceptance criteria being comparatively relaxed than **condition-1**, there was a solution for every cost value of C.
- Number of iterations to generate best proposal was not the least but not as high as **condition-1**.
- Least effective solution saves 13.6% over 61 iterations and most effective solution saves 60.63% over 29 iterations. Average saving is 26.9%, where average number of iterations is 43.5. Comparing this result with **condition-1** suggests that overall quality of solutions can be improved by loosening up the negotiation constraints.

Fig 4.2.1.2 Cost Effect Dataset 1 Condition 2

Condition-3:

- As shown in **figure 4.2.1.3**, results were similar to condition-2 with lesser number of iterations. Decrease in current rate being faster than concurrent decrease makes acceptance ratio of proposals even higher than condition-2. The results suggest that easy negotiation constraints certainly lead to less consumption of time.
- Least effective solution saves 13.6% over 21 iterations and most effective solution saves around 59.3% over 21 iterations. Average saving is 26.94% and average number of iterations is 20.40.
- Equally good results for lower cost values of C with lesser time consumption than **condition-2**. However, higher cost values of C showed some compromise in quality over time when compared to **condition-2**. Because of higher acceptance rate some existing solutions did not get reprocessed and missed further improvements.

Fig 4.2.1.3 Cost Effect Dataset 1 Condition 3

Condition-4:

- As shown in **figure 4.2.1.4**, similar results as condition-3 with lesser time for most of initial x values.
- Smaller number of iterations than former ones because of smaller modifiable range as very few proposals are reprocessed after generated
- Least effective solution saves 13.6% over 21 iterations and most effective solution saves around 59.3% over 21 iterations. Average saving is 26.87% and average iterations are 19.7. Having the higher acceptance rate than former conditions reduces the time consumption overall.

Fig 4.2.1.4 Cost Effect Dataset 1 Condition 4

Condition-5:

- As shown in **figure 4.2.1.5**, results were close to **condition-3** and **condition-4**, but a little less effective. .
- Least effective solution saves 13.76% of cost over 28 iterations and most effective solution saves around 59.3% of cost 21 iterations. Average saving is 26.7% with 21.75 iterations on average.
- Results implied that for optimal results negotiation criteria have to be in a balanced range without being extremely stringent or relaxed. Overall, results were more cost effective with stringent acceptance criteria for higher cost differences. For lower cost differences, lower acceptance criteria proved to be better.

Fig 4.2.1.5 Cost Effect Dataset 1 Condition 5

4.2.1.2. Dataset 2:

AgentA: (MPG, MPH)=(10, 50) (w1, w2)=(1, 1) coordinates=(0,-20)

AgentB: (MPG, MPH)=(40, 35) (w1, w2)=(1, 1) coordinates=(-15,15)

AgentC: (MPG, MPH)=(x, 50) (w1, w2)=(1, 1) coordinates=(15,15)

where $x = (2, 4, \dots 40)$

and $\text{cost}(\text{Agent}) = \text{distance} / \text{MPG} * w1 + \text{distance} / \text{MPH} * w2$

Results were similar in nature for all conditions with little variation in numeric values.

Here too, condition-1 stood out among all because of its criterion and we did not see much difference in results after the negotiation for that particular condition. Given below are general observations for different graphs for all conditions.

Unlike **Dataset-1**, A and B agents already had pretty close cost values here. C's cost starts from a higher one and gets closer to both A and B during the middle of graph and then later on C happens to have lowest travel cost. Because of the same, agent did not have that much difference in distances travelled or net payment, unlike before.

(1. Upper left) Total distance travelled by each agent: As cost of agent C decreases, distance travelled by him increase. For the same reason, distances travelled by A and B decreases. However, for a long time span in middle of graph, there are no negotiations among agents and all agents get to travel the equal distance. That situation occurs because of closeness of costs in all agents for that period.

