UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF
NATURAL & APPLIED SCIENCES

HEURISTIC OPTIMIZATION THROUGH NEGOTIATION

INDUSTRIAL ENGINEERING
PhD THESIS

ZEYNEP DİDEM UNUTMAZ DURMUŞOĞLU
JUNE 2012

Heuristic Optimization through Negotiation

PhD Thesis
in
Industrial Engineering
University of Gaziantep

Supervisor
Prof. Dr. Adil Baykasoğlu

by
Zeynep Didem UNUTMAZ DURMUŞOĞLU
June 2012

T.C.
UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF
NATURAL & APPLIED SCIENCES
NAME OF THE DEPARTMENT

Name of the thesis: Heuristic Optimization through Negotiation
Name of the student: Zeynep Didem UNUTMAZ DURMUŞOĞLU
Exam date: 11.06.2012

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Ramazan KOÇ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Türkay DERELİ
Head of Department

This is to certify that we have read this thesis and that in our consensus/majority opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Adil BAYKASOĞLU
Supervisor

Examining Committee Members	signature
Title and Name-surname Prof. Dr. Adil BAYKASOĞLU	
Prof. Dr. Türkay DERELİ	
Prof. Dr. Hadi GÖKÇEN	
Prof. Dr. Rızvan EROL	
Prof. Dr. Lale Canan Dülger	

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Zeynep Didem UNUTMAZ DURMUŞOĞLU

ABSTRACT

HEURISTIC OPTIMIZATION THROUGH NEGOTIATION

UNUTMAZ DURMUŞOĞLU, Zeynep Didem
PhD in Industrial Engineering
Supervisor: Prof. Dr. Adil BAYKASOĞLU
June 2012,157 pages

Finding realistic and express solutions to several problems has been a fundamental requirement in this rapidly changing conditions and environment. Conventional approaches, which are solving dynamic problems, "as they are static" or reinventing their models after each corresponding change, have all expired. In this respect, it has been understood that, the most effective solution strategies have been the agent-based strategies. These strategies, which are mostly constructed upon a heuristic, are capable to provide parallel and distributed solutions and furthermore they let agents to take autonomous decisions that can add value to the objectives of the problem. This PhD thesis aims both 1) setting up a representation scheme for agent-based approaches and 2) providing solutions to dynamic variants of Travelling Salesman Problem (TSP), which has been solved by static approaches up to now.

Classical TSP consists of "n number of cities" where, each city is visited once using an optimal route. Similarly, Generalized Travelling Salesman Problem (GTSP) covers "c number clusters" where, each cluster is visited exactly once by visiting one of cities of the clusters using an optimal route. However, in some specific cases, assumptions of classical TSP and GTSP may not be satisfactorily enough to reflect physical reality and dynamism of actual systems. During solution of those problems, some cities are added to city domain or some disappear from the system. Thereby, the assumption of keeping the number of cities constant fails. Consequently, "a solution provided for a specific to state defined for a time" may lose its superiority immediately after these unexpected changes. In this respect, these dynamic types of TSP and GTSP may not be modeled with the conventional research methods since much time and memory spaces are required to setup novel models and solve them repeatedly. With those considerations in mind, this thesis covers different agent-based solution policies/strategies for providing promising solutions to different problems in domain of dynamic TSP. In this respect, two main solution strategies are proposed for the solution of the defined problem types in this PhD thesis. One of these is competition of agents without a heuristic and the other is competition of agents using Great Deluge Algorithm (GDA). All those strategies have proved themselves competed with other findings in literature.

Key Words: Dynamic Travelling Salesman Problem (DTSP), Great Deluge Algorithm (GDA), agent-based modelling.

ÖZ

MÜZAKERE ARACILIĞIYLA SEZGİSEL ENİYİLEME

UNUTMAZ DURMUŞOĞLU, Zeynep Didem
Doktora Tezi, Endüstri Müh. Bölümü
Tez Yöneticisi: Prof. Dr. Adil BAYKASOĞLU
Haziran 2012, 157 sayfa

Hızla değişen koşullar ve çevre, dinamik problemlerin gerçekçi ve hızlı şekilde çözümünü gerekli kılmıştır. Dinamik problemleri; statik problemler gibi ele almak veya her değişim sonrasında yeniden model kurmak ve çözüm için uzun süreler beklemek gibi yaklaşımlar geçerliliğini yitirmeye başlamıştır. Bu bağlamda en etkili çözüm yaklaşımlarının; paralel ve dağıtık çözümler yapabilen, otonom kararlarla genel çözüme katkıda bulunabilen ve genellikle sezgisel yöntemleri içeren etmen tabanlı çözümler olduğu anlaşılmıştır. Bu bağlamda bu tez ile amaçlanan; 1) Gerek etmen tabanlı çözüm stratejileri için ortak bir gösterim şemasının oluşturulması 2) Gerekse bugüne kadar statik çözümler aranan Gezgin Satıcı Problemi'nin (GSP), dinamik türevlerine etmen tabanlı etkin çözümler bulunmasını içermektedir.

Klasik GSP; her şehir bir kez ziyaret edilecek şekilde "n sayıdaki şehrin" en uygun rota kullanılarak dolaşılmasını içermektedir. Benzer şekilde; klasik Genelleştirilmiş Gezgin Satıcı Problemi (GGSP) de "c sayıdaki bölgenin" her bir bölgesinden en az bir şehir ziyaret edilecek şekilde en uygun rota kullanılarak dolaşılmasıdır. Ancak bazı özel durumlarda, klasik GSP ve GGSP'nin sabit şehir ve/veya bölge varsayımları sistemlerin fiziki gerçekliğini ve dinamikliğini yansıtmada tatmin edici nitelikte değildir. Problem çözümü esnasında, şehir listesine yeni şehirler eklenebilmekte veya bazı şehirler sistemden ayrılabilmektedir. Bu nedenle, gerçek sistemlerde, şehir sayısını sabit tutma varsayımı geçerliliğini yitirmektedir. Netice olarak, "belirli bir an veya durum için bulunmuş sonuç" beklenilmeyen değişiklikler nedeniyle geçerliliğini yitirebilmektedir. Bu bağlamda dinamik GSP ve dinamik GGSP'nin geleneksel yöntemlerle modellenmesi oluşacak maliyet ve hafıza nedeniyle uygun olmayacaktır. Bu sebeplerden dolayı; bu tez kapsamında, dinamik GSP ve türevleri olan çeşitli problemler değişik ajan tabanlı stratejileri/politikaları ile çözülmektedir. Tanımlanan problemlerin çözümü için etmen tabanlı iki temel strateji sunulmaktadır. Bu stratejilerden biri; ajanların sezgisel kullanmadan; diğeri ise Büyük Kara Delik (BKD) sezgiselini kullanarak birbirleriyle rekabet etmesini içermektedir.

Anahtar Kelimeler: Dinamik Gezgin Satıcı Problemi (DTSP), Büyük Kara Delik Algoritması (BKD), etmen tabanlı modelleme.

ACKNOWLEDGEMENTS

My studies through this PhD research, has been an informative journey for me and it has been a remarkable part of my both personal and academic experience. I have had great opportunity to work with many interesting, intelligent and helpful people. I would like to thank all the people who have accompanied me and supported me.

First, I would like to express my sincere thanks to my supervisor Professor Dr. Adil BAYKASOĞLU for his time, guidance, invaluable feedback and vital comments throughout my PhD research studies. I am also thankful to him for assigning me to one of his most vital and exciting research studies. I am also very grateful to our department chair Professor Dr. Türkay DERELİ. He provided motivation and both personal and academic support several times through my studies.

I would like to address my special thanks to my husband, Alptekin DURMUŞOĞLU. He delivered heartfelt support and unfailing patience during my doctoral study. My mother, my brother, my brother's wife and their little daughter Nevin Eda UNUTMAZ, have also provided motivation for me. Therefore, I should appreciate their limitless helps and endless love.

I should not also forget Professor Wayne Wakeland for his guidance during my research visit to Portland State University.

Finally, I would like to dedicate this dissertation to my dear father whom we lost very early. He loved me and thought me to be honest and respectful to my job.

CONTENTS

LIST OF FIGURES

page

LIST OF TABLES

page

LIST OF SYMBOLS / ABBREVIATIONS

DVRP	Dynamic Vehicle Routing Problem
ABDOPSS	Agent-Based Dynamic Optimization Problem Solution
ABMS	Agent-Based Modeling and Simulation
APSO	Agent-Based Particle Swarm Optimization
CmOPs	Combinatorial Optimization Problems
CSP	Constraint Satisfaction Problems
DBSCAN	Density Based Spatial Clustering of Applications with Noise
DOPs	Dynamic Optimization Problems
DPOP	Distributed Pseudotree Optimization Procedure
DTSP	Dynamic Travelling Salesman Problem
EAs	Evolutionary Algorithms Problems
GTSP	Generalized Travelling Salesman Problem
MAEA-CSP	Multi-Agent Evolutionary Algorithm for Constraint Satisfaction Problems
MAGA	Multi-Agent Genetic Algorithm
MBA	Market Based Algorithm
OR	Operations Research
PSO	Particle Swarm Optimization
RFQ	Request for Quotes
SA	Simulated Annealing
SBDO	Support Based Distributed Optimization
SGA	Standard Genetic Algorithm
SOA	System Optimal Agent
TS	Tabu Search
TSP	Travelling Salesman Problem
VE	Variable Elimination

CHAPTER 1

INTRODUCTION

1.1. General Remarks

Dynamism is an attempt to clarify the phenomena of the universe against to several instant changes. All scientists dealing with the systems and phenomena of the universe have unsurprisingly faced with a variety of different change. Therefore, they are usually obligated to ignore more than one variable changing simultaneously. However, with the storm of global change, it has been difficult to understand the increasing nature of dynamism and the uncertainty with the existing approaches. Providentially, new computing and programming utilities like agent-based technologies have enabled more realistic modeling abilities. In this respect, a typical operations research problem including constraints, objective functions and variables could have been altered with the more realistic ones where constraints, objective functions and domains of variables can also be a matter of any kind of change at any time.

To cope with such dynamism, there have been several efforts to adapt some of the meta-heuristics to work in a harmony and in integrity without ignoring the objective(s) of modeling. This usually requires agent-based approaches letting the elements (such as ants, bees or gens) to communicate and negotiate with each other in order to adapt themselves with respect to the changes in domain of variables, constraints and objective functions.

In parallel to those changes, several difficult problems have been solved through use of agent-based modeling. However, some dynamic problems still have not been

studied with agents. Therefore, it is not wrong to state that, there is still a wide research space in the area.

This PhD thesis, also attempts fulfill two vacancies in the area. One is about the all agent-based applications that have lied in a wide range of area. Although some dynamic optimization problems have common features, there is not a systematic scheme that represents similarities and differences of those applied solution strategies. Therefore, a researcher studying and focusing on a specific DOP may have many troubles to classify existing studies and their solution approaches. Systematic observations on several practical applications in the area have inspired a solution regarding the development of such a scheme that is capable to indicate features of a problem and solution strategy that is followed. Several sample problems were also used to test usability of this scheme.

The second vacancy that this PhD thesis focuses is application of agent-based modeling approaches to dynamic travelling salesman problem where number of cities can immediately change. In this respect several new variant of TSP are introduced and solved via different agent-based strategies. Promising solution have be obtained and statistically analyzed.

Details of the proposed solution approaches that are regarding both of the described vacancies in the area can be found in next section.

1.2 Thesis Lay-Out and Organization

In the light of the declarations given in the previous section, this research study within this PhD thesis focuses on agent-based modeling and dynamic optimization problems.

In this respect, in Chapter 2, DOPs that are solved via agent-based modeling are discussed with details. This chapter attempts to be a warm-up chapter for the people who are not familiar with DOPs, agent-based modeling and the fundamental concepts of them.

In Chapter 3, a new representation scheme called ABDOPSS, which is constructed for standard representation and classification of agent-based solution approaches employed for DOPs, is presented. ABDOPSS is also exemplified to present different applications in the area.

Since there are numerous software packages that are widely used for agent-based modeling it has been a necessity to introduce them. Therefore Chapter 4 is devoted to those software packages.

Chapter 5 presents five different types of Dynamic Travelling Salesman Problems (DTSP) that are highly dynamic due to random change in number of cities. In those defined problems, new cities arrive to the system and sometimes they disappear from it. In this respect, two main agent-based solution strategies are proposed for the solution of the defined problem. One is competition of agents without a heuristic and the other is competition of agents using Great Deluge Algorithm (GDA).

Chapter 6 finally discusses the possible benefits and handicaps of the solutions provided by this PhD study. Future study extensions for new students and the people studying in the area, is also declared in that final chapter.

1.3 Concluding Remarks

The proposed solutions in this PhD thesis are all novel to the literature. They have been presented in several conferences and published in several different journals. These solutions are also expected to be useful in the industrial area. It is also anticipated that, some ideas and implementations presented in this thesis can create new research directions for other researchers.

The contribution of this thesis can be summarized in two folds:

1) The classification scheme (ABDOPPS: Agent Based Dynamic Optimization Problem Solution Strategy) for agent-based approaches is new to the literature and it is expected to be beneficial to researchers in many ways. Similarities of the features

located in ABDOPPS can be used to define classes of solution strategies by their descriptions. In this regard, classes of the problems may orient researchers to focus on certain strategies. Using the dynamism related features of the corresponding DOPs presented in ABDOPPS, unpredictability levels of certain problems can be determined and be used to reclassify problems. These representation forms can also be used to discover the role of presented features and their importance for solution quality.

2) Thesis is the provision and extension of knowledge on how to analyze and design new computational models for travelling salesman integrating the city addition and deletion at any time during the visits.

CHAPTER 2

LITERATURE REVIEW ON DYNAMIC OPTIMIZATION PROBLEMS

2.1 Introduction

Tendencies such as the increasing spread of market globalization, new technological developments, the reduction of product life cycles and aggressive competition, are generating high levels of environmental changes and uncertainty for organizations of all types (Volberda, 1997 and Sanchez, 1997). In this regard, in today's competitive environment, managers in various industries face with complex problems with the increasing uncertainty where they are not familiar too much. Therefore, resource scarcity, increasing costs, shorter response times are the main factors which creates necessities for novel solution approaches for those complex problems.

In this perspective, operations research as being a scientific approach has already attempted to deal with these complex problems by providing several techniques and approaches. On the other hand, most of the researchers studying operations research (OR) have just focused on the well-known traditional optimization techniques like linear programming, integer programming and ignored the novel approaches which are capable of dealing with much more uncertainty. Although, these traditional techniques and the approaches guarantee the optimality, their applicability for the real life problems have been limited in some cases. There have been numerous examples of traditional OR techniques applied in the literature, which has many impractical assumptions letting some factors to be accepted as constant. Those assumptions like limiting the number of variables, handling the problem as static etc… could only provide partial solutions for these problems. However, most of the real world problems are large-sized, complex and thereby it is not realistic to keep

some factors as constant. In addition to that, in most of the real cases, solving such much difficult problems with classical optimization techniques is almost impossible.

On the other hand, in the last decade, various heuristic techniques have evolved that facilitate solving optimization problems, which were previously difficult or impossible to solve. It is known that, heuristic techniques are the solution approaches, which mimic the successful strategies found in nature. Solving the complex problems with heuristic optimization techniques has offered two main advantages for solutions: speed and size of instances that can be handled. Although, several benefits are obtained with the heuristics optimization, some of the difficulties are not still released. As well known, most of the real-world industrial problems are dynamic and they are all subject to the several changes, unfortunately, most of the heuristic optimization techniques treat these problems as being static.

As stated by Allmendinger and J. Knowles (2011), problems that are subject to changes, belongs to the class of dynamic optimization problems. Therefore, many real world problems should be handled as dynamic optimization problem (DOP) to have realistic modeling. DOPs are defined as "*optimization problems whose optimal solution changes over time during the optimization, which could result from change of environmental parameters, change of constraints, change of objectives and change of problem settings (representations)*" by Yaochu (2004). As stated by vandenBergh and Engelbrecht (2004), there has been a growing research interest on the resolution of Dynamic Optimization Problems (DOPs), due to its closeness to real-world situations.

Nevertheless, addressing DOPs has been a difficult task, since it is necessary to find new techniques\approaches\algorithms, which can handle the problem correctly and adapt to changes easily. As indicated by Yan et al. (2010) for DOPs, the goal of an algorithm is no longer to find an optimal solution but to track the moving optima in the search space. In this respect, classical heuristic optimization techniques need to be adapted to DOPs. With the advances in computer (i.e. hardware and software) and communication technology, researchers have been searching for new paradigms such as multi-agent systems in order to solve distributed and\or DOPs. In this regard, reusable agents have provided many opportunities, they saved researchers from

moving beyond reinventing, representing, and re-implementing the problem, and thereby they have decreased the cost of providing solutions (Neches et al., 1991). Agent-based modeling and simulation (ABMS) has also been a relatively new approach to modeling complex systems, composed of interacting, autonomous 'agents' (Macal and North, 2010).

Multi-agent system, also called 'self-organized system' is a computational system in which multiple interacting intelligent agents work together to solve difficult problems, which may be impossible for an individual agent (Yan et al., 2010). In order words, multi-agent systems are computational systems in which several agents interact or work together to achieve some purpose (Liu et al., 2010). Therefore, cooperation and communication have been the most important features of agent-based approach for solving DOPs.

As anticipated, agent based system has generated lots of excitement in recent environments since it has been suggested as a promising technique for conceptualizing and solving various optimization problems (Yan et al., 2010).