(2. Upper right) Absolute final payment made by each agent: With reducing cost, agent C became more receiving toward incoming proposals. This resulted in a decrease in its net payment and increase in expensive agents' net payment. But, as mentioned for

previous graph, net payments by all agents are zeros during the middle of the graph when they have cost values very close to each other.

(3. Lower left) Number of iterations to find the best solution: Number of iterations can be loosely divided into three sections for all conditions in the given Dataset. First phase is during initial phase of graph, when C incurs a higher cost, second phase comes during middle part when there are no negotiations being made and last phase with C has lowest cost among all. For the phase when there are no negotiations, there is single iteration throughout as there is no solution. For the first phase number of iteration moves downward according cost effectiveness of solution. For the third phase it goes in upward direction then stays constant.

(4. Lower right) Total original cost to travel for all agents and total cost after finding the best proposal: For the period with no negotiation, total final and original costs are same. Apart from that, Difference among total costs is higher initially when C has higher costs compare to later stage. As with higher cost of travel during initial phases, C is able to negotiate more with other agents and over all cost also sees a reduction in that phase. During later phase with C being more expensive, differences among agents' cost are comparatively less, which results into lesser negotiation exchanges. Overall, results were more cost effective with stringent acceptance criteria for higher cost differences. For lower cost differences, lower acceptance criteria proved to be better.

Detailed descriptions of results for different negotiation conditions of agents are discussed further in this section.

Condition-1:

- As shown in **figure 4.2.1.6**, no saving was observed except for initial part when C had a very higher cost than A and B.
- Generated solution took very high time compare to other conditions with similar results for particular cost values.
- Least effective solution saves no money and most effective solution saves around 56.85% of cost with 65 iterations. Average saving is 2.84% where average number of iterations is 4.20.
- Results were similar to **condition-1** for **dataset-1**, which confirms the observation that having both acceptance and modification criteria too high results in failure of algorithm in delivering good solutions. Savings were lesser because agents had cost ranges comparatively closer to each other than dataset-1.

Fig 4.2.1.6 Cost Effect Dataset 2 Condition 1

Condition-2:

- As shown in **figure 4.2.1.7**, all phases except middle ones where C's cost was closer to B and C, had some negotiation exchanges.
- Among all conditions that actually produced some results, this one turned out to be most time consuming after **condition-1**.
- Least effective solution saves no money and most effective solution saves around 54.84% of total cost with 21 iterations. Average saving is 6.57% and average iterations are 15.1.
- Similar to previous dataset, results were overall better with negotiation criteria being loosened up, as it's denoted by higher average rate than condition-1.

Fig 4.2.1.7 Cost Effect Dataset 2 Condition 2

Condition-3:

- As shown in **figure 4.2.1.8**, similar results as condition-2 were observed with lesser number of iterations
- Decrease in current rate being faster than concurrent decrease in original rate makes acceptance rate high.
- Least effective solution saves no money and most effective solution saves around 55.16% of cost with 20 iterations. Average saving is 5.51% with average of 6.25 iterations.
- Results were less cost effective than condition-2 for higher MPG i.e. for lesser-cost values of C. Similar to previous dataset, higher acceptance rate caused lesser number of iterations and possibility of missing some improvement over existing accepted solutions.

Fig 4.2.1.8 Cost Effect Dataset 2 Condition 3

Condition-4:

- As shown in **figure 4.2.1.9**, results were pretty close to condition-2 but a slightly less cost effective at places where agents had close cost ranges.
- Least effective solution saves no money and most effective solution saves around 55.16% of cost with 20 iterations. Average saving was 5.36% with 5.5 iterations on average.
- Reason for above, Narrow range for modifiable proposals and lower acceptance criterion causes to skips improvement in few proposals. Hence, average saving was little lesser than consition-3.