There have been also some studies that employed classical optimization techniques with agent-based approach in the literature. Persson (2005) compared the agent approaches with classical optimization techniques and concluded that classical optimization techniques and agent-based approaches are the complements of each other. In another study, performed by Jian (2003), evolutionary soft agents were used to solve integer programming. There are also some researchers who have used meta-heuristic techniques with agent-based systems in order to solve complex problems. For example, Xu and Liu, (2006) and Ahmad et al., (2007) combined the Particle Swarm Optimization (PSO) algorithm with multi-agent based approach.

All of the mentioned approaches defined above for DOPs can be applied for a wide-range of application area. The problems, which have been attempted to be solved by multi-agent systems, are expected to contain high uncertainty, unpredictability and thereby high dynamism. Although, some problems like scheduling problem, production planning problem and travelling salesman problem have been previously solved with the classical OR techniques, indeed these problems are not fully static

problems due to the nature. Consider for example scheduling and production-planning problems where new machines with advanced capacities are required to be included to the systems or some machines are required to be removed from the system frequently and/or new products are frequently introduced to the system etc. This challenging dynamism is apparently a technology management problem and it can only be effectively handled by using more advanced approaches like multi-agents. In this regard, dynamism in technological environments requires the technology-managers to be knowledgeable on techniques, which can be used to model and optimize dynamic systems more effectively.

Although meta-heuristic approaches for the solution of dynamic optimization problems are relatively novel to the literature, in this chapter it is intended to review and analyze existing studies that is available in the literature.

2.2 Literature Review

Agent based computing has often been suggested as a promising technique for problem domains that are distributed, complex and heterogeneous (Persson, 2005). As it is known, DOPs are complex in nature. Beside this, many real-world problems are not only dynamic but also distributed. Tsui and Liu, (2003) indicated that distributed problem solving by a multi-agent system represents a promising approach to solving complex computational problems.

Nowadays, in order to solve DOPs, researchers have been seeking for cooperative strategies. The natural correspondence between autonomous entities and meta-heuristics and problem solving with an optimization problem, paves the way for the development of nature-inspired cooperative strategies for optimization (Pelta et al, 2009). Pelta et al. (2009) stated that, the fundamental part of cooperative strategies for optimization is the use of a population of elements which may be solutions to the problem at hand, “gens”, “antigens”, “ants”, ”particles” etc... As indicated by Pelta et al. (2009), when those elements used adequately, they form the basis of successful meta-heuristics for problem solving.

The summary of the articles, which are handled for this part of thesis, is presented at Table 2.1. In order to solve dynamic and distributed optimization problems, Wangermann and Stengel, (1999) studied the principled negotiation, which effectively coordinates distributed optimization in multi-agent systems. Generally, in principled negotiation, agents proposed options for mutual gain. It was implemented, unless other agents disagree to the proposal. However, under certain conditions an agent could search for options for individual gain without affecting other agents. In these cases, the agent could negotiate with a coordinator, rather than obtain an agreement from all other agents. Researchers pointed out the theory of principled negotiation and represented it mathematically. In order to answer the following questions, two examples were presented. "*How good is the action profile developed by principled negotiation compared to that created by centralized system*?" "*What is the effect of agent knowledge on negotiation*?" and "*How does principled negotiation perform in constrained and unconstrained situations*?" In the first example, there was no coupling between the agent actions if coordination criteria were met. The second one was highly coupled, constraining each agent's available set of actions. It was based on the air-traffic management problem of negotiating arrival slots. Results revealed that principled negotiation worked well in both constrained and unconstrained situations.

Researchers indicated that the optimization performance of multi-agent systems using principled negotiation was as good as that of a centralized system that used the same optimization method and perfect knowledge about other agents. It was concluded that, principled negotiation offers great advantages for Aircraft/Airspace systems since it would increase the freedom of system users to optimize their operations while maintaining safety.

In the literature, there are many papers, which combined the multi-agent approach with heuristic techniques such as Tabu Search (TS), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), etc… in order to solve the dynamic problems more effectively.

For example, Shigehiro et al. (2002) considered a new optimization method for combinatorial optimization problems based on many autonomous agents. The

proposed method was a multi-point search method, where many agents explore in the solution space concurrently. These agents carry out local search method and the action of each agent can be controlled by software (algorithm), not by parameters. Therefore, the process of optimization can be controlled directly. In this study, two kinds of agents namely; "searcher agent" and "manager agent" were employed. These agents worked cooperatively. Many searcher agents explored in the solution space by means of local search method and several manager agents controlled searcher agents. Two or more neighbors were given and each manager agent corresponded to each neighbor. Each searcher agent applied the neighbor corresponding to the manager agent by which the searcher agent was controlled. In order to improve searcher agents, two operations were applied: the searcher agent was transferred to another manager agent. The position of the searcher agent in the solution space was not changed. The searcher agent was transported to another solution. The position of the searcher agent in the solution space was moved. The best solution in all solutions stored in searcher agents, who were under control of the manager agent was searched. Then, the searcher agent was moved to a solution "near" the best solution searched.

The proposed method has been programmed in Ruby language. Each searcher agent runs on separate "ruby thread". The proposed method applied to a TSP "HOKKAIDO62" with 62 cities. The Simulated Annealing (SA) method has been also programmed and applied to the same problem. In the early stage of the exploration, researchers observed that the cost by the proposed method was much better than the one by SA method. In the later stage of the exploration, it was monitored that costs of two methods were almost same. Results revealed that ability of proposed method was not inferior to one of the SA method.

Xu and Liu, (2006) presented *a multi-agent based particle swarm optimization framework HMAS.* The main idea is to add intelligence in the effective searching of traditional particle swarm optimization (PSO) technique. In this framework, particles were represented as agents and a swarm was composed of agents. Each agent had the following features; flying actions, self-locating action, communication, learning action, local search space analysis, autonomous self-population maintenance and life cycle. Thanks to these features, an agent would add intelligence into the swarm.

Agents form agent groups called subpopulations. A mechanism was assigned to each sub-population, which was implemented by an agent. These agents formed one of the two models: the master-slave model and the island model. In the master- slave model, one special agent has been defined as master agent. Each subpopulation has been assigned a slave agent.

The master agent has supervised the whole population assigned tasks and operations. In addition, the master agent has chosen good individuals from the whole population and migrate the seeds to all subpopulations. However, in the island model, one agent has been assigned to each sub-population. Agent has supervised the evolution process of its sub-population. Sub-populations have exchanged good individuals as a certain time interval mutually. A test application in cluster analysis was given. Based on the results of this application, the researchers indicated that the multi-agent based particle-swarm optimization framework was a promising new searching model.

Ref.	Strategy	Agent Behavior	The problem
(Wangermann et al., 1999)	Principled Negotiation	Agents propose options for mutual gain. In some cases, an agent could search for individual gain. In individual gain cases, the agent negotiates with a coordinator.	Aircraft\Airspace traffic systems
(Shigehiro et al., 2002)	Autonomous agents with simulated annealing	Agents explore in the solution space concurrently. Searcher agents use local search method and the action of each agent is controlled by an algorithm. The best solution in all solutions stored in searcher agents, which are under control of the manager agent.	Travelling sales person problems
(Xu and Liu, 2006)	Multi-agent based particle swarm optimization	*Master slave model:* One agent is defined as master agent. Each subpopulation is assigned to a slave agent. The master agent supervises the whole population assigned tasks and operations. In addition, the master agent chooses good individuals from the population and migrate the seeds to all subpopulations. *Island model:* One agent is assigned to each sub-population. Agent supervises the evolution process of its sub-population. Sub-populations have exchanged good individuals as a certain time interval mutually.	Cluster analysis
(Ahmad et al., 2007)	Combination of *Particle* Swarm Optimization technique with multi agent system	The environment is modeled as an agent for providing information about the problem space to the agents. A Boolean value is attached with each point to identify whether that point has been already visited by an agent. As in the original PSO, particles search for the optimal solution by traveling through the problem space in a multi-dimensional environment.	N/A
(Persson, 2005)	Comparison of agent based approaches with optimization techniques.	For the agent-based approach it is assumed that control is distributed and concurrent. For optimization techniques, researchers focused on methods using a central node which has the entire responsibility of computing the optimal (near optimal) solution to the problem.	Dynamic distributed resource allocation
(Zhong et al., 2004)	Multi-agent systems and genetic algorithms	A candidate solution to the optimization problem in hand is represented by an agent. All agents lived in a lattice like environment, with each agent fixed on a lattice-point. Agents competed or cooperated with their neighbors and used their knowledge in order to increase their energies.	Numerical optimization problems
(Liu et al., 2006)	Multi-agent evolutionary algorithm	According to characteristics of each problem type, several behaviors are defined for agents in order to give the ability of agents to sense and act to the environment. The behaviors are controlled by evolution. The minimum conflict encoding was employed to eliminate the shortcomings of the general encoding methods.	Constraint satisfaction problems
(Liu et al., 2010)	Multi-agent systems integrated with evolutionary algorithms	Competition behavior and self-learning behaviors are defined for minimizing the objective function value. These behaviors are controlled evolution so that the agent lattice can evolve in each generation. At each generation, the competitive behavior is first performed by agents. This behavior cleaned the agents with low energy from the lattice so there was more space developed for the agents with high energy. After this process, self-learning behavior is performed iteratively, until the stopping conditions were satisfied.	Combinatorial optimization problems
(Pelta et al, 2009)	Multi-agent decentralized strategy	A population of cooperative agents moved over a grid containing solutions. The basic decentralized optimization strategy used is based on the joint of two populations. Cooperation through the environment, peer to peer cooperation, mixed strategy mechanisms is used. Two mechanisms are also employed for diversity. First, the grid of solutions is per se an implicit diversity mechanism. Second, a specific diversification mechanism in two stages, which are perturbation of solutions and random repositioning.	Different configurations of the moving peaks benchmark problems
(Hadeli et al., 2004)	Multi-agent system with indirect communication	The coordination between agents is done by using food foraging mechanism of an ant. Thereby, information is spread and global information is made available locally. Intentions of the agents are forecasted in short-term.	Flexible manufacturing system
(Boughaci and Drias, 2005)	An evaluator agent based on taboo search	First of all a plan is generated randomly. The desired number of tasks and relation precedence are created. In addition, different values for duration are specified by customer agent. Then bids are generated for random sets of contiguous tasks within the request for quotes (RFQ) by the vendor agent. Finally, bids are evaluated. Once the bidding deadline is past, the customer agent evaluates the sets of bids received. The evaluation based on Taboo Search and the goal is to find a combination of bids that provides coverage of all tasks and minimizes costs and risks.	Complex bid (bundled bid) with scheduling constraints

Table 2.1- Summary of the articles listed in the literature

Another study that combined the PSO with agent-based approach was carried out by Ahmad et al., (2007). In their paper, researchers proposed agent-based Particle Swarm Optimization (APSO), which combined the optimization technique with the feature set of a multi agent system. Similar to Xu and Liu (2006), particle has been termed as an agent. By introducing autonomy and learning to PSO, particles become more intelligent and autonomous and could achieve performance that is more effective and use of resource. In addition to particle, the environment was modeled as an agent and was responsible for providing extra information about the problem space to the agents. A Boolean value was attached with each point in the problem space to identify whether an agent has already visited that point. As in the original PSO, particles search for the optimal solution by traveling through the problem space in a multi-dimensional environment. Based on this methodology, the points in the problem space were being continuously tagged by particles after they visit them for the first time therefore the environment was transformed *from a static one to a dynamic version*. The environment agent applied a density based cluster algorithm (DBSCAN-Density Based Spatial Clustering of Applications with Noise) to discover clusters of the tagged points. Those clusters could then be utilized by an agent to areas in the problem space that it could possibly away. If a particle agent has arrived at a visited point, which meant this point has been evaluated before and not found optimal, the agent could skip the fitness evaluation and request its neighbors' positions and cluster information to update its own position. Besides tagging points in problem space, each particle inquire the current best solution from its neighbors to calculate the global best location, return its personal best solution to its neighbors, and to request information regarding the surrounding clusters from the environment agents. The algorithm terminated either maximum iteration was reached or minimum error criteria was fulfilled. This technique not only reduces the chance that a particle agent visits non-optimal points in the problem space but also accelerates the optimization process by preventing unnecessary fitness evaluation.

Differently from Xu and Liu, (2006) and Ahmad et al., (2007), Persson (2005) compared the strengths and weakness of agent based approaches and classical optimization techniques. They evaluated their appropriateness for a special class of resource allocation problems, namely dynamic distributed resource allocation. The purpose was to find hybrid approaches, which capitalize on the strengths of two

approaches. In the class of problems studied, information and\or resources are distributed and the exact conditions; e.g. the demand and availability of resources are not known in advance and changing. They compared the two approaches with respect to how they were able to handle some important properties of the problem domain. For the agent-based approach, it was assumed that the control was distributed and concurrent. On the other hand, for the optimization techniques, researchers focused on methods using a central node which has the entire responsibility of computing the optimal (or near optimal) solution\allocation to the problem. Further, they focused on methods that have the potential to provide solutions of guaranteed good quality.

According to the preliminary analysis, they stated that agent based approaches tend to be preferable when; the size of the problem is large, communication and computational stability is low, the time scale of domain is short, the domain is modular in nature, the structure of the domain changes frequently (i.e., high changeability), there is sensitive information that should be kept locally; and classical optimization techniques when: the cost of communication is high, the domain is monolithic in nature, the quality of solution is important, it is important that the quality of the solution can be guaranteed. They pointed out that, the properties of agent-based approaches and optimization techniques complement each other. Further, they investigated two hybrid approaches and particularly focused on aspects related to time scale, quality of solution and cost of communication. One of them was "using an optimization technique for coarse planning and agents for operational re-planning" (i.e., for performing local adjustments of the initial plan in real time to handle the actual conditions when the plan is executed), other was "embedded optimization in an agent. A case study, which concerned the planning and allocation of resource in combined production and transportation, was examined. The case study was based on a real world case within the food industry. The problem was to determine how much to produce and how much to send to different customers each time. The problem is dynamic since one often has to plan based on forecast, which may be rather uncertain; and there are uncertainties associated with the availability of resources. Four different approaches namely; *Pure Agent Approach*, *Embedded Optimization*, *Pure Optimization* and *Tactical\Operational Hybrid Approach*, were tested. The results showed that, when the problem size increases significantly, guaranteed optimal solutions to the formulations are unlikely to be

obtained within reasonable time using of-the shelf optimization software. The result for the approaches *Pure Agent and Pure Optimization* are compatible with the results of the theoretical analysis with respect to time, quality and communication properties. Further, by comparing objective values between approach *Pure Agent, Embedded Optimization* and *Tactical\Operational Hybrid Approach*, the results indicated that adding optimization to agents, improves the agents' decision making with respect to the quality of solution. Beside this, in *Tactical\Operational Hybrid Approach* and *Pure Optimization* approaches the quality of the decisions might depend on what time period that is considered.

Zhong et al., (2004) integrated multi-agent systems and genetic algorithms to construct a new algorithm, which is named as multi-agent genetic algorithm (MAGA). This algorithm was employed for solving the global numerical optimization problem. In MAGA a candidate solution, to the optimization problem in hand is represented by an agent. All agents lived in a lattice like environment, with each agent fixed on a lattice-point. Agents competed or cooperated with their neighbors and used their knowledge in order to increase their energies. By using the agent-agent interactions, MAGA tried to minimize the value of objective function. In theoretical analysis, researchers observed that MAGA converged to the global optimum. For the first part of the experiment, ten benchmark functions were selected in order to test the performance of MAGA. It was revealed that MAGA accomplished good performances when the dimensions of the problem rose from 20 to 10,000. In addition to this, MAGA could find the high quality solutions with low computational cost when the dimensions were as high as 10,000. This result proved that MAGA has good scalability and is a competent algorithm for solving large-dimensional optimization problems. Researchers stated that, this paper was the first study, which optimized the functions with 10,000 dimensions by means of evolution. Further, MAGA was applied for the approximation of linear systems as a practical case. MAGA also gave satisfactory results for this case.

Following the work of Zhong et al., (2004), Liu et al., (2006) considered the constraint satisfaction problems (CSP). They divided the problem into two types which were named as permutation CSPs and non-permutation CSPs. According to characteristics of each type, several behaviors were defined for agents in order to

give the ability of agents to sense and act to the environment. These behaviors were controlled by evolution, which results Multi-Agent Evolutionary Algorithm for Constraint Satisfaction Problems (MAEA-CSPs). The minimum conflict encoding was employed to eliminate the shortcomings of the general encoding methods. It was observed by the theoretical analysis that, MAEA-CSP had a linear space complexity and converged to the global optimum. For the first part of the problem 250 benchmark binary CSPs and 79 graph coloring problems were employed in order to analysis the performance of the proposed algorithm for non-permutation CSPs. For comparison purpose, six well-defined algorithms were selected and the effect of parameters analyzed systematically. For the second part of the experiment, the performance of MAEA-CSPs for permutation CSPs was tested by using a classical CSP, n-queue problems, and more practical case job shop scheduling problems (JSPs). For both n-queue problems and JSPs, MAEA-CSPs were demonstrated good performance.

Subsequent to Zhong et al., (2004) and Liu et al., (2006), Liu et al., (2010) integrated multi-agent systems and Evolutionary Algorithms (EAs) in order to solve Combinatorial Optimization Problems (CmOPs). A new algorithm, which was called as multi-agent EA for CmOPs (MAEA-CmOPs), was proposed. Two agent behaviors, namely competition behavior and self-learning behavior, were defined for minimizing the objective function value. These behaviors were controlled by means of evolution so that the agent lattice could evolve generation by generation. At each generation, each agent first performed the competitive behavior. This behavior cleaned the agents with low energy from the lattice so there was more space developed for the agents with high energy. After this process, self-learning behavior was performed iteratively, until the stopping conditions were satisfied. In the proposed algorithm, each agent could only sense its local environment so its behaviors can only take place between it and its neighbors. Therefore, global selection was not required. By transferring the information between neighbors, the information was diffused to the whole agent lattice. Researchers indicated that, the evolutionary mechanism based on the agent lattice used in MAEA-CmOPs was closer to the real evolutionary mechanism in nature than that based on the population model used in traditional EAs.