Fig 4.2.1.9 Cost Effect Dataset 2 Condition 4

Condition-5:

- As shown in **figure 4.2.1.10**, proposals were significantly improved cost wise over previous conditions overall with increased profits and lesser duration without a solution.
- Least effective solution saves no money and most effective solution saves around 55.16% over 20 iterations. Average saving is 7.01% with 10.3 iterations on average.
- Average cost savings have concurrently decreased from condition-2 to condition-4, but condition-5 shows best average savings. Condition-5 having a wider modifiable range results in reprocessing more number of proposals. This produces better results when agents have their costs in close ranges or less scope of negotiation. The results are different than dataset-1 because dataset-1 has comparatively higher costs differences among agents.

Fig 4.2.1.10 Cost Effect Dataset 2 Condition 5

4.2.2 Effect of Distance on Results:

We prepared two data sets that had same initial values for location and cost. For first data set we changed agents A's location in a way that it kept going farther from both B and C with equal distance. For second dataset we changes location of B and C both in a way that they keep going farther from A with equal distance. Through these we observed the impact of distance on results of algorithm. Both datasets increased the distances in different manners. While former dataset making two objects closer than third one, later one increased overall distance among all agents. Since agents had different positions elated to each other in both datasets, generated results were also different in both the cases.

4.2.2.1. Dataset 1:

AgentA: (MPG, MPH)=(40, 50) (w1, w2)=(1, 1) coordinates=(0,y)

AgentB: (MPG, MPH)=(20, 35) (w1, w2)=(1, 1) coordinates=(-15,15)

AgentC: (MPG, MPH)=(10, 35) (w1, w2)=(1, 1) coordinates=(15,15)

where $y = (-20, -22, -24 \dots -60)$

and $\text{cost}(\text{Agent}) = \text{distance} / \text{MPG} * w1 + \text{distance} / \text{MPH} * w2$

Results shown below conclude the observation of one agent getting far away form initial position, which is A in our case. With A getting farther, negotiations from A's part or with A would be more difficult compare to other two agents. For this reason, we kept A's

cost very low compare to B and C. Following observations were made during algorithm run for this particular data set.

(1. Upper left) Total distance travelled by each agent: For condition-1 initially there were some negotiations and distances walked by agents were opposite of their cost values. After few distances there were no negotiations. Results were different for other conditions. C having maximum travel cost did not travel at all for most of the x values for all conditions. However, it was observed during condition-3, condition-4 and condition-5 that for higher values of A's distances from others, negotiations became more difficult. For this reason, C had to walk a little for those values even after having most expensive travel. Yet, final destination point in every solution was closer to C overall. Since A kept getting farther from C, his travel distance increased with highest rate with x. B had almost constant travel distance, as his distance with C never changed.

(2. Upper right) Absolute final payment made by each agent: Differences among net payment by agents kept increasing with increase in distance. Both A and B received equal amounts most of the time as C was the only payee in most of the negotiations. Though, for condition-1, net payments were little closer to each other but not equal for all agents as every agent had some contribution in negotiation.

(3. Lower left) Number of iterations to find the best solution: Increment in number of iterations with distance clearly states that finding a solution will take time proportional to

distance among objects. However, increment rate for iterations was different for different conditions for proposal acceptance.

(4. Lower right) Total original cost to travel for all agents and total cost after

finding the best proposal: Increments in total cost for best solution were not as steep as increments in original total cost. For this reason, difference in original total cost and final total cost kept increasing. It led us to the conclusion that quality of solutions increase with overall distance among all agents. Since with increased distances, agents had more scope to negotiate, final solutions were more cost effective than original solutions when compared to lesser distances. This proposition stayed true for condition-1 as well until graph reached to a point from where there were no negotiations being made.

Detailed descriptions of results for different negotiation conditions of agents are discussed further in this section.

Condition-1:

- As shown in **figure 4.2.2.1**, some solutions were generated during initial phase of graph but no solution was generated after distance values went little higher.
- Least effective solution saves no money and most effective solution saves around 18.52% over 20 iterations. Average saving is 0.92% and average iterations are 1.95. Even the most effective solution of this condition generates lesser benefits than least effective solutions of other conditions.
- Higher distances among agents leave more scope for negotiation. But, Having very high initial acceptance and modification criteria caused the algorithm to quit early. This resulted in missing proposals that would have turned into very beneficial solutions after modifications.