The common point between this study and the work in Zhong et al., (2004) and Liu et al., (2006) was to combine multi-agent systems with EAs. However, each of them was proposed to solve different type of problems because of this; different types of agent behaviors were employed. Researchers stated the core of each algorithm was the agent behaviors, which showed that three algorithms were completely different both implementation and application fields. In addition, each work was tested by using benchmark problems in each field. To validate the performance of the MAEA-CmOPs, deceptive problems with strong linkage, overlapping linkage and hierarchical tree problems with tree like structures were employed. Similar to the result of previous studies (i.e. (Zhong et al., 2004) and (Liu et al., 2006)), by theoretical analysis, it was revealed that MAEA-CmOPs converged to global solutions. In contrast to EAs which has the slow convergence property, MAEA-CmOPs showed fast convergence rate and obtained a good performance even for large-dimensions hierarchical problems. Researchers pointed out that, MAEA-CmOP obtained a polynomial time complexity for all test problems and the parameters of MAEA-CmOPs were simple and easy to be tuned. They concluded that MAEA-CmOPs was a competent algorithm for practical applications.

Pelta et al, (2009) presented a multi-agent decentralized strategy (MADCOS) for dynamic optimization problems. Researchers aimed to test and analyze different communication schemes for the agents in MADCOS and to test and analyze the real need of an explicit diversity preserving mechanism in MADCOS. In this strategy, a population of cooperative agents moved over a grid containing solutions. Different configurations of the moving peaks benchmark problems were employed in order to test the performance of MADCOS. They focused on cooperation and diversity mechanisms, and they studied how different alternatives affect the performance of the strategy. The basic decentralized optimization strategy used in this paper was based on the joint of two populations. Three basic mechanisms, namely cooperation through the environment, peer-to-peer cooperation, mixed strategy, were used. In order to handle the diversity two mechanisms were employed. First, the grid of solutions was per se an implicit diversity mechanism. Second, a specific and simple diversification mechanism in two stages, which were perturbation of solutions and random repositioning, was implemented. A set of computational experiments were done in order to asses: how different communication strategies affect the search; and

the usefulness of having explicit and implicit diversity mechanisms. Accuracy and error were selected in order to measure the performance. Results showed that having an adaptive probability of cooperation should lead to better results than the ones achieved there. Researchers indicated that regarding diversity, the implicit mechanism provided by their multi-agent model seems to be enough for the scenarios tested. However, it was stated that the explicit mechanism proposed, based on randomization, did not perform as expected.

Hadeli et al., (2004) studied the concept of multi-agent system where agent used an indirect communication mechanism, called stigmergy. The design used in this paper aimed at handling changes and distributions. The coordination between agents was achieved by using a technique, which was used by food foraging ant. This mechanism ensured that the information was spread and global information was made available locally. In order to determine whether it was possible to create short-term forecasts based on the intentions of the agents, a prototype implementation of this design for manufacturing control system was realized. The prototype involved a flexible manufacturing system model, which had dynamic order arrival, probabilistic processing time, and some general perturbations such as machine breakdowns. Results showed that it was possible to generate and use short-term forecasts based on intentions of the order agents in relatively simple systems by using stigmergic approach. It was also revealed that it was possible to build a system in which individual agents had limited exposure, permitting them to operate in a wide range of situations and conditions. In other words, the proposed manufacturing control system adequately reacts to change and disturbance. By using the coordination and control mechanism, the congestion in the manufacturing plant could be handled easily.

Boughaci and Drias (2005) proposed an evaluator agent; based on taboo search methodology. The work considered complex bid (bundled bid) with scheduling constraints. In particular, the bid specified a set of tasks (items), a price and resources availability data that included task durations and early and late start limits. Different from the previous works, where classical methods such as linear or dynamic programming were employed to solve bid evaluation for small set of bids, this study focused on meta-heuristics. Researchers stated that this work was the first one dealing with taboo search based meta-heuristic for bid evaluation in e-commerce.

Researchers tried to find an assignment of plan tasks to bids including the task that provides coverage of all tasks, minimize a cost and risk and allows for feasible schedule. In order to generate a problem for implementation of the proposed methodology, first a plan was generated randomly. The desired number of tasks was generated and relation precedence was created. In addition, different values for duration were specified by customer agent. Then bids were generated for random sets of contiguous tasks within the request for quotes (RFQ) by the vendor agent. Finally, bids were evaluated. Once the bidding deadline was past, the customer agent evaluates the sets of bids received. The evaluation based on Taboo Search meta-heuristic and the goal was to find a combination of bids that provides coverage of all tasks and minimizes costs and risks.

This implementation was performed by using JAVA programming language. Researchers indicated the scientific contribution of this study as: proposition of an agent treating the bid evaluation using taboo search, conception of the clients\server model using such agent, utilization of a randomization strategy to create a combination of bids, proposition of context-dependent moves to create neighborhood combinations, adding a feasibility strategy to the taboo search algorithm to ensure the temporal constraints of all the tasks., and adding a convergence test into taboo search algorithm ensuring convergence of all the tasks to achieve the overall goal.

2. 3 Concluding Remarks

In this chapter, previous studies that employed agent-based approaches to solve dynamic optimization problems are investigated. As it was indicated before, solving DOPs are more difficult than solving static problems since optimal solution changes over time in DOPs. Therefore, the techniques used for this type of problems should be adapted to the moving optima.

As stated by Montoro et al. (2007) nowadays, multi-agent paradigm has become one of the most active research fields in computer science. The communication\negotiation and coordination abilities of agent-based systems are very useful for handling DOPs.

In the next chapter, more research papers in the literature are presented and these studies are classified by answering some questions such as “Is the number of decision variable changes among time”, “Is the search space change with time?” “Is the structure of the objective function dynamic?” “Is the problem distributed or not?”…etc. In addition, previous studies will be classified according to type of study such as approach, model, application etc...

CHAPTER 3
CLASSIFICATION OF DYNAMIC OPTIMIZATION PROBLEMS

3.1 Statement of Purpose

Optimization efforts through system modeling have been one of the most convenient approaches employed by many researchers to obtain promising solutions for several problems. Nonetheless, optimizing the dynamic systems enclosing continuous change including the change in objective function(s), constraint/restriction(s) and parameter(s) has been a difficult task for the researchers (Baykasoğlu and U. Durmuşoğlu, 2012).

On the other hand, some of the researchers studying Operations Research (OR) have attempted to solve these problems by using the well-known traditional optimization techniques like linear programming and integer programming (Baykasoğlu and U. Durmuşoğlu, 2011). However, solutions obtained for a certain time that is desirable or optimal for that time slot, often have not be preferable for another time slot. Furthermore, obtaining exact solutions using mathematical methods for each time slot has been unaffordable or even sometimes has not ended-up with feasible solutions. Therefore, notwithstanding its widespread use, the static optimization approach has increasingly approached its limits (Borst et al., 2005). This fact certainly inspired many researchers to employ agent-based modeling for the optimization of dynamic systems. As indicated by Jung et al. (2011), the agent paradigm has been shown to be a promising approach to develop intelligent, heterogeneous, and open systems, because of its features, such as *autonomy*, *flexibility*, and *cooperative problem solving behavior* (Jung et al., 2011). Agent-based models contain the agent(s), which are capable of behaving in an adaptive, flexible and autonomous manner in accordance with the objectives defined. In this regard, reusable agents have provided many opportunities, they saved researchers

from moving beyond reinventing, representing, and re-implementing the problem, and thereby the cost of providing solutions has been decreased (Neches et al., 1991).

Agent-based models addressing Dynamic Optimization Problems (DOPs) has also allowed several optimization mechanisms to track the moving optima in the search space. This fact certainly inspired many researchers to employ agent-based modeling for the optimization of dynamic systems (Baykasoğlu and U. Durmuşoğlu, 2012).

Even though, there have been several papers in the literature employing agent-based modeling approach to provide significant solutions for DOPs, each of these existing studies employs different agent-based modeling approaches having different strategies and agent features. Therefore, currently, we do not have a common successful methodology for the design of agents (Corchado et al., 2008). With dependability and safety in mind, it is vital that the mechanisms for representing and implementing agents are clear and consistent (Fisher et al., 2007). In this regard, it is certain that there is a vacancy in the standard representation and classification of agent-based solution approaches employed for DOPs.

A previously developed scheme (Calégari et al., 1999) for evolutionary algorithms for combinatorial optimization has also inspired us to prepare a standard representation format. In this perspective, one of the objectives of this chapter is to present the fundamental features/ingredients of DOPs solved by agent-based modeling. The features positioned in a standard scheme provide an opportunity to present DOPs in a standard way. It can succeed further and in future studies, these representation forms can be used to study the role of these features and their importance for solution quality. Thereby, this chapter provides a basis to analyze the best ways of implementing agent-based approaches to the DOPs. In this perspective, this chapter presents an attempt to develop a notion for DOPs solved by agent-based approach. It also shows how solution strategies with agent-based approach for DOPs can be summarized in a concise manner by informing about the agents employed and the key elements of DOP's. A classification scheme is introduced and presented in a tabular form called Agent Based Dynamic Optimization Problem Solution Strategy (ABDOPSS). ABDOPSS distinguishes between different classes of agent-based algorithms (via communication type, cooperation type, dynamism domain and etc.)

by enumerating the fundamental ingredients of each of these agent-based approaches. At the end, possible uses of the ABDOPSS are illustrated and exemplified for some agent-based approaches.

This chapter is organized as follows. Section 3.2 and Section 3.3 introduces the features listed as the ingredients of ABDOPSS. Section 3.2 specifically focuses on the features of typical DOPs. Subsequently, Section 3.3 introduces the generic features of agent-based approaches. In Section 3.4, some typical agent-based solution approaches that are implemented for DOPs are described by using the ABDOPSS. Section 3.5 illustrates the application of ABDOPSS for 18 relevant articles collected from the literature. Finally, Section 3.6 presents concluding remarks with the possible benefits of the proposed classification scheme.

3.2 Problem Specific Features: A Typical Dynamic Optimization Problem

A dynamic problem is considered to be a problem that changes some or all of its characteristics in time (Lung and Dumitrescu, 2009). Thereby, optimization problems that are subject to changes, belongs to the class of dynamic optimization problems (DOPs) (Allmendinger and Knowles, 2010). In other words; DOPs are "*optimization problems whose optimal solution changes over time during the optimization, which could result from change of environmental parameters, change of constraints, change of objectives and change of problem settings (representations)*" (Jin, 2004).

All of the changes that are mentioned above take place in the dynamic environment and these changes create unpredictability even chaos in some situations. These changes can be opportunities improving the solutions or sometimes they can be threats decreasing level of desirability of a solution (Baykasoğlu and U. Durmuşoğlu, 2012).

There are different approaches in the literature that is classifying these changes in the dynamic environments. Yang (2007) defines environmental change as: cyclic and random. In cyclic change of environments, with the time going, an old environment

reappears exactly (Yang, 2007). A cyclic change covers the same fixed length sequence of contraction and/or growth is repeated over time (Abbass et al., 2004). The pattern repeats itself indefinitely (Abbass et al., 2004). However, in random change, there is no information about the structure of change. Random chance is also called as acyclic and in random change there are no repeated patterns over the period; that is, there is an indefinite random sequence of contractions, growths, and random changes (Abbass et al., 2004).

All of these changes in dynamic environment cause significant changes in model's structure, objective function, restrictions/constraints, parameters, and in the decision variables. Bui et al. (2011) describe three of these changes through the examples as presented below.

1- Time-varying objective functions: For example, enemy units arrive at a location, making some parts of the objective more difficult. The objective function is not constant over time. Therefore, the objective value of a solution can be different at different times. This usually causes the occurrence of new optima (Bui et al. 2011).

2- Time-varying variables: An example problem of this category is dynamic machine scheduling where unexpected new jobs arrive. A time-varying variable can be used for determination of the objective value, but can be a late addition or change (Bui et al. 2011).

3- Time-varying constraints: For example the precedence relationship of tasks. The objective function and variables do not change for this category; however, the constraints may change over time. This category of change does not change the fitness landscape driven by the objective function, but it will affect the areas of feasibility. It is interesting to note that this category of change has not been analyzed in detail in comparison to the other two categories (Bui et al. 2011).

Cruz et al. (2010) provides a valuable survey of research done on optimization in dynamic environments over the past decade. In addition to the content of their paper, they provide a web site (www.dynamicoptimization.org) and present characteristics of the DOPs with respect to objective function or the restrictions change with time.

We have considered the study of Abbass et al. (2004) to illustrate correspondence between the change types and the changes in the modeling components. Abbass et al. (2004) defines six common kinds of changes for a typical system having certain data flow structure. These changes are change in the model, change in the number of concepts, change in the number of features, change in the level of noise, change in the class distribution, and change in the sample bias (Baykasoğlu and U. Durmuşoğlu, 2012).

Table 3.1 both summarizes the change states defined by Abbass et al. (2004) and their corresponding change in a typical mathematical model which is similar to the one defined by Cruz et al. (2010). Examples of each change categories have also been provided through the possible changes in a dynamic production facility. According to the example case, a typical production facility that is operating in a dynamic environment faces with several expected and unexpected changes with the above given change states (Baykasoğlu and U. Durmuşoğlu, 2012).

Change 1- Change in the model: Change in objectives of a system may lead a significant "change in the model". Think of the production facility trying to minimize the cycle time and adjusting all of its production strategy with respect to cycle time minimization. If the cost of tardy job increases significantly and the managers will be obligated to change their objective as the minimization of the number of tardy jobs. In this case, objective function, constraints and the number of decision variables along with the type of parameters may change.

Change 2- Change in the number of concepts: Sometimes, concept classes may not be as clear-cut as initially thought so that additional classes may arise or old classes may vanish over time (Abbass et al., 2004). In case of change in the number of concepts, resources used by the system can increase or decrease. New machines can arrive to the system or some machines can be removed from the system because of the oldness or in functionality. This change can affect the model by changing the variables used in objective function and constraints. This type of change is not expected to create structural change. It is a kind of reconsideration of the model with added/removed variables that is significant for decisions.

Change 3- Change in the number of features: In some problems, the number of available features characterizing a problem instance may vary over time. Additional information may become available or new tools may be developed that allow more accurate classification (Abbass et al., 2004). Orders received for new product types are a typical example of this change. In case of such a change, production facility may face with new operations required and this can lead changes in objective function, constraints and the number of decision variables.

Change 4- Change in the level of noise: The noisiness of the data may change as well (Abbass et al., 2004). This type of change is particularly common when dealing with data coming from sensors. For example, acoustic data collected in an open area can have different noise levels based on the state of the environment (Abbass et al. 2004). In this respect, in the production facility example, if some customers broke the deal and cancel their orders for a production facility, the plans unfortunately changes and it may cause change in objective function and constraints.

Change 5- Change in the class distribution: Abbass et al. (2004) exemplifies the change in the class distribution with SARS virus. When the SARS outbreak took place, the class distribution changed almost every day with more healthy people becoming infected effectively increasing the proportion of positive cases. Similar to SARS example, in case of an economic boom, arrival of some certain jobs to the system can be more frequent and this leads change in parameters.

Change 6- Change in the sample bias: Since system parameters like processing times for machines are estimated by the sampling of several actual processes, an unexpected sampling error can end up with the change in actual processing times and/or system arrival times.

All those changes defined above also directly effects the unpredictability of the problem domain. Level of unpredictability is vital to determine a proper agent-based solution strategy. Although there are several efforts to match the corresponding changes with the unpredictability, they are still in their infancy. Therefore, there is a

serious need for measuring the level of unpredictability of the DOPs (Baykasoğlu and U. Durmuşoğlu, 2012).

The relation between the corresponding changes in constraints, objective function and the both with uncertainty and dynamism has been previously presented by (Cruz et al. 2010) in the web site (www.dynamicoptimization.org) as shown in Figure 3.1. While their framework as shown in Figure 3.1 does not cover the variability in parameters and variables, it can still be a good reference for future studies. Considering parameter variability and variability in the number of decision variables will certainly increase the complexity of such an analysis (Baykasoğlu and U. Durmuşoğlu, 2012).

Table 3.1 Change types for a dynamical environment and their corresponding affects in the mathematical model (Baykasoğlu and U. Durmuşoğlu, 2012)

Change State	Changes defined by (Abbass et al. 2004)	Corresponding change in the model	An example from dynamic production facility
Change 1	Change in the model	Objective function change, Constraint change, Number of decision variables, Type of parameters	Objective of the system can be altered from minimizing the cycle time to minimizing number of tardy jobs
Change 2	Change in the number of concepts	Objective function change, Constraint change, Number of decision variables	New machines can arrive to the system or some machines can be removed from the system.
Change 3	Change in the number of features	Constraint change, Objective function change, Number of decision variables	New product orders can arrive to the system requiring different operations
Change 4	Change in the level of noise	Objective function change, Constraint change	Some customers can broke the deal and cancel their orders
Change 5	Change in the class distribution	Parameter change	In case of economic boom, arrival of jobs to the system can be more frequent
Change 6	Change in the sample bias	Parameter change	Since processing times for machines are estimated using sampling, an unexpected sampling error can end up with the change in actual processing times

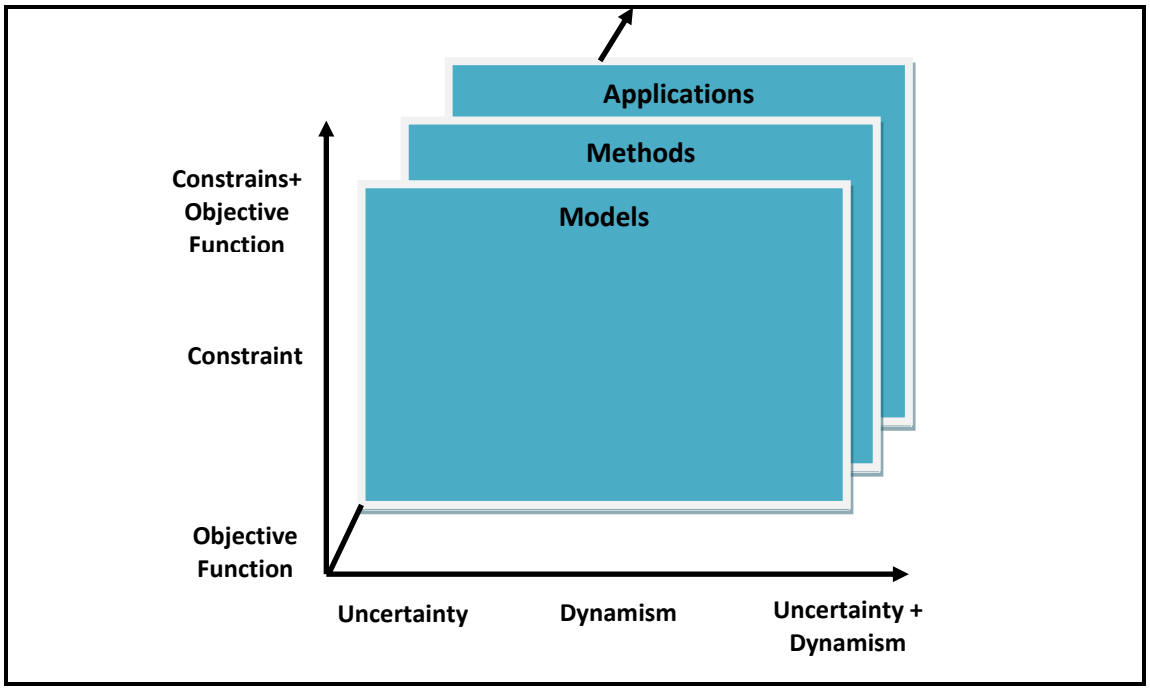

Figure 3.1 Dynamic and uncertainty characteristics of a Dynamic Optimization Problems (www.dynamicoptimization.org)

It is also remarkable that dynamism is one of the components of environmental uncertainty. Indeed, as presented in Figure 3.2, environmental uncertainty has three dimensions: dynamism, heterogeneity, and hostility (Newkirk and Lederer, 2006). Dynamism, as being the one dimension of the environmental change, (Teo and King, 1997) found to have two dimensions. The dimensions are referred to here as changeability (i.e., concerned with the rate of obsolescence and of technology change) and unpredictability (i.e., concerned with competitors' moves and demand changes). The rigor of that study and relative recency of the findings motivated the use of those two dimensions in the current research (Newkirk and Lederer, 2006). In this regard, the above-defined six different changes cover different level of changeability and unpredictability. Categorizing changes into these two dimensions appears as a future research opportunity to perform (Baykasoğlu and U. Durmuşoğlu, 2012).

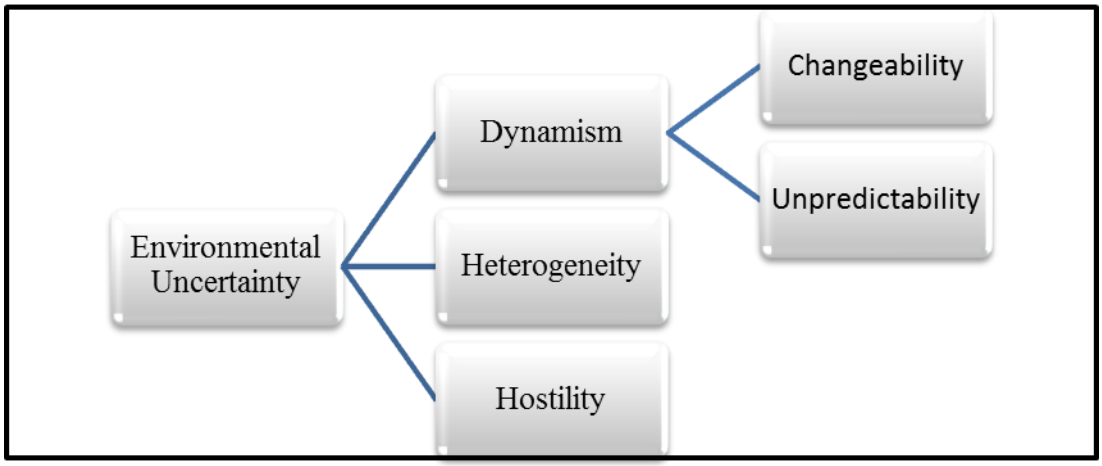

Figure 3.2 Type of environmental uncertainty (Baykasoğlu and U. Durmuşoğlu, 2012)

Different DOPs tried to be solved in the literature for different scopes by using various approaches. Some of the existing studies on DOPs via agent-based modeling propose only frameworks some other studies also present applications. It is remarkable to state that a framework is not a detailed hypothesis or set of hypotheses; rather, it is a suggested point of view for an attack on a scientific problem, often suggesting testable hypotheses (Aggestam and Söderström, 2005). Therefore, studies covering solely the frameworks do not provide comparable outputs. Some of the applications cover real-life instances and some have hypothetical data obtained via simulation. Some of the studies attempting solve DOPs by using agent-based modeling use test problems that has been previously presented in the literature. Thereby, they compare their findings with the best-known results in the literature. Some other studies compare the results with themselves by providing different methodologies to solve the problems. Examples of these cases will be discussed in the applications section (Baykasoğlu and U. Durmuşoğlu, 2012).

3.3 Solutions for DOPs by ABS

Although some problems like scheduling, production planning and travelling salesman have been previously solved as they are static problems, indeed these problems are not totally static problems due to the nature of the real life. Consider an example of scheduling and production planning problems where new machines with advanced capacities are required to be included to the systems or some machines are required to be removed from the system and/or new products are can also be

introduced to the system etc. Another example is for a more real life oriented travelling salesman problem (TSP) has been defined by (Homayounfar et al., 2003). They defined three different types of changes for TSP:

* Changing the distances (time) between the cities (due traffic congestion, road repair etc.)
* Changing the number of the cities (i.e. adding or deleting some cities)
* Swapping the cities

A distance (e.g. the distance between the first and second cities) is changed after a specific time or specific generation, determined by Dynamic Frequency. The amount of change is referred to as the dynamic rate. These two parameters are set prior to the test. After each period of time, which is after a generation specified by Dynamic Frequency, dynamic rate is added to the specified cost value (i.e. distance between city 1 and 2). This shifts the peak (optimum cost) to another location, so a new global solution can be found (Homayounfar et al., 2003).

Numerous examples can be considered for real-life oriented versions of these well-known problems. The distinction between these conventional problems and real-life oriented ones can be performed as follows. If during the solving of an optimization problem, parameters of the system (i.e. objective and constraint functions) do not change due to nature of the problem then optimization is static, otherwise it is considered to be dynamic (Homayounfar et al,. 2003).

In such environments, optimization process can never be terminated. In the real world, a function to be optimized may vary from time to time and the optima have to be found in time (Guan et al., 2005). Therefore, it is very difficult to optimize such type of dynamic optimization problems with conventional methods, which are developed to deal with non-stationary environments. Moreover, it is also quite hard to define dependencies or correlations between the solutions obtained at a time slot and in another time slot. As stated by O'Hare et al. (2008), the real world is both unpredictable and unforgiving. Decisions often need to be made where the contributory evidence is uncertain, incomplete, contradictory and highly dynamic

(O'Hare et al., 2008). Therefore, performing design of experiment may not end up with significant findings.

Since finding a feasible solution to the static problem is NP-hard, we must make certain assumptions about the input. Intuitively, if the input is such that even finding a static solution is hard we cannot expect to find a good solution with respect to the dynamic objective function. Thus, the problem instance must be "easy enough" that a relatively straightforward agent can find a feasible solution at each time step. In practical terms, this means there must be enough resources to satisfy easily the demand if we ignore the quality of the solution in the sense of the dynamic objective function. In the worst case, we can fall back on the heuristic to find a feasible solution. In fact, agent-based algorithm will degrade gradually to this extreme, but should perform much better in the common case. This challenging dynamism was apparently a significant problem for researchers and it could only be effectively handled by using more advanced approaches like agent-based modeling (Baykasoğlu and U. Durmuşoğlu, 2012).

3.4 Agents and their features

Although there is no universal agreement in the literature on the precise definition of agent, their property of being autonomous has been common to all definitions (Kulkarni and Tai, 2010). An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives (Jennings et al., 2001).

Agents that act in an autonomous fashion to perform delegated tasks have aroused much attention over the last decade (Liu et al., 2006) which is named as intelligent agents. An intelligent agent performs interactive tasks tailored to a user's needs without humans or other agents telling it what to do (Máhr et al., 2010). The intelligent agent has been proposed as a software design paradigm, which is different from the sequential, the structural and the object-oriented approaches by its autonomy in deciding when to invoke its action (Parunak, 1997).

The fundamental approach of agent-based system is to simulate real-world systems with a group of interacting autonomous agents modeled as computer programs (Zhou et al, 2009). Since agent based modeling is a consideration that is developed according to system requirement, it is expected that they have several common and distinguishing characteristics. In fact, in spite of the growing interest in multi-agent systems, there is no agreement on what actually constitutes agent hood, that is, what are the fundamental characteristics of agents (Garcia et al., 2004). On the other hand, Jou and Kao (2002) defined the characteristics of an agent based on agent's behaviors: fundamental, auxiliary, implicit and application oriented (role based). The first one is fundamental characteristics. As a goal is state of affairs to be achieved in the environment, an agent must perceive and process the environment changes (Jou and Kao, 2002). The second one is auxiliary characteristics. An agent also needs information from users and other agents through communication mechanisms such as user interfaces and agent communication language to complete the delegated tasks (Jou and Kao, 2002). In addition to fundamental and auxiliary characteristics, some implicit characteristics with presumptive feature exist within an agent (Jou and Kao, 2002). Trustworthiness (veracity and benevolence), perceptibility, rationality, persistence, flexibility, and competence are some examples (Jou and Kao, 2002). The final characteristics are related to specific applications, e.g., the surfing behaviors of cyberspace, the facial expressions attached to entertainment, and so forth (Jou and Kao, 2002). All these features make agent technology an interesting approach for a wide set of applications (Garcia-Montoro et al., 2007).

3.4.1 Number of agent types and agents

An agent-based model has a set of agents defined by the model creator. Number of agents can be variable or constant. Their types or functions may also vary or they can be assigned for repetitive tasks. In meta-heuristics based agents, number of agents can directly be proportional (or the same) with the population/colony size. Both "number of agent types and number of agents" should be considered in the design of agent-based modeling for DOPs since they can affect the solution space directly. In his study, Tan (1993), claims that, as the number of agents is increased, state space exponentially increases in terms of the number of agents. A large state space means

more state exploration for the model that Tan (1993) considers, and this yields slower learning.

3.4.2 Communication type

Multi-agent system, also called ‘self-organized system’ is a computational system in which multiple interacting intelligent agents work together to solve difficult problems, which may be impossible for an individual agent (Yan et al., 2010). Multi-agent based modeling allows complex natural behavior of various interacting entities to emerge from a set of simple individual rules (Razavi et al., 2010). Communication is the most common means of interaction among intelligent agents. Any observable behavior and its consequences can be interpreted as a form of communication (Mataric, 1995). Two different approaches have been defined for communication (Genesereth and Ketchpel, 1994). Direct communication covers the agents handling their own coordination. Designer of an agent-based system can consider direct communication with respect to problem domain. The cost of communication and availability of obtaining qualified solutions are the factors to be considered for employing direct communication mechanisms in the models.

On the other hand, assisted/indirect coordination covers the systems in which agents rely on special system programs to achieve coordination (Genesereth and Ketchpel 1994). One of extensively employed indirect communication method is stigmergy. Stigmergy is a class of mechanisms that mediate animal-animal interactions. It consists of indirect communication that is taking place between individuals of an insect society by local modifications induced by these insects on their environment (Hadeli et al., 2004).

Although communication is essential for agent-based modeling, it may not be perfect as expected. In most of the current research on multi-agent systems, people assume that communication of agents is guaranteed (Satoh et al., 2000). However, communication can be broken, suspended or delayed depending on the flow of data or information. In this regard, research of problem solving under incomplete communication is very important (Satoh et al., 2000).

3.4.3 Coordination type

Since DOPs may have varying objectives, constraints/restrictions and parameters; agents cannot be informed on the global state of the system by themselves. Therefore, in order to achieve global objectives, coordination is essential to allow the agents to adjust their local states or conditions. Communication provides channels for coordination. Coordination is a property of a system of agents performing some activity in a shared environment (Huhns and Stephens, 1999). The degree of coordination is the extent to which they avoid extraneous activity by reducing resource contention, avoiding livelock and deadlock, and maintaining applicable safety conditions (Huhns and Stephens 1999). Figure 3.3 illustrates the taxonomy for the types of coordination defined by Huhns and Stephens (1999). Panzarasa et al. (2001) prefer to use the generic term "interaction" instead of "coordination". They (Panzarasa et al., 2001) define cooperation as working together to achieve a common objective and they also define negotiation as coming to a mutually acceptable agreement on some matter (Panzarasa et al., 2001). In other words, cooperation is coordination among nonantagonistic agents, while negotiation is coordination among competitive or simply self-interested agents (Huhns and Stephens, 1999).

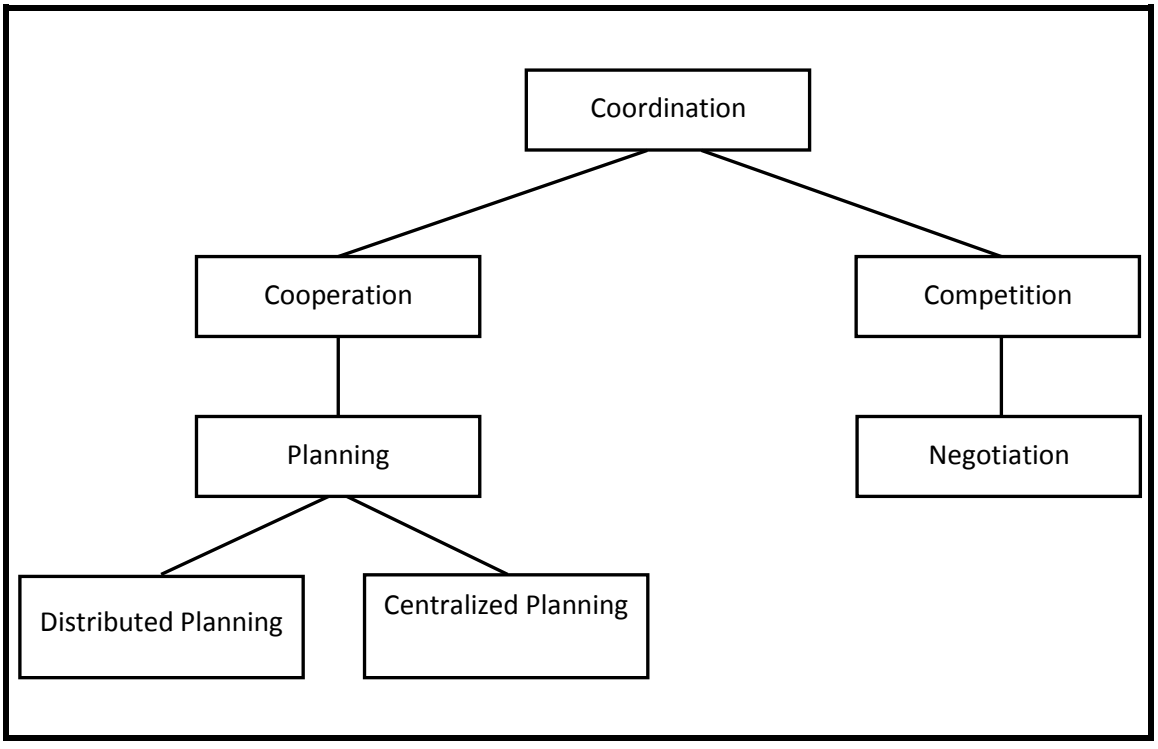

Figure 3.3 Taxonomy of coordination for agent-based modeling (Huhns and Stephens 1999)

However, perhaps the most fundamental and powerful mechanism for managing inter-agent dependencies at run time is negotiation- the process by which a group of agents come to a mutually acceptable agreement on some matter (Jennings et al. 2001). Negotiation underpins attempts to cooperate and coordinate (both between artificial and human agents) and is required both when the agents are self-interested and when they are cooperative (Jennings et al. 2001).

3.4.4 Optimization mechanisms

There are different approaches used as optimization mechanism in the agent-based systems for DOPs. These methods can be classified as: heuristic based approaches, exact mathematical methods, and market based approaches. Heuristic based approaches cover, evolutionary based approaches and other techniques like: particle swarm optimization, immune-based algorithms ant colony optimization etc (Baykasoğlu and U. Durmuşoğlu, 2012).