Fig 4.2.2.1 Distance Effect Dataset 1 Condition 2

Condition-2:

- As shown in **figure 4.2.2.2**, results were much more cost effective than **condition-1** for all x-values.
- Similar to previous datasets in section 4.2.1, among all conditions the actually produced results throughout, this one turned out to be most time consuming.
- Least effective solution saves 22.05% over 60 iterations and most effective solution saves around 40.04% of total cost with 87 iterations. Average saving is 32.4% with average number of iterations being 74.6%. Acceptance criteria being comparatively relaxed than **condition-1**, negotiations brought better solutions.

Fig 4.2.2.2 Distance Effect Dataset 1 Condition 2

Condition-3:

- As shown in **figure 4.2.2.3**, results were slightly less cost effective but highly time effective when compared to condition-2.
- Least effective solution saves 21.59% over 20 iterations and most effective solution saves around 39.8% of total cost over 63 iterations. Average saving is 32.33% where average number of iteration is 33.05. Average saving was very close to condition-2 with lesser than half time consumption.
- Having a higher acceptance and modification rate than condition-2 did cause missing slight improvement possibility for already accepted solutions but a little trade of in quality saved significant negotiation time.

Fig 4.2.2.3 Distance Effect Dataset 1 Condition 3

Condition-4:

- As shown in **figure 4.2.2.4**, negotiations took lesser time than condition-3 for higher distances because of the higher acceptance rate and narrower range for modifiable proposals.
- Least effective solution saves 21.63% over 20 iterations and most effective solution saves around 38.5% of total cost over 35 iterations. Average saving is 31.56% over 29.4 iterations on average. The difference from condition-2 was observed for higher distances because of the higher scope of negotiation there. Having a narrower modifiable range there missed some possible modifications through negotiation over solutions, which were already in accepted range.

Fig 4.2.2.4 Distance Effect Dataset 1 Condition 4

Condition-5:

- As shown in **figure 4.2.2.5**, results were slightly improved over **condition-4** for higher distance values and closer to **condition-3**.
- Least effective solution saves around 21.63% of total cost with 20 iterations and most effective solution saves around 39.16% of cost with 51 iterations. Average saving is 32.31% with average of 32.6 iteration.
- Since modification range was wider than condition-4, solutions for higher distances had liberty of going through further negotiations for improvement. For the same reason, time consumption was higher than condition-4 and close to condition-3.

Fig 4.2.2.5 Distance Effect Dataset 1 Condition 5

4.2.2.2. Dataset 2:

Results below are plotted for different coordinate locations of A and C denoted as (x1, y1) and (x2, y2). Locations of A and C are shifted vertically on Y-axis and horizontally on negative and positive X-axis respectively. Using the results we examined the effect of increasing the distance among all agents, on total cost and effectiveness of solutions.

AgentA: (MPG, MPH) =(20, 10) (w1, w2)=(1, 1) coordinates=(-15, 15)

AgentB: (MPG, MPH) =(20, 40) (w1, w2)=(1, 1) coordinates=(0, y)

AgentC: (MPG, MPH) =(20,50) (w1, w2)=(1, 1) coordinates=(x, 15)

Where:

where y = (-20, -22, -24 .. -60)

and x = (15, 17, 19, ----, 45)

and $\text{cost}(\text{Agent}) = \text{distance} / \text{MPG} * w1 + \text{distance} / \text{MPH} * w2$

Results shown below work differently than previous dataset. Relative positions of agents with each other would play a crucial role here. In previous dataset, two agents being close to each other had advantage from each other's negotiations, compared to third agent.

Hence, for higher distances, negotiations were practically between two sides rather than all agents, which resulted in easier and more effective negotiations than current dataset.

With current dataset, distance increased evenly for all agents. Therefore, differences between original and final solutions were not as high as previous ones.

Each result set includes following observations:

(1. Upper left) Total distance travelled by each agent: Two observations were made for this part:

(1) Difference in distance travelled by agents kept increasing with distance.