3.4.4.1 Heuristics

For large size problems, it is hard to obtain the optimal solution due to the large size of the problem files. In order to reduce the computational time, heuristic approaches can be used for obtaining solutions (Baykasoglu et al. 2011). Heuristics includes some set of procedures for obtaining acceptable solutions to the problems by decreasing the time requirements. There is a natural correspondence between autonomous entities and meta-heuristics, and problem solving with an optimization problem (Pelta et al., 2009). In the literature, agents are sometimes defined as individuals, particles etc. (Wang and Shixin, 2010) which is the main actors for heuristics. If one looks at natural process like bird flocks or fish schools a strong similarity to multi-agent systems can be found very quickly (Wagner et al., 2003). In this regard, there are many studies adapting heuristic approaches to agent-based systems (Baykasoğlu and U. Durmuşoğlu, 2012).

3.1.4.2 Mathematical Methods

Although it is too complex to deal with DOPs using classical mathematical methods, increasing power of computer technologies let some researchers to use mathematical

methods. Change frequency and the response time of the mathematical methods directly affect the solution's availability at the desired time. In this regard, the preferability of mathematical methods appears to be less when compared with the other methodologies (Baykasoğlu and U. Durmuşoğlu, 2012).

3.4.4.3 Market Based Approaches

For an optimization approach to be considered as market oriented, it must at least fulfill the following basic requirements (Karlsson et al., 2007):

- There must be a well-defined market mechanism, which includes some notion of prices (which often are expressed in terms of some monetary unit). The market mechanism regulates how negotiations and trade are performed among the participating agents and hence determines how certain commodities can be traded for certain other commodities (Karlsson et al. 2007).

- There must be some arguments for why the agent strategies are reasonably realistic, given the market model. That is, assume we have some model of information available for the different agents and a well-defined model for how they interact (the market mechanism). Then the strategies must be consistent with the agents' attempt to maximize utility, given bounded rationality (Karlsson et al. 2007).

3.5 Introduction to agent-based dynamic optimization problem solution strategy (ABDOPSS)

The first generic step of developing a solution strategy for any optimization problem is to define specifying class of the problem (Baykasoğlu and U. Durmuşoğlu, 2012). Class of the problem is an important factor that is affecting the required solution approach. Therefore, the ingredients that characterize an agent-based modeling approach for a DOP must somehow possess a presentation scheme to provide a solution strategy and a basis for comparison of one with another. Therefore, this presents how solution strategies with agent-based approach can be summarized in a concise manner. In this regard, a classification scheme is designed and presented in a tabular form called ABDOPSS (Agent Based Dynamic Optimization Problem Solution Strategy).

ABDOPSS distinguishes different classes of agent based algorithms applied to DOPs (via agent-related features like: communication type, cooperation type and problem related features like: dynamism of objective functions, parameters etc.) by enumerating the fundamental ingredients of each of these algorithms with respect to problem domain. The ingredients that have been considered for ABDOPSS have been discussed in the previous sections.

The main structure of the table consists two rows (row 1, row 2 as illustrated in Table 2), indicating the problem related features (row 1) and agent related features (row 2), respectively. At each corresponding column of a row, an entry, gives the necessary indication for the corresponding criteria. Table 3.2 shows such an empty table and corresponding values for the table (Baykasoğlu and U. Durmuşoğlu, 2012).

Table 3.2 Representation of ABDOPSS (Baykasoğlu and U. Durmuşoğlu, 2012)

PROBLEM RELATED FEATURES (ROW 1)							
Search Space/ Solution Environment	Objective Function Dynamism	Restriction/ Constraint Dynamism	Parameter Dynamism	Performance Measure	Benchmarked	Applied/ Framework	Real Life/ Test Problem/ Simulation
AGENT RELATED FEATURES (ROW 2)							
Number of Agents	Number of Agent Types	Communication Type	Coordination	Coordination Type	Population Type	Optimization Mechanism	
If dynamic: D If Multiple: M If directly related with population size: PS	Any numeric value	If direct: D If indirect: I	If exists: 1; else: 0	Cooperation, Data Sharing, Decision Aid	N/A, Static, Population, Dynamic Population Size	If Heuristics: H If Mathematical Methods: MM If Market Based Algorithm: MBA	

3.6 Sample applications of the ABDOPSS

Since it takes a considerable time to determine features listed by ABDOPSS, 18 articles are exemplified within the contents of this chapter. It is remarkable to state here that, this chapter does not seek to find all publications on DOPs and represent

those using ABDOPSS, but it is rather sought for exemplification of the proposed ABDOPSS. In this regard, typical versions of different approaches have been tried to be illustrated in Table 3.3. Two rows are merged into one for easiness of readability. The studies covered here are roughly classified and some unclear parts made it relatively difficult to prepare ABDOPSS. It is certain that those 18 papers could be better presented by their developers (Baykasoğlu and U. Durmuşoğlu, 2012).

Wang and Liu (2010) presented an agent-based evolutionary search algorithm (AES) for solving dynamic travelling salesman problem (DTSP). In their study, the term agent indicated a candidate position in the search space. Agents resided in a lattice-like environment and a collection of such agents were termed as population. They applied a recombination and local updating procedure to the fittest agent referring to a predefined neighborhood. They also combined the perturbation learning strategy to further reinforce the performance of the local updating rule and hope to seek the global optimum rapidly under changing environments. Dynamic version of KroA100 TSP was selected as the case of implementation. Offline performance which was the best of generation fitness averaged over 100 runs and then averaged over the data gathering period. Experimental result and relevant t-test result indicated that the performance of AES was excellent. Researchers indicated that the superiority of AES lie in its faster convergence and optimum tracking ability in dynamic environments. It is concluded that AES algorithm had a more quickly and robust convergence capability than standard genetic algorithm (SGA) on dynamic problem.

Billiau and Ghose (2008) proposed a new algorithm for solving Distributed Constrained Optimization Problems (DCOPs). The proposed algorithm, which has been called as Support Based Distributed Optimization (SBDO), has used agent level objectives instead of weighted or soft constraints that have been used by other DCOP algorithms. In DCOPs, constraints/ objectives of the problem change over time by adding or removing constraints/objectives. Researchers compared the performance of this algorithm with Asynchronous Distributed Optimization (ADOPT) and Distributed Pseudotree Optimization Procedure (DPOP) by using 120 meeting scheduling problem. Result showed that, although SBDO algorithm has not guaranteed to find the optimal solution, it reliably found solutions within two percent

of the optimal solution. Results also revealed that SBDO was able to find near optimal results significantly faster than ADOPT and in a comparable time to DPOP.

Máhr et al. (2010) compared two structurally different planning approaches, which were agent-based solution and on-line optimization approach, for drayage operations in an uncertain environment. The structurally differentiating feature of the two solution approaches was the level of control: centralized (on-line optimization) and decentralized (agent-based solution). Dynamic vehicle routing problem (DVRP) with two types of uncertainty, arrival time and service time, was employed in order to compare the performance of agent-based solution and on-line optimization approach. Moreover, scenarios were developed under different arrival time and service time. In their study, containers and truckers were defined as agent. Each container agent sold itself on an auction to the truck agents. Researchers concluded that when the online optimization approach performed better, it was by capitalizing the optimal (or near optimal) balance between routing and rejection cost. When agents performed better, their flexibility provided by their distributed nature was the competitive advantage.

Voos (2009) focused on dynamic resource allocation problems especially in continuous systems. In this study, resource allocation was expressed as an optimization problem, which could be decomposed into single optimization problems, under certain constraints. Researchers proposed to solve this optimization problem in a distributed fashion by using multiple agents. These agents have acted as local optimizer and coordinated their local solutions to an overall consistent solution. In this study, market based interaction mechanism was employed and the agents have calculated and negotiated complete supply and demand trajectories using model based predictions. In order to test the performance of the proposed approach, researcher employed this approach to three technical examples. Results revealed that this approach could be used to cope with resource allocation in dynamic environments (Baykasoğlu and U. Durmuşoğlu, 2012).

Li and Li (2009) proposed a multi-agent coordination and integration method which has been called intercommunication job-sharing hybridization, for solving complex problems. These complex problems could be decomposed into smaller separate sub-problems, with communications/exchange between sub-problems identified, human

decision-makers' roles clearly defined and managerial judgment incorporated. With the proposed method, the overall problem was divided into distinct jobs which were then assigned to relatively independent and distributed agents. These agents have shared data, information knowledge and carried out different tasks synchronously or asynchronously in the context of Internet/intranets/extranets to produce solutions. In addition, these agents have linked human participants' judgments and intuition together for joint problem solving. The architecture of the Internet-enabled multi-agent-based hybrid support system for international marketing planning was exemplified. Researchers constructed a prototype, which has been called Agent International, of the proposed multi-agent hybrid framework. This prototype covered some key features of the conceptual framework and its architecture. The potential and value of the prototype was evaluated by a questionnaire, which was prepared and delivered to the corresponding firms in London. According to the responses, researchers concluded that the prototype system was rated somewhat moderately in helping understand pertinent marketing decision making factors, exploring various alternatives, performing analysis, coping with uncertainty, incorporating managers' judgment an improving the confidence and quality of international marketing decision-making.

Tang et al. (2004) presented an auction based dynamic optimization decision support system. The presented decision support system was applied in solving an automobile load makeup-planning problem. The proposed system included three types of agents. These were the yard agents, representing the shipping yard, the truck scheduler agent, representing the transportation company who sold trucks and executed transportation and load agent representing a truck during the planning state. Each type of agent had its own behaviors. In order to solve the static load make up problem, researchers implemented 3 heuristics which were empirical (EM), minimum spanning tree (MST), vehicle routing optimization (VRO). In addition to these heuristics, a virtual market enabled by auction mechanism was employed to solve dynamic optimization problem. Researchers developed the mixture of static optimization with MST algorithm and dynamic optimization with the auction mechanism, MST Dyn, at the beginning of the day, they apply static optimization on the guaranteed information, and then they apply the dynamic optimization until the loads are fixed at the beginning of lining up. In order to test the performance of the

algorithms, same scenario was used and each algorithm has run for 120 days. Transportation cost and dwell time were selected as the performance measures. Results showed that, EM algorithm produced the worst performance and MST Dyn the best. The MST algorithm produced the worst performance and EM algorithm the best based on the transportation cost. Researchers indicated that MST Dyn algorithm produced a little bit more transportation cost than EM algorithm. Therefore, it was concluded that MST Dyn algorithm produced the best comprehensive performance (Baykasoğlu and U. Durmuşoğlu, 2012).

Berro and Duthen (2001) presented an optimization method in dynamic environment. The proposed method, using software agents, tried to provide accurate solutions and react quickly to changes in the state of the problem. In this method, software agents were randomly created and they try to colonize an optimum of the function. The system composed of two parts namely control system and agent. An agent has not communicated but it perceived the presence of other agents. When the functions to optimize and the dimensions of the search space have been defined, the user must specify two parameters namely FORCE which would influence the searching speed and EPSILON, which calculated precision of the optimal points. In order to test the proposed method, 2 cases of optimization problems, multimodal function and multi-objective function, were used. The test results were compared with the Genetic algorithm based approaches. Researchers concluded that the first tests result of the proposed method were satisfactory in particular for the computing speed and precision.

Jiang and Han, (2008) presented a Simulated Annealing (SA) based algorithm to solve real time multi-agent decision-making problem. In this study, the SA algorithm was implemented in a centralized version and performed by the agents in parallel, without assuming the availability of communications. Since there was no standard benchmark to evaluate multi-agent decision algorithm, a random generator was used to generate all test sets. The SA algorithm was tested by comparing it, with other algorithms, especially with variable elimination (VE) algorithm with respect to the scalability and relative payoff. Results revealed that, the proposed method was almost optimal with a small fraction of the time that VE took to compute the policy of the same coordination problem. In addition, researchers indicated two main

benefits of this approach as follows: the time taken by the algorithm has grown polynomial with the number of agents and the algorithm could report a near-optimal answer at any time. Researchers concluded that, SA was a feasible approach for action selection in large complex cooperative autonomous systems (Baykasoğlu and U. Durmuşoğlu, 2012).

Zhou et al. (2008) proposed a model combining discrete event systems and Multi Agent Systems (MAS) to simulate a real time job shop. All entities of the generic job shop were modeled as autonomous agents namely job agent, machine agent, work-center agent, shop floor agent, controller agent and job releaser agent. All agents pursued their own interest with unique functions. All communications in the MAS were realized through message passing. Proportion machines busy (work-center), average number in queue (work-center), maximum number in queue (work-center), average daily throughput (shop floor), average time in system (shop floor), average total time in queues (shop floor), maximal size of work in process (shop floor) were selected as performance indicators. Results of the case study demonstrated the advantage of distributed data collection and analysis. It was indicated that, the case study also validated the proposed system by statistical analysis and comparison to existing simulation results in similar test case.

González et al. (2010) presented a new centralized cooperative strategy based on trajectory methods (tabu/taboo search) for solving DOPs. The proposed strategy was compared with two methods namely a multi-QPSO and Agents. The multi-QPSO is a Particle Swarm Optimization (PSO) variant with multiple swarms and different types of particles where there exists an implicit cooperation within each swarm and competition among different swarms. On the other hand, agents are an explicit decentralized cooperation scheme where multiple agents cooperate to improve a grid of solutions. Researchers tried to assess the possibilities of trajectory methods in the context of DOPs and to draw attention on explicitly including cooperation schemes in methods. A set of solver\threads were consisted in the proposed strategy. Each solver could implement the same of a different resolution algorithm for the problem at hand. The coordinator was responsible for processing the information received from the solver and producing subsequent adjustments of their behavior by sending "orders". Exchange of data was achieved by using a blackboard model, with two

blackboards. One of them was written by the solvers that wrote the reports of their performances and read by the coordinator; and another was written by the coordinator that wrote the orders and read by the solvers. The information flow in the proposed strategy was achieved by using the following three steps: firstly, performance information was sent from the solvers to the coordinator, and then this information was processed and stored by the coordinator and finally, the coordinator sent directives to the solvers. A set of rule, based on Reactive Search Ideas, was employed to control the solvers. In order to test compare the performance of the proposed strategy, moving peaks benchmark problem and three commonly used multimodal real test functions were selected. To emphasize the importance of dynamism and optimal tracking, and to reduce the number of variables to control in the experiments, only the position of the peaks was altered. Proposed strategy and the other two methods were compared according to offline error. Just before a change, offline error of each algorithm was recorded and this value was averaged over all changes, for all independent runs. Therefore, Mean Fitness Error was calculated. Researchers indicated that the proposed strategy could consistently outperform the results of the two other methods. They concluded that the cooperation included in agents provided some benefits over multi-QPSO on the easier problems, but since the optimization done in agents relied only on using simple random perturbations of solutions it may be enough to cope with more difficult problems, even with the help of the cooperation.

Lepagnot et al. (2010) presented a new method called multi agent dynamic optimization (MADO) to solve dynamic optimization problems. In MADO, a population of agent has explored the search space. Three modules namely, memory module, agent manager and coordinator were employed. The number of agents in the system may have varied temporarily, but the number of agents along the whole search process tended to be equal to the predetermined value. This could be achieved by the coordination who would send a delete instruction if the number of agents was higher than a predetermined value. To test the performance of the MADO, Moving Peaks Benchmark problem was employed. Offline error and standard deviation were selected as performance measure. Different variants of MADO were compared with the proposed one. Results indicated that MADO was better than all the simpler

variants. Researchers concluded that the proposed MADO was a promising method for solving dynamic optimization problems.

Yan et al. (2010) proposed an agent based evolutionary search (AES) search method. In AES, a population of agents has represented potential solutions. Similar to EAs, AES gradually converged in the search space during the run, especially when the environment has been stationary for some time. In order to improve the performance of AES for DOPs, two diversity maintaining mechanisms, random immigrants and adaptive dual mapping were employed. Dynamic 0-1 optimization problems which were generated from static optimization problems by using XOR generator were investigated. Nine different dynamic test problems were constructed. The environment was periodically changed every predetermined number of generations. Different change severities were employed to test the performance of AES. The performance of AES was compared with traditional SGA, the primal dual GA and the GA with random immigrant, where the worst 10% individuals were replaced with random individuals every generation. Mean best of generation fitness was selected as performance measurements. According to the mean best of generation fitness, researchers indicated that AES could always outperform other EAs on almost all problems. Researchers stated that the competitive and learning behaviors of agents could always help AES to obtain a better performance than the peer GAs could do. They concluded that some dynamic characteristics of the environments might affect the performance of the algorithm.

Hanna and Cagan (2009) presented an evolutionary multi-agent system (EMAS) for adaptive optimization. EMAS which has employed the evolution of design strategies within a cooperative virtual team has evolved as conditions change and as new solution states were discovered during the optimization process. A set of strategies for creating solutions were represented by population of agents. In EMAS, the strategies for generating solutions have been recombined, altered, and removed by applying genetic operators that are in typical genetic and evolutionary algorithms. However, researchers have made EMAS different from genetic and evolutionary programming techniques by adding cooperation dimension. Cooperation was employed to combat the problem of not knowing which strategy to use in an unknown but static design space. It has been provided by embodying each strategy in

an autonomous agent and allowing the population of agents to communicate. Researchers used the well-known combinatorial optimization problem of travelling salesman. Researchers tried to illustrate that the proposed framework was capable of increasing the effectiveness of individual solution strategies by evolving and coordination them a decentralized manner. Three simple heuristic construction algorithms called nearest insertion, farthest insertion and arbitrary insertion were employed. Beside these, three heuristics based algorithms, 2-Opt, 3-Opt and simple mutation were also used. Two basic reduction algorithms, random reduction and best partial reduction were employed. Different from the typical genetic algorithms, gene strings represented agent strategies for generating solutions, not the solutions themselves. The design architecture used in this study was similar to asynchronous team architecture. In this study fitness was based on the ability of the agent to make positive changes in its destination memory. First of all researchers selected a 48-city problem to compare the resulting tours generated by EMAS with those generated by both individual algorithms on their own as well as priori determined hybrid algorithms. When the results of EMAS were compared with the results from running construction algorithms from each starting city and then running improvement algorithms on the resultant tour, researchers found that trials that correlated more strongly with the average patterns had better final values in terms of distance from the optimal value of the average solution quality. In addition to the 532-city problem and 48-city problem, researchers applied these patterns to a team solving a Euclidean 237-city problem modeled on a very large scale integration layout problem. Researchers has seen that the cooperative teams of individual strategies evolved to generate better solutions than both individual strategies alone and a priori set hybrid strategies. It was indicated that the strength of the EMAS algorithm lies in its ability to evolve the best team of agent dynamically. It was also stated that, one of the strength of the EMAS algorithm was as a predictive or learning guide for which set of algorithms or strategies should or should not employed and when. Researchers concluded that, utilizing EMAS in the proposed way has been shown to lower computation time while maintaining or even improving solution quality.