(2) Agents with highest cost walked even lesser when total distance travelled were higher.

Point 1 and 2 together suggest that more solutions were generated and accepted respectively with increased distance. Since, agents with higher costs were able to negotiate more with increased distances, their travel distance did not increase.

(2. Upper right) Absolute final payment made by each agent: Differences among net payment by agents kept increasing with distance. Similar to chapter 4.2.2.1 - dataset-1, scope for negotiation increased among agents for higher distances. For this reason, agents with higher cost paid more to less expensive agents.

(3. Lower left) Number of iterations to find the best solution: Increment in number of iterations with distance clearly states that finding a solution will take time proportional to distance among objects. However, increment rate for iterations was different for different conditions for proposal acceptance.

(4. Lower right) Total original cost to travel for all agents and total cost after finding the best proposal: Increments in total cost for best solution was not as sharp as increments in original total cost. For this reason, difference in original total cost and total cost for best solution kept increasing. It led us to the conclusion that quality of solutions increase with overall distance increasing among all agents.

Above propositions stood true for most of the conditions except condition-1. For that case no proposal got finalized for any distance.

Detailed descriptions of results for different negotiation conditions of agents are discussed further in this section.

Condition-1:

- As shown in **figure 4.2.2.6**, no saving observed for any distance value.
- With very high criterion for accepting proposals, no proposal got accepted for given condition.
- Average saving and average number of iterations for the proposal are zero. This indicates that for given cost and distances, acceptance criteria imposed by condition-1 is too high to produce any solution.

Fig 4.2.2.6 Distance Effect Dataset 2 Condition 1

Condition-2:

- As shown in **figure 4.2.2.7**, acceptance criteria being comparatively relaxed than **condition-1**, after few initial distance values, solutions started being generated here.
- Among all conditions the actually produced some results, this one turned out to be most time consuming.
- Least effective solution saves no money and most effective solution saves around 23.6% of total cost with 134 iterations. Average saving is 16.22% with 79.75 iterations on average. Similar to all other datasets, here too condition-2 proved to be a little too stringent to produce cost and time effective solutions when compared to other conditions except condition-1.

Fig 4.2.2.7 Distance Effect Dataset 2 Condition 2

Condition-3:

- As shown in **figure 4.2.2.8**, results were close to condition-2 with far lesser number of iterations and slightly more cost effective.
- Least effective solution saves no money and most effective solution saves around 23.8% of total cost with 46 iterations. Average saving is 16.46% with average 34.15 iterations. Higher acceptance and modification rate resulted in lesser number of iterations with no loss of quality compared to condition-2.
- If we generalize the observation from all datasets presented here, condition-3 delivers consistently cost effective results with time consumption being significantly lesser than previous and not too higher than further conditions.

Fig 4.2.2.8 Distance Effect Dataset 2 Condition 3

Condition-4:

- As shown in **figure 4.2.2.9**, provides little lesser cost effective results than condition-3 for few initial distance values. Narrow range for modifiable proposals and lower acceptance criterion causes to skips improvement in some proposals unlike former conditions.
- Least effective solution saves no money and most effective solution saves around 23.8% of total cost over 46 iterations. Average saving is 16.34% with 27.05 iterations on average. Similar to all other datasets, smaller number of iterations than former ones was observed. As due to the smaller modifiable range, very few proposals are reprocessed after generation.

Fig 4.2.2.9 Distance Effect Dataset 2 Condition 4

Condition-5

- As shown in **figure 4.2.2.9**, proposals were significantly improved cost wise over previous conditions, especially for the first half of distance values. However, Number of iterations was higher compare to **condition-3** and **condition-4**.
- Least effective solution saves no money and most effective solution saves around 23.8% of total cost over 46 iterations. Average saving is 17.28% with 34.2 iterations on average. A higher average of saving than previous conditions complies with following observation that was made previously.
- Keeping a large range for modifiable solution can help in bringing out better results, especially in initial cases when agents needed small changes to generate solutions. As it results in contributions from multiple agents to improve the proposal by further negotiations while trading off some time for quality.