Pelta et al. (2009) investigated if the type of rules employed previously in cooperative systems for static optimization problems have had sense when applied to DOPs. The proposed strategy was based on the joint use of a population of solutions

and optimizers (agents). Researchers analyzed the roles that diversity and decentralized cooperation mechanisms in the performance of the methods. Researchers aimed to propose and compare two kinds of control rules to update the solution's set. These rules were a simple frequency based re-sampling (probabilistic) rule and a fuzzy-set based rule. Researchers also tried to understand the behavior of both rules in order to develop more efficient cooperative strategies for DOPs. In this study, cooperation was understood as a mechanism for information sharing. The population of solutions had two purposes, which were to serve as an implicit memory that has evolved by means of the action of agents and to be a communication channel for them. An explicit cooperation mechanism was proposed. This mechanism, which was not always triggered, was based on a simple idea. In order to test the performance of the proposed strategy, a set of experiments were developed by using Moving peaks benchmark problem. The main aim of the experiments was to evaluate the behavior of both rules, which may trigger the explicit cooperation mechanisms. Offline error was used as performance measurement. The performance of the cooperative system for different setting of each rule and the system's dynamic behavior were also investigated. The results of the proposed strategy were compared with some published results. Researchers stated that the fuzzy-based rule has been better than the frequency rule. As a conclusion, researchers indicated that both rules have been competitive when compared with a state-of-the art of the algorithm.

Xiang and Lee (2008) proposed an agent-based dynamic scheduling approach, which employs Ant Colony Intelligence (ACI) with local agent coordination. The goal of their study was to represent a dynamic manufacturing system through an MAS. They also used ACI to improve the global performance of the system. In the proposed system, entities were modeled as intelligent agents with related knowledge of their own functions and goals. MAS was used to provide parallel execution of commands. Beside this, MAS had the intelligence of negotiation to enhance system performance. The agent coordination mechanism used in the study was inspired by both foraging and division of labor of ant colony in MAS. There were five types of agents in the proposed MAS. They were order agents, job agents, machine agents, work center agents and shop floor agents. Researchers indicated that, different from the previous studies, a more generic manufacturing model with less unrealistic assumptions was considered. Furthermore, ACI was integrated with both machine agents and job

agents to solve both task allocation and task scheduling problems. Two types of disturbances were introduced. One of them was resource related disturbance including machine break down and machine recovery. The impact of integrating ACI in agent coordination was investigated by simulating a realistic shop as a multi-agent manufacturing system. In this disturbance, unreality was expressed in terms of mean time between failure and the mean time to repair. Another was source related disturbance including new order\job arrival and existing order\job cancellation. To types of agent coordination, namely coordination based on ACI (MAS + ACI) and coordination using FIFO dispatching rule (MAS + FIFO) were compared. Mean flow time, mean tardiness, throughput, buffer size and machine utilization was employed to measure the performance of the proposed approach. Results showed that a MAS +ACI reduced buffer size, max queue number, mean flow time and tardiness. Researchers concluded that a MAS+ACI were outperformed MAS + FIFO.

Wang and Usher (2002) presented an agent-based job shop model which employed the contract-net protocol as an agent's negotiation mechanism. In this study, two types of agents, named as job agents and machine cell agents, were used with pure hierarchical control structure. Researchers considered routing flexibility to provide more options for job agents. Average flow time and average queue time were selected as performance measurement. In order to measure the impact of the collaborative factor that was incorporated into the contract-net protocol on the performance, a job shop with five different loading levels were simulated. According to the findings, the collaborative factor did not have much effect on mean flow time when the system load is light, but a significant decrease was observed when the system was heavily loaded. When they examined the average queue time for jobs at each machine cell, they observed that the negotiation scheme with the proposed factor reduced the WIP levels of the bottleneck machine cells when the system was under heavy load. Based on the simulation results, researchers concluded that the collaborative factor could improve the performance of the contract net-based negotiation scheme in agent-based scheduling problems.

Wang et al. (2008) proposed a multi-agent approach to study the dynamic scheduling problem in a flexible manufacturing system (FMS) shop floor. The proposed approach was combined with a filtered beam search (FBS). Researchers aimed to

show the feasibility of the proposed approach and to evaluate the approach via computational experiments. Dynamisms in this system were provided by new job arrivals. Minimization of a weighted quadratic tardiness function was selected as the objective function. There were two types of agents, a system optimal agent (SOA) and several cell-coordinated agents (CCAs). Cooperation and coordination among distributed CCAs and SOA were employed to realize the scheduling goal. Five modules, which were called as communication, cooperation and coordination, FBS-based algorithm for decision making, execution and monitoring, human interface, were used. In addition to these modules, one local knowledge base and one capacity database were also included. FIPA CNIP-based negotiation protocol was selected. FBS was performed by filtering phase and beam selection phase. A prototype system was built to show the practicability of the proposed approach. The performance of the proposed scheduling scheme on the prototype system was compared to two dispatching rule combinations. Two dispatching rules were used for cell selection, named as dispatching by objective function value and dispatching by make-span, one dispatching rule for routing assignment, named as modified shortest processing time in the selected cell and one dispatching rule for determination of starting time of an operation on the selected machine. When the results on the performance of the average number of jobs tardy were investigated, researchers indicated that the quality of the global schedules generated with the proposed scheduling scheme was better than those of two dispatching-rue combinations. In addition, researchers sought the time required for a complete process of scheduling negotiation. They concluded that the proposed scheduling scheme was promising for real world implementation in multiple manufacturing cells of size.

Table 3.3 Examples of ABDOPSS (Baykasoğlu and U. Durmuşoğlu, 2012)

	PROBLEM RELATED FEATURES (ROW 1)									AGENT RELATED FEATURES (ROW 2)						
Authors	**Title**	**Search Space**	**Objective Function Dynamism**	**Constraint Dynamism**	**Parameter Dynamism**	**Performance Measure**	**Benchmarked**	**Applied/ Framework**	**Real Life/ Test Problem/ Simulation**	**Number of Agents**	**Number of Agent Types**	**Communication Type**	**Coordination**	**Coordination Type**	**Population Type**	**Optimization Mechanism**
Pelta et al. (2009)	A Study on Diversity and Cooperation in a Multi-agent Strategy for Dynamic Optimization Problems	D	1	1	1	OE Relative Error	S	B	TP	M	1	Direct	1	Cooperation	S	H
Wang and Liu (2010)	An Agent Based Evolutionary Search to Dynamic Traveling Salesman Problem	D	0	1	1	Offline performance	OM	B	TP	PS	1	Direct	1	Collaborative	S	H
Billiau and Ghose (2008)	Robust, Flexible Multi-agent Optimization Using SBDO	D	1	1	1	Solution time Avg # iteration Quality	OM	B	TP	NA	NA	Direct	1	Data sharing	NA	H
Máhr et al. (2010)	Can agents measure up? A comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty	D	0	1	1	Total cost	OM	B	R	M	2	Direct	1	Data sharing	NA	MM H
Voos (2009)	Agent-Based Distributed Resource Allocation in Continuous Dynamic Systems	S	0	1	1	Allocated amounts of steam in the simulation	NA	B	R	M	3	Direct	1	Data sharing	S	MBA
Li and Li (2009)	A Multi-Agent Based Hybrid Framework for International Marketing Planning under Uncertainty	D	0	1	1	-	NA	B	P	M	10	Direct	1	Data sharing	NA	Other
Tang et al. (2004)	Wireless-based Dynamic Optimization for Load Makeup Using Auction Mechanism	D	0	1	1	Transportation cost Dwell time	OM	B	HS	M	3	Direct	1	Decision aid	S	H
Berro and Duthen (2001)	Search for optimum in dynamic environment: a efficient agent-based method	D	1	1	1	Value of objective function	OM	B	TP	PS	1	Direct	0	-	D	H
Jiang and Han (2008)	Real Time Multi-agent Decision Making by Simulated Annealing	D	1	1	1	Scalability of SA algorithm, Relative payoff comparison	OM	B	HS	M	1	Indirect	0	-	S	H

Authors	Title	Search Space	Objective Function Dynamism	Constraint Dynamism	Parameter Dynamism	Performance Measure	Benchmarked	Applied/ Framework	Real Life/ Problem/ Test Simulation	Number of Agents	Number of Agent Types	Communication Type	Coordination	Coordination Type	Population Type	Optimization Mechanism
Zhou et al. (2008)	Simulating the Generic Job Shop as a Multi-Agent System	D	0	1	1	M. Flow Time WIP Buffer size Resource Utilization	OM	B	HS	M	6	Direct	1	Data sharing	S	MM
González et al. (2010)	A cooperative strategy for solving dynamic optimization problems	D	1	1	1	OE, M. Fitness Error	OM	B	TP	D	1	Direct	1	Centralized	S	H
Lepagnot et al. (2010)	A new multi-agent algorithm for dynamic Continuous optimization	D	1	1	1	OE, standard deviations	OM	B	TP	D	2	Indirect	1	Coordination	D	H
Yan et al. (2010)	Agent based Evolutionary Dynamic Optimization	D	0	0	1	OE	OM	B	TP	PS	1	NA		Competition	S	H
Hanna and Cagan (2009)	Evolutionary Multi-Agent Systems: An Adaptive and Dynamic Approach to Optimization	0/1	0/1	0/1	0/1	-	NA	F	-	PS	1	NA	1	Data sharing, decision aid	S	H
Pelta et al. (2009)	Simple control rules in a cooperative system for dynamic optimization problems	D	1	1	1	OE	OM	B	TP	M	1	Direct	1	Data sharing	S	H
Xiang and Lee (2008)	Ant colony intelligence in multi-agent dynamic manufacturing scheduling	D	0	1	1	M. flow time, M. tardiness, throughput, buffer size machine utilization	S	B	HS	M	4	Indirect	1	Data sharing	S	H
Wang and Usher (2002)	An Agent-Based Approach for Flexible Routing in Dynamic Job Shop Scheduling	D	0	1	1	M. flow time	S	B	HS	M	2	Negotiation	1	Contract Net (negotiation)	NA	Bidding mechanism
Wang et al. (2008)	FBS-enhanced agent-based dynamic scheduling in FMS	D	0	1	1	Avr. value of system objective function Avr. # jobs tardy	S	B	HS (Protoype)	M	2	Negotiation	1	Negotiation	NA	H

3.7 Concluding Remarks

This chapter proposes a classification scheme (ABDOPPS: Agent Based Dynamic Optimization Problem Solution Strategy) for agent-based approaches which are employed for solving Dynamic Optimization Problems (DOPs). In this chapter, 18 typical articles providing agent-based solutions to the DOPs are scanned through the literature and represented using the ABDOPPS.

The ABDOPPs is expected to be beneficial to researchers in many ways. Similarities of the features located in ABDOPPS can be used to define classes of solution strategies by their descriptions. In this regard, classes of the problems may orient researchers to focus on certain strategies. Using the dynamism related features of the corresponding DOPs presented in ABDOPPS, unpredictability levels of certain problems can be determined and be used to reclassify problems. These representation forms can also be used to discover the role of presented features and their importance for solution quality.

CHAPTER 4

AGENT-BASED MODELING

4.1 Introduction

Several applications and theoretical approaches, which utilize agent-based modeling methodologies, have been discussed through Chapter 2 and Chapter 3. Being different then the Chapter 2 & Chapter 3, this chapter will discuss two popular software/platform, AnyLogic and NetLogo (used for agent-based modeling) and their pros and cons separately. Although those platforms are evolving by the time, current versions will be issued in this chapter. For the easiness of reading, it is worth to state that, that software which is structured to perform agent-based modeling will be called as "agent technology" throughout this chapter.

4.2 Is It Worth to Use Agent Technology?

Agent-based modeling has been a purely academic topic up to now. However, the increasing demand for business optimization caused leading modelers looking at combined approaches to gain a deeper insight into complex interdependent processes having very different natures. Therefore, agent-based modeling using agent technology is expected to beneficial for many of the dynamical cases. However, there are some concerns that should be taken into account to take full benefits of agent-based modeling. Its computational affordability and easiness of developing models and some other relevant factors should be carefully considered before the use of agent-based modeling. A cost-benefit can be performed to see if it is vital to used agent technology. Sometimes it can very difficult and challenging to model sophisticated systems. As Wilensky (1999), states that even using something as simple as NetLogo can be challenging for those with little programming skill. Although some commercial solutions are provided via software packages like

AnyLogic, NetLogo, StarLogo, Simpack, AgentBuilder, JADE, JAS, and SEAS etc…, it is crucial to know the programming language behind those packages since some adjustments may be very critical to model the real system. Table 4.1 lists some of those software packages, their domain and the programming language behind them. Table 4.1 has been gathered and abstracted from Wikipedia (http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software)

Table 4.1 Agent Technology Platforms

Platform	Primary Domain	Programming Language
AgentBuilder	General purpose multi-agent systems	Knowledge Query and Manipulation Language (KQML); Java; C; C++
AnyLogic	Agent-based general purpose; Distributed simulations	Java; UML-RT (UML for real time)
JADE	Distributed applications composed of autonomous entities	Java
JAMEL (Java Agent-based MacroEconomic Laboratory)	Building agent-based macroeconomic simulations	No programming required
JAS	General purpose agent based	Java
JASA (Java Auction Simulator API)	Computational economics; Agent-based computational economics	Java
jES (Java Enterprise Simulator)	A single enterprise or a system of enterprises	Java
SEAS (System Effectiveness Analysis Simulation)	The US Air Force's Multi-Agent Theater Operations Simulation	Tactical Programming Language (TPL)
Jade's sim++	Parallel simulation; Applied simulations; Network planning; Electronic CAD; Real time communication simulation	C++
SimAgent	Research and teaching related to the development of interacting agents in environments of various degrees and kinds of complexity	Pop-11, like Common Lisp, is a powerful extendable multi-purpose programming language supporting multiple paradigms.

Platform	Primary Domain	Programming Language
StarLogo	Social and natural sciences; educators; for students to model the behavior of decentralized systems	StarLogo (extension of Logo)
NetLogo	Social and natural sciences; help beginning users get started authoring models	NetLogo

4.3 AnyLogic

AnyLogic has been a commercial solution introduced as a simulation-modeling tool. It was developed by XJ Technologies. It has been agent-modeling software with numerous advantages differentiating it from the others. With the solution provided by AnyLogic; different simulation approaches system dynamics (SDs), discrete events (DEs) and agent-based simulation (ABS) has been in use in the same model (this feature is named as multi-method modeling by XJ Technologies) as demonstrated in Figure 4.1.

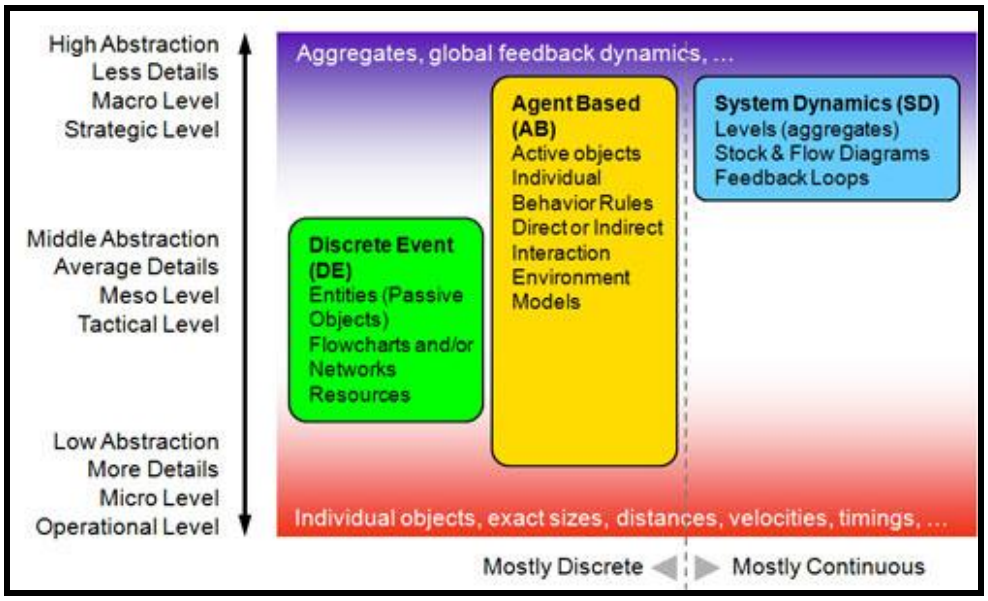

Figure 4.1 Abstraction levels of AnyLogic (http://en.wikipedia.org/wiki/AnyLogic)

Thereby, it is avoided to consider all models in a single approach. With the visual environment utilities of AnyLogic, development cost and time has correspondingly

decreased. On the other hand, AnyLogic makes it available to use and insert JAVA code to any place of the program, which gives the flexibility to user to adapt their models for their needs. Beside this facility, it has been possible to create a JAVA applet for all working models and thereby it can run anywhere without depending on the operating system.