Fig 4.2.2.10 Distance Effect Dataset 2 Condition 5

Chapter 5

Conclusion and Future Work:

5.1 Conclusion:

Although, algorithms for bilateral negotiations have been successfully developed before, our algorithm focuses on situations when one negotiation involves more than two parties in the process. Proposed strategy emulates stock trading in a free market. The basic strategy in our algorithm of initiating a negotiation for an agent is to offer an equal share in its profit in exchange of service from other agents. The results from our experiments reveal that while idea of proposing the profit-share is standard for all agents, effectiveness of algorithm lies in agents' strategy for evaluating the proposals. Our experiments performed for various costs and distances in **chapter-4** led us to some generalized conclusion for agents' negotiation strategy. Rather than only accepting the perfect solutions, if agents keep their acceptance benchmark comparatively liberal, desired solutions can be generated using less efficient solutions through collaborative inputs. Our algorithm compels the agents to modify existing solutions from other agents that are nearly efficient, before generating their own new solutions. This strategy improved the quality of results in all cases regardless of costs and distances. Especially, when agents were in close proximity of costs with each other, their collaborations made significant difference. Overall cost effectiveness of solutions became notably better when

agents extended their consideration for modification of each other's proposals for further improvement. However, based on specific requirements and constraints of a problem for instance time constraints or size of solution space, acceptance criteria can be decided according to what works best in given situation. Among all of the proposed conditions in chapter 4.1, different conditions might give best time, best profit or better consistency. For example condition-4 always consumed least amount of time over other conditions, but condition-5 provided best profits when minute changes were affecting the solution space. To add further, Condition-2 had best possible solution in most cases but always consumed too much of time, where condition-3 never delivered best results in any case but performed consistently close to best-solutions strategy in terms of profit and cost both. Therefore, according to need of a specific problem and experimental results, best strategy can vary from case to case.

As it's mentioned before, agents in our algorithm have no explicit cost information about each other. Still, they achieve the optimal solution by working on each other's proposals for improvement during negotiation. The constraint of incomplete information among agents emulate the real world situation as in many cases agents either don't want to share much of their constraints and preferences or they don't have communication to share this information, especially when individual preferences and constraint are likely to change over time. Through our algorithm, our attempt is to introduce a method for problem solving in a distributed system that stays generic for all agents with different personal interests and constraints. Different agents in the system might have varying cost factors but their deals are generated and finalized through same algorithm. Our approach ensures

uniformity of profit distribution among all agents while meeting everyone's personal criteria without including a centralized broker.

Our strategy can be particularly useful in environment where goal is to establish equilibrium among different subsets of system by providing one solution. Some real time examples are mentioned below to explain the practical usage of our algorithm.

- In a computer network, our algorithm can provide useful solutions for efficient router placement among different data lines by applying negotiation mechanism in end systems.
- A very close real time example of our strategy can be seen in transportation planning. Optimal meeting points in transportation networks can be obtained through proposed negotiation algorithm.
- Furthermore, any distributed system such as a computer or robotic cluster can utilize our algorithm for resource sharing or task allocations to perform the computing process efficiently.

Next section discusses the future applications and possibilities of improvement in our approach.

5.2 Future Work:

At present, our algorithm applies equitable trading among agents for generating profits. However, the fact that agents in the distributed systems have varying individual interests can be utilized to improve the negotiation speed, increasing individual and/or overall benefits. Mentioned below are few suggestions for future improvement that we could envision for the given approach.

- An agent can be enabled to learn about other agent's preferences by observing the proposals they accepted or rejected. Using the self-learning, it can change its profit sharing strategies accordingly. This provision can increase the rate of negotiation acceptance and speed up the process to reach an optimum solution.
- Negotiation protocol can be extended further to allow different cliques of agents in a large group to negotiate among them or allowing different agents to form coalitions and create proposal considering common benefits. This provision can cover wider range of problems and might reach the solution faster than current algorithm.
- Usually, since agents have different individual costs, successful solutions are similar in nature. High-cost agents share their profits in exchange of services from agents. If agents could be enabled to self-learn about their contribution or cost differences, an agent can decide for it-self to continue or stop generating proposal in order to minimize the time consumption from ineffective proposals in the process.