The latest major version, AnyLogic 6.8, was released in 2012. The platform for AnyLogic 6.8 model development environment is Eclipse. AnyLogic 6.8 is cross-platform simulation software as far as it works on Windows, Mac OS and Linux. AnyLogic 6.8 covers the fundamental elements listed below:

Stock and Flow Diagrams are used for System Dynamics modeling. It is possible define stock and flow variables one by one or using a flow tool. Shadow variables can also be added for easiness of readability of the models. Table functions can be employed to see interpolation between variables. Array variables with an arbitrary number of dimensions can also be used. One of the notable features of AnyLogic stock and flow diagram is that the dependency arrows are always synchronized with the actual formulas; that is, the dependency arrow from A to B appears automatically if you type A in the formula of B and disappear when you exclude A. For flow arrows, it also works the other way around; if you delete the arrow, A will be excluded from the formula of B (XJ Technologies, 2012). Figure 4.2 illustrates view of stock and flow diagram from Anylogic.

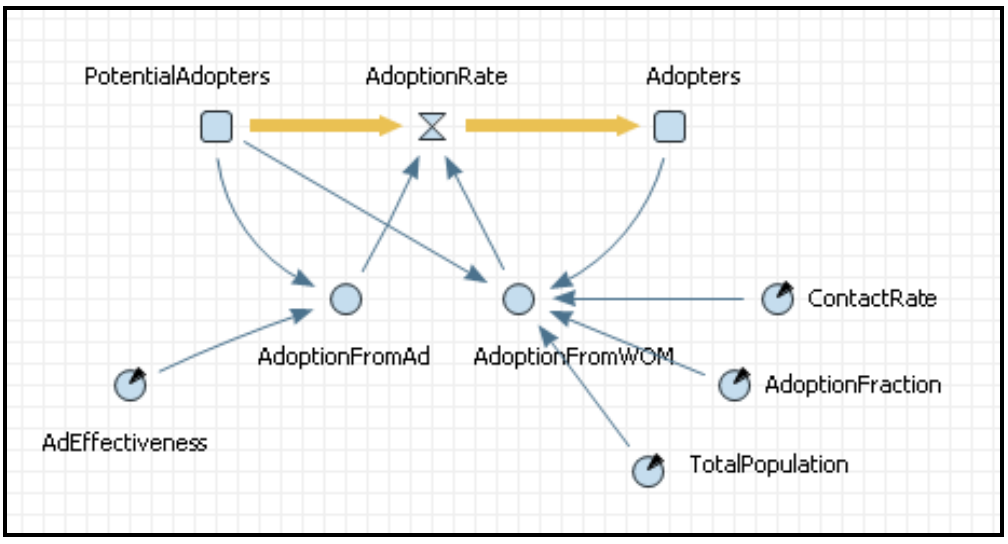

Figure 4.2 Stock and flow diagram from Anylogic.

State charts are used mostly in Agent Based modeling to define agent behavior. They are also often used in Discrete Event modeling, e.g. to simulate machine failure. State represents a location of control with a particular set of reactions to conditions and/or events. A state can be either simple or, if it contains other states, composite. Control always resides in one of simple states, but the current set of reactions is a union of those of the current simple state and of all composite states containing it -i.e., a transition exiting any of these states may be taken (XJ Technologies, 2012). Figure 4.3 illustrates view of State Charts from Anylogic.

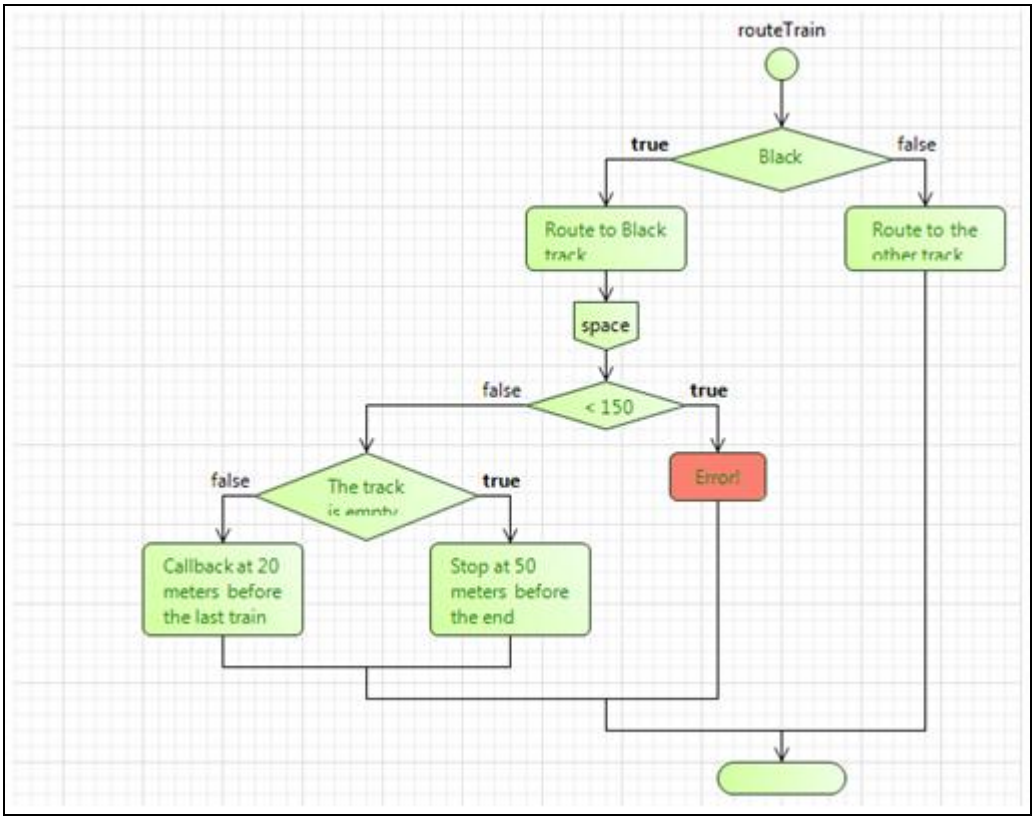

Figure 4.3 State charts of Anylogic.

Action charts are the charts that are employed to define algorithms. They may be used in discrete event modeling, e.g. for call routing, or in agent-based modeling, e.g. for agent decision logic. Complex simulation modeling cannot go without algorithms that usually perform some data processing or calculations. Action charts are very helpful since using them you can define algorithms even if you are not familiar with syntax of Java operators. In fact, action chart visually defines a function and it can be used for ordinary AnyLogic functions. However, using *action charts* gives one

benefit that is more evident: it visualizes the implemented algorithm, making it more intuitive to other users. Figure 4.4 illustrates view of action charts from Anylogic.

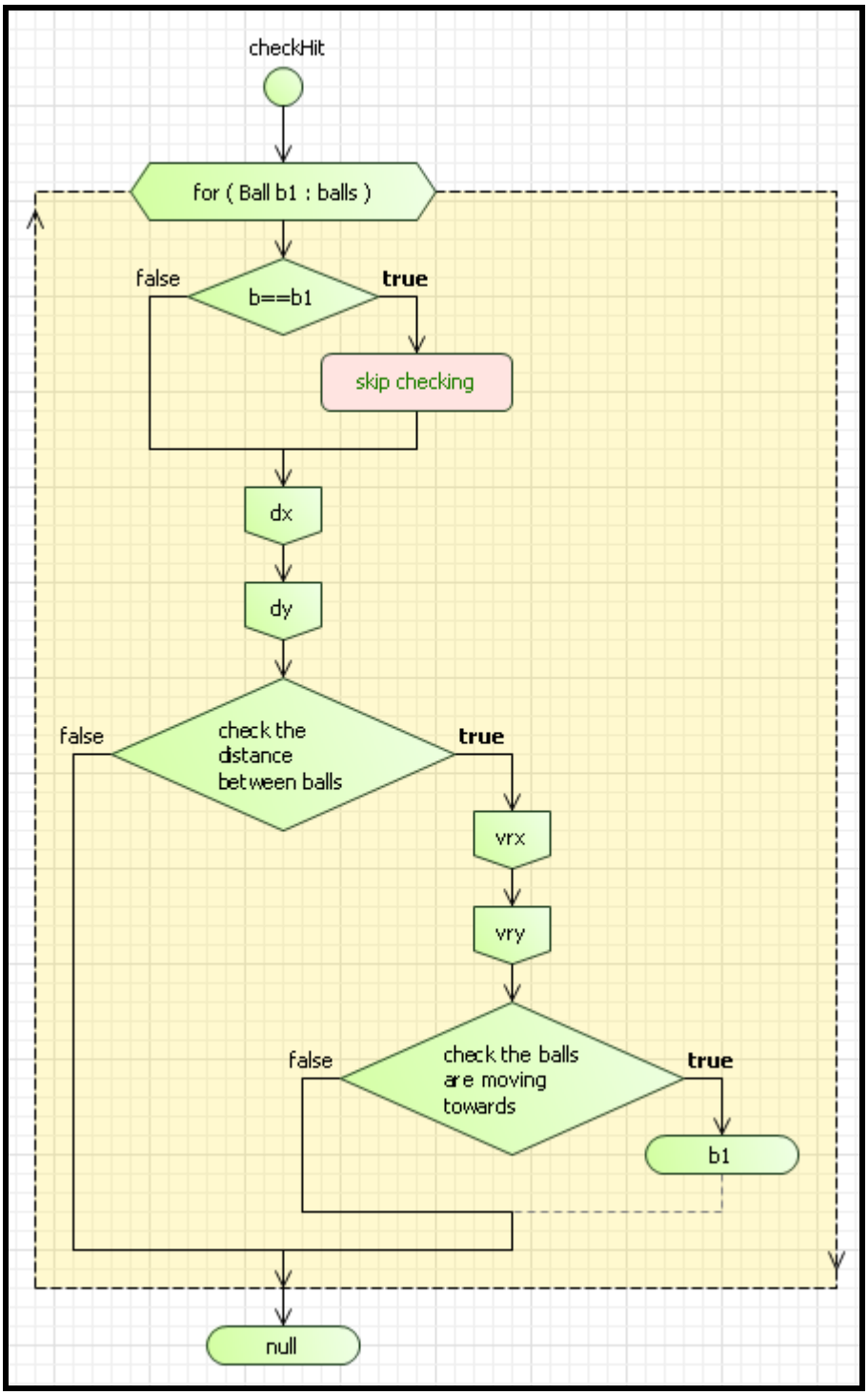

Figure 4.4 Action charts of Anylogic.

Process flowcharts are the basic construction used to define process in discrete event modeling. Looking at this flowchart you may see why discrete event style is often called Process Centric.

The language also includes low level modeling constructions (variables, equations, parameters, events etc.), presentation shapes (lines, polylines, ovals etc.), analysis facilities (datasets, histograms, plots), connectivity tools, standard images, and experiments frameworks. Figure 4.5 illustrates view of process flow chart of Anylogic.

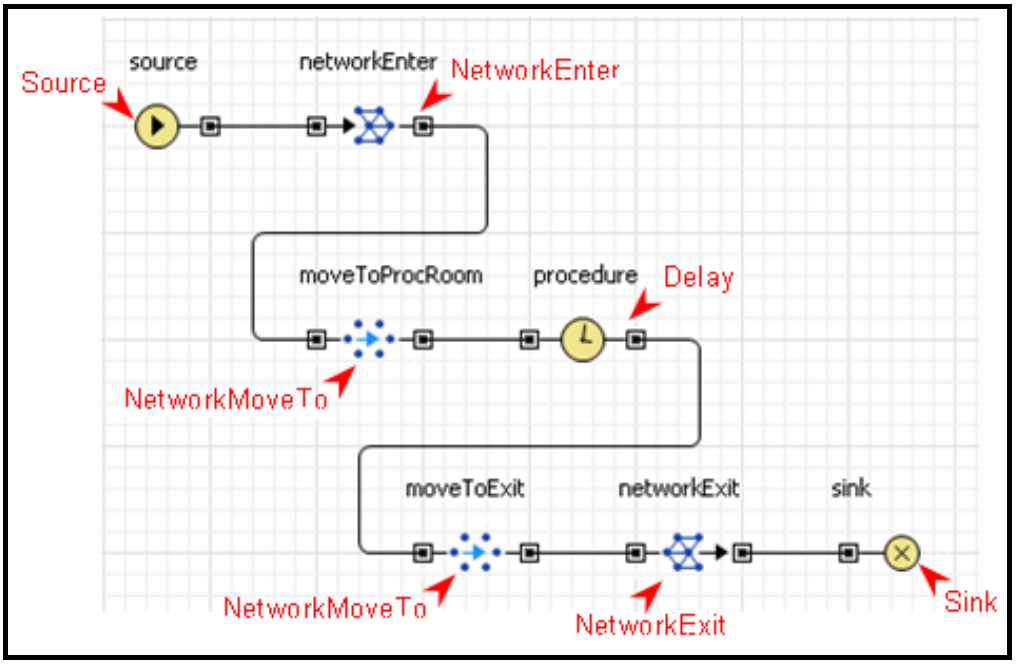

Figure 4.5 Process flow chart of Anylogic

AnyLogic includes a graphical modeling language and allows the user to extend simulation models with Java code. The Java nature of AnyLogic lends itself to custom model extensions via Java coding as well as the creation of Java applets, which can be opened with any standard browser. These applets make AnyLogic models very easy to share or place on websites. In addition to Java applets the Professional version allows for the creation of Java runtime applications which can be distributed to users. These pure Java applications can be a base for decision support tools.

4.4 NetLogo

NetLogo is a multi-agent programmable modeling environment. It has become very popular in recent years (Wilensky, 1999). It is used by different interest groups like students, instructors and researchers. NetLogo was first created in 1999 by Uri Wilensky at the Center for Connected Learning and Computer-Based Modeling, then at Tufts University in the Boston area. NetLogo grew out of StarLogoT, which was authored by Wilensky in 1997. In 2000, the CCL moved to Northwestern University, in the Chicago area. NetLogo 1.0 came out in 2002, 2.0 in 2003, 3.0 in 2005, 4.0 in 2007, 4.1 in 2009, and 5.0 in 2012. NetLogo is written mostly in Scala, with some parts in Java. Scala code compiles to Java byte code and is fully interoperable with Java and other JVM languages. NetLogo does include a compiler that generates Java byte code. However, this compiler does not yet support the entire language, so some parts of user code are interpreted. We are working on expanding the compiler to support the entire language. Note that our compiler generates Java byte code, and Java virtual machines have "just-in-time" compilers that in turn compile Java byte code all the way to native code, so much user code is ultimately translated to native code.

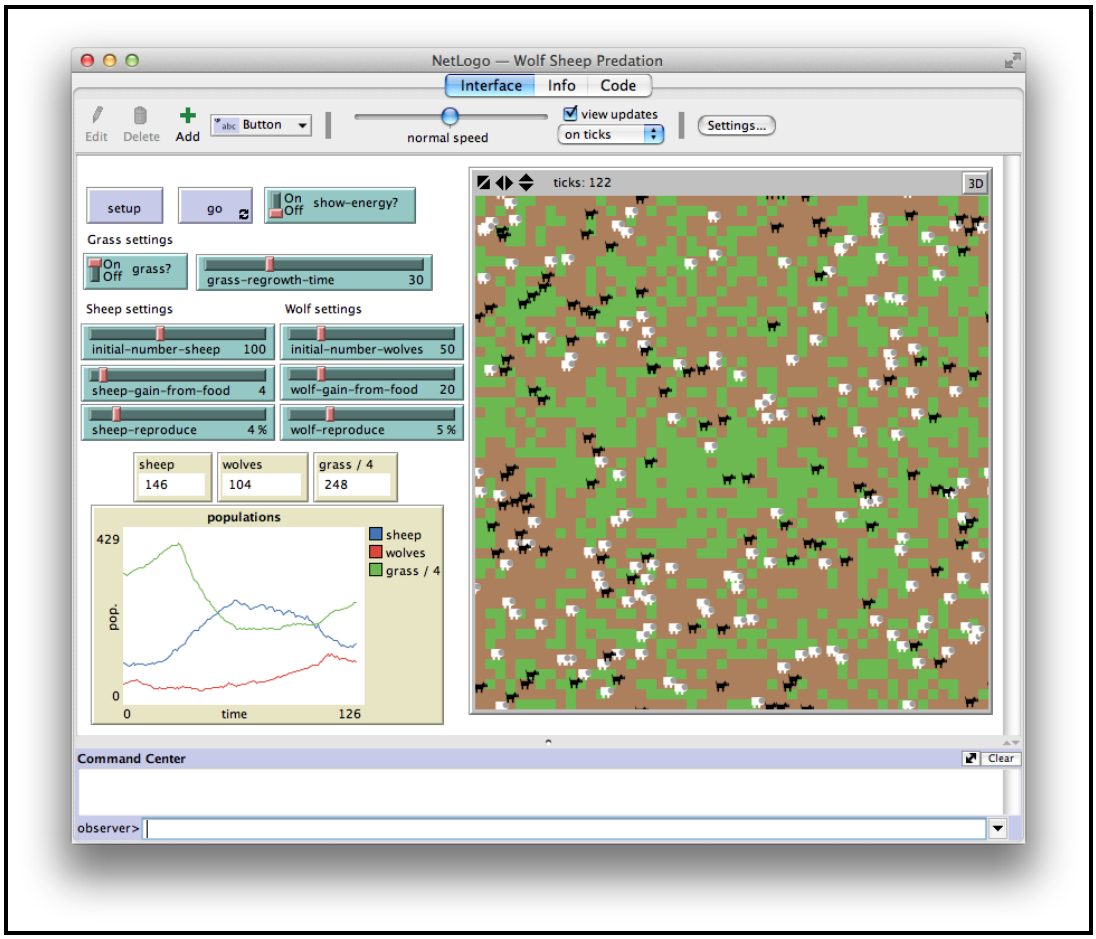

Figure 4.6 An example to interface of NetLogo

The NetLogo software is made up of agents. Agents are the utilities that can understand instructions and obey the defined rules. A NetLogo model consists of four types of agents, which are turtles, patches, links, and the observer. Turtles are a kind of agents that move around in the environment. The environment is two-dimensional and is divided up into a grid of patches. Each patch is a square piece of "ground" over which turtles can move. Links are agents that connect two turtles. The observer does not have a certain location and it only observes the environment form an overall view. On the other hand, it remarkable to state that, the observer does not witness the events just inactively, it also coaches to the other agents. When NetLogo is started up, no turtle exist in the system. The observer is able to create new turtles for the designed system. Patches are also able to create new turtles. Patches cannot move, but otherwise they are just as "alive" as turtles. Patches have coordinates. The patch at coordinates (0, 0) is called the origin and the coordinates of the other patches are the horizontal and vertical distances from this one. Turtles have coordinates too: xcor and ycor. A patch's coordinates are always integers, but a turtle's coordinates

can have decimals. This means that a turtle can be positioned at any point within its patch; it does not have to be in the center of the patch. Links do not have coordinates. Every link has two ends, and each end is a turtle. A link is represented visually as a line connecting the two turtles. If a turtle dies, the corresponding link dies immediately too.