While creating the proposal some other external concerns are not considered that algorithm need to face during real time implementation. For example, cost for broadcasting message to all agents, cost dependency of communication over physical distance between agents are some of the practical issues that might affect the efficiency of algorithm. However, environmental factors in themselves are a large and altogether different area to cover. In this particular research, we disregard other external hazards and concentrate on proposing a non-centralized approach for problem solving in a distributed system while agents are dealing with incomplete set of information. We have aimed to establish a protocol in which agents work collaboratively over their negotiations in order to figure out the solution that meets their individual interests.

Bibliography:

- [1] S. Kraus “Negotiation and cooperation in multi-agent environment” Artificial Intelligence, volume 94, 1997, Pages: 79 - 97
- [2] S. S. Fatima, M. Wooldridge, N. R. Jennings “An Agenda-Based Framework for Multi-Issue Negotiation” Artificial Intelligence, 2004, volume 152, pages: 1–45
- [3] Z. Zhang, H. Ghenniwa, W. Shen “A Framework for Adaptive Negotiation in Multi-Agent Systems” 11th International Conference on Computer Supported Cooperative Work in Design, 2007, Pages: 406 - 411
- [4] S. Aknine, S. Pinson, M. F. Shakun “An Extended Multi-Agent Negotiation Protocol” Autonomous Agents and Multi-Agent Systems, 2004, pages: 5–45,
- [5] Z. A. Bian, J. Luo “A Cooperative Game Model for Agent Negotiation in Network Service” 10th International Conference on Computer Supported Cooperative Work in Design , 2006, Pages: 1 - 6
- [6] R. P. McAfee “A Dominant Strategy Double Auction”, Journal of Economic Theory, 1992, volume 56, Pages: 434-450
- [7] X. Su, S. Chan, G. Peng “Auction in Multi-Path Multi-Hop Routing” 2009, Volume 13, Pages: 154-156
- [8] M. Yokoo, Y. Sakurai, S. Matsubara “Robust Double Auction Protocol against False-name bids” Distributed Computing Systems, 2001, Pages: 137 - 145
- [9] R. G. Smith “The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver” Computers, IEEE Transactions on, 1980, Volume: C-29, Pages: 1104 - 1113

- [10] T. Sandholm “An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations” Proceedings of the eleventh national conference on Artificial intelligence, 1993, Pages: 256-262
- [11] S. Kraus “Automated Negotiation and Decision Making in Multiagent Environments” Multi-agents systems and applications, 2001, Pages: 150 - 172
- [12] S. Kraus, K. Sycara, A. Evenchik “Reaching agreements through argumentation: a logical model and implementation.” Artificial Intelligence, 1998, volume 104, Pages: 1–69,
- [13] H. Luo, X. J. Hu “Multi Agent Negotiation Model for Distributed Task Allocation” Information Management and Engineering (ICIME), 2010, Pages: 54 - 57
- [14] T. Sandholm “Agents in electronic commerce: Component technologies for automated negotiation and coalition formation” Multi Agent Systems, 1998, Pages: 10- 11
- [15] O. Shehory, S. Kraus “Feasible formation of stable coalitions among autonomous agents in non-super-additive environments.” Computational Intelligence, 1999, volume 15, pages: 218–251
- [16] T. W. Sandholm, V. R. Lesser “Coalition formation among computationally bounded agents.” Artificial Intelligence, 1997, volume 94, pages: 99-137
- [17] O. Shehory, S. Kraus “Methods for task allocation via agent coalition formation.” Artificial Intelligence, 1998 volume 101, pages: 165–200
- [18] <http://www.gametheory.net/about/FAQ.html>