4.4 Concluding Remarks

There are a numerous tools and commercial solutions provided for agent-based modeling. All of those tools and software have both different advantages and disadvantages changing by the objectives of their developers. Therefore, it is usually very difficult prefer one to the other where one may not provide all of the desired facilities and utilities. It is also remarkable to state that all of those platforms are still in improvement and it is still certain which will be functioning much in near future. Existing's platforms most important lack appears to deficiency of examples and good documentation about the defined classes.

CHAPTER 5
SYSTEM DESCRIPTION

5.1 Introduction

Travelling Salesman Problem (TSP) is one of the classical operation research problems, which consists of “n number of cities” where, each city is visited once using an optimal route. Similarly, Generalized Travelling Salesman Problem (GTSP) covers “c number clusters” where, each cluster is visited once by visiting one of city of the each cluster using an optimal route. These definitions of TSP and GTSP cover typical and classical versions of their types. However, in some specific cases, assumptions of classical TSP and GTSP may not be satisfactorily enough to reflect physical reality and dynamism of actual systems. During solution of those problems, some cities may be added to city domain or some may leave the system. In those circumstances, the assumption of keeping the number of cities constant fails. Consequently, “a solution provided for a specific to state defined for a time” may lose its superiority immediately after these unexpected changes.

In this respect, these dynamic types of TSP and GTSP may not be modeled with the conventional research methods since much time and memory spaces are required to setup novel models and solve them repeatedly. Multi agent-based systems, a relatively new area of distributed artificial intelligence, provide strategies that are more effective for the management of such dynamic environments. With those considerations in mind, this chapter covers different agent-based solution policies/strategies for providing promising solutions to different problems in domain of TSP. It is remarkable to state here that, promising solutions mentioned here do not equally mean the optimal solutions, it is rather the feasible solutions obtained in affordable time and in an acceptable level as demonstrated in Figure 5.1.

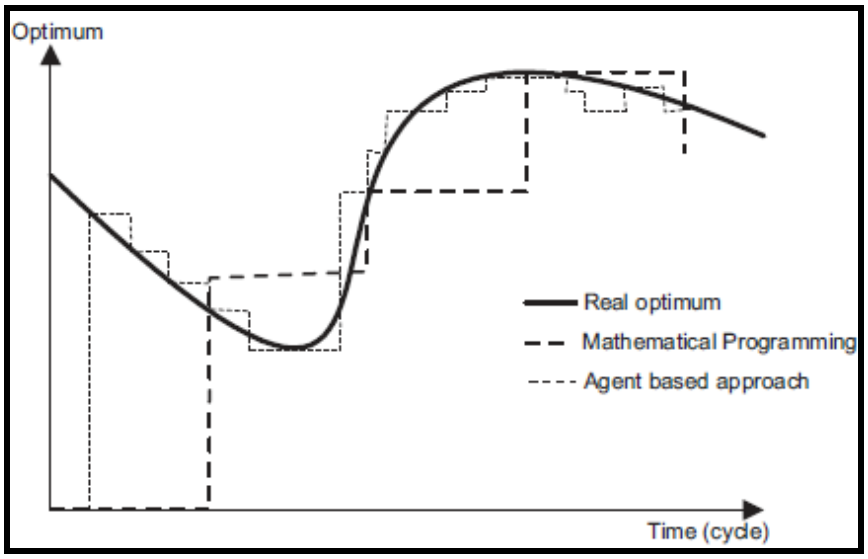

Figure 5.1 Different modeling approaches in reaching optimum in dynamic situations (http://www.whitestein.com/library/company-and-product-resources).

A continuous agent based simulation approach was used to simulate TSP and its variants and was programmed using AnyLogic by XJ Technologies. As a final product, user-friendly software with adjustable parameters has been developed (Codes of this developed software can be found in Appendix A).

Two main solution strategies are proposed for the solution of the defined problem types in this chapter. One is competition of agents without a heuristic and the other is competition of agents using Great Deluge Algorithm (GDA). Results obtained via those strategies have been compared with the solutions provided for static snapshots of certain time slots. It has been understood that, agents-based adaptable and continuous strategies outperforms the conventional approaches.

5.2 Problem Statement

For many companies, only a subset of customers require a pickup or delivery each day (Campbell, 2006). Therefore, number cities to be visited by a delivery company may change on daily base. Some new deliveries can also be received from the visited addresses. Thereby, some new destinations are added to the current visit plan. In this respect, solving a classical travelling salesman problem (TSP), which has constant number of cities to be visited, may not provide satisfactory solutions for this real case. As Chang et. al, (2009) state, improper application of routing or scheduling

policy on any such stochastic and dynamic service network often results in dissatisfied customers for late service.

In this respect, handling classical TSP problem as Dynamic TSP (DTSP) may avoid such customer dissatisfactions and inefficiencies. These unexpected changes in the number of cities transform a TSP into a Dynamic TSP (DTSP). One of the simplest ways to react to environmental changes in such dynamic problems is to consider each change as the arrival of a new problem and restart the algorithm to solve it from scratch (Raman and Talbot, 1993). However, remodeling and resolving of a problem repeatedly may consume too much time and resource.

With or without time windows, traveling salesman problem (TSP) is already NP-hard in deterministic settings (Savelsbergh, 1984). For small instances, it may be no problem to generate all solutions and pick the shortest, but when the number of cities increases the number of possible solutions 'explodes' (Eyckelhof and Snoek, 2002). Within this domain, heuristics that find acceptable solutions using an acceptable amount of resources are necessary (Eyckelhof and Snoek, 2002). Since, we can always wait longer for a solution in static optimization problems; in dynamic optimization problems, the trade-off between performance and time complexity is determined by the problem difficulty and the relationship between the rate of change and the available computing resources (Kang et al., 2004). In this respect, most cost, too much time to gain good solutions, so the general algorithms are inefficient (Li et al., 2006).

However, multi agent-based systems can provide strategies that are more effective than those conventional approaches. Intelligent agents, which are capable of informing each other and negotiating some decisions about path selection, can effectively modify existing solution in their hand, without remodeling and resolving the problem. Thus, considerable amount of time and memory space can be saved using agent-based modeling.

There is limited number of studies on DTSP. One of these studies was performed by Eyckelhof and Snoek (2002). In their paper, dynamism occurs due to change in travel

times between the cities and they present a new “ant system approach” for solution of DTSP.

Huang et al. (2001) considers a DTSP where the relative position between each other city is indefinite and there can be changes in the connecting relations of some cities. They present a framework providing differences between TSP and DTSP. Mavrovouniotis and Yang (2010), define a problem type where cities are replaced by new ones during the execution of an algorithm. They investigate a hybridized Ant Colony Optimization (ACO) with local search, called Memetic ACO (M-ACO)

Takahashi (1998) also developed a mathematical framework for solving DTSP with adaptive networks and provided a particular example of solving DTSP method. In his DTSP, distance between any pair of cities in the TSP is extended into a time variable.

Wang and Liu (2010) presented an agent-based evolutionary search algorithm (AES) for solving dynamic travelling salesman problem (DTSP). In their study, the term agent indicates a candidate position in the search space. Agents reside in a lattice-like environment and a collection of such agents is termed as population. They apply a recombination and local updating procedure to the fittest agent referring to a predefined neighborhood. They also combine the perturbation learning strategy to further reinforce the performance of the local updating rule and hope to seek the global optimum rapidly under changing environments. Dynamic version of KroA100 TSP is selected as the case of implementation.

Psaraftis (1988) stated that DTSP is in the preliminary stages and many open questions need to be discussed. Although there have several studies conducted on different types of DTSP, his claim still appears to be valid. As discussed through the previous studies, variability in number of cities has not been considered much yet.

This chapter provides different agent-based solution approaches for one dynamic TSP and four different types of dynamic Generalized Travelling Salesman Problem (DGTSP). In these variants of TSPs, a salesman with a vehicle starts transporting the deliveries from their pick-up locations to their delivery locations. Consequently, this salesman visits certain number of cities at different coordinates (geographical

coordinates) and returns to the initial city at the end of tour. Existing customers can be lost or new customers can arrive to the system thereby new cities can appear or disappear in the visit plan. This arrivals and departures from problem set occur with known statistical distributions that change with the time in the problem horizon. Customers can be served in any sequence. However, final objective is to minimize the total touring/routing time. Each processing sequence of customers defines a route of the vehicle. On each route, the vehicle moves from one customer to another, just spending time in traveling among customers at their locations. The vehicle(s) travel according to the Euclidean metric and their velocity is constant.

The rest of this chapter is organized as follow; in Section 5.3, problem types and initial setups for them are introduced; Section 5.4 presents the test study and its structure. Section 5.5 provides agent-based simulation results, and finally in Section 5.6 conclusions are drawn.

5.3 Problem Types and Initial Setups

In this section, problem types are formally defined. Within the content of this PhD thesis, one dynamic TSP and four different types of dynamic GTSP are issued for testing and comparing purposes. Before introducing these dynamic variants of TSPs it will be better to define their classical forms.

Given a list of cities and their pairwise distances, the traveling salesman problem (TSP) is to find the shortest tour that visits each city once and returns to its original city (Flood, 1995).

The generalized traveling salesman problem (GTSP) is a generalization of the traveling salesman problem (TSP), one of the outstanding intractable combinatorial optimization problems. In the GTSP problem, we are given *n* cities that are grouped into mutually disjoint districts (clusters) and nonnegative distances between the cities in different districts. A traveling salesman has to find the shortest tour that visits exactly one city in each district (Dimitrijević and Šarić, 1997).

Features of TSP variants considered in this thesis are as tabularized in Table 5.1. TSP and GTSPs that are within content of this chapter are all dynamic optimization problems (DOPs) where cities can be added or removed from the system at any time. Since there is high dynamism, all solution approaches are based on agent-based modeling where this dynamism is handled much easily.

In problem type 1, where Dynamic TSP is considered, objective function is to minimize the total cost of tour. With that purpose in mind, each city is modeled as an agent. Those city agents are authorized to communicate with general-manager agent (GMA). GMA is a central agent that is responsible to initialize auctions, create other city agents and make final decisions about the routes. Whenever GMA requires initializing an auction, it informs city agents and asks for their bids. There is also a vehicle agent, which is responsible from the delivery and transportation of loads.

Problem type 2 and problem type 3 cover Dynamic GTSP where a city from each region/cluster must be visited while trying to minimize the total cost of a tour. GTSP is a generalization of the well-known Traveling Salesman Problem (TSP) (Bontoux et al., 2010). In these types, there are regional agents (RAs), which are responsible from all activities about the cities located in their regions. Whenever GMA initializes an auction, GMA informs all RAs and asks for their bids. RAs evaluate the possibilities within their regions and provide bids for the open auctions.

Difference of type 2 and type 3 problems is only the region determination strategies that they employ. These region determination strategies are discussed in section 5.8 with details.

Table 5.1 Problem types and their properties

Problem Type:	**Type 1**	**Type 2**	**Type 3**	**Type 4**	**Type 5**
Problem Name:	Dynamic TSP	GTSP where regions are predefined	GTSP where regions are defined via a clustering algorithm	Total cost of TSP & GTSP where regions are predefined	Total cost of TSP & GTSP where regions are defined via a clustering algorithm
Objective Function:	Minimize tour cost	Minimize tour cost	Minimize tour cost	Minimize sum of main tour (GTSP) costs and local tour (TSP) costs	Minimize sum of main tour (GTSP) costs and local tour (TSP) costs
Agents:	City Agents, General Manager Agent, Vehicle Agent	Region Agents, General Manager Agent, Vehicle Agent	Region Agents, General Manager Agent, Vehicle Agent	Region Agents, General Manager Agent, Vehicle Agent, Local Vehicle Agents	Region Agents, General Manager Agent, Vehicle Agent, Local Vehicle Agents
Initial tour determination strategies:	*Agent competition if home city is predetermined ** Great Deluge Determines	*Agent competition if home city is predetermined ** Great Deluge Determines	*Agent competition if home city is predetermined ** Great Deluge Determines	*Agent competition if home city is predetermined ** Great Deluge Determines	*Agent competition if home city is predetermined ** Great Deluge Determines
Clustering Strategies	N/A	Location of clusters is already defined. Therefore, location of a created city determines the cluster of that city.	Two different clustering algorithms are employed.	Location of clusters is already defined. Therefore, location of a created city determines the cluster of that city.	Two different clustering algorithms are employed.

Problem type 4 and problem type 5 cover both Dynamic GTSP and Dynamic TSP as described in Figure 5.1. Objective function in these types include total cost that is generated by the grand tour (tour of GTSP where a city from each region is visited by vehicle agent- Cost A in Figure 5.2) and by the local tour (where vehicle agents visit all cities within a region- Cost B, C, D in Figure 5.2). Difference of these type 4 and type 5 problems is only the region determination strategies that they employ. These region determination strategies are discussed in section 5.8 with details.

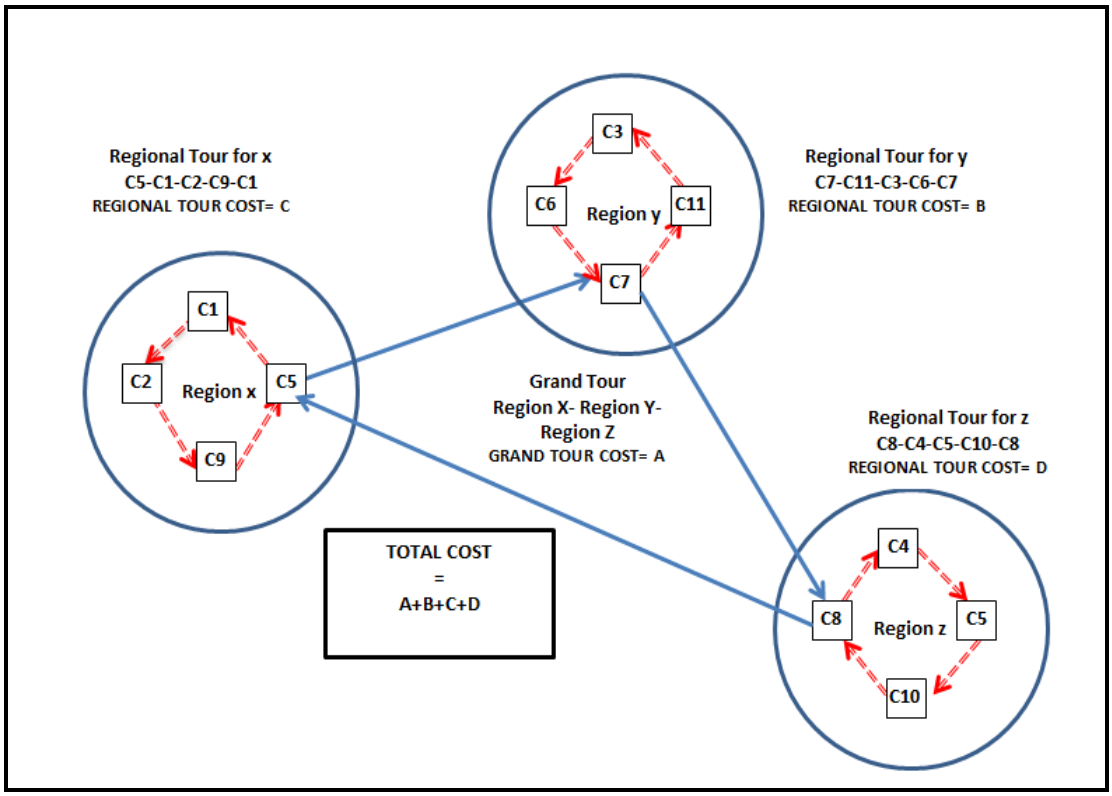

Figure 5.2 Figurative illustration problem type 4 and problem type 5

All tasks of agents are defined with the state charts in AnyLogic. The state chart of GMA is as illustrated in Figure 5.3.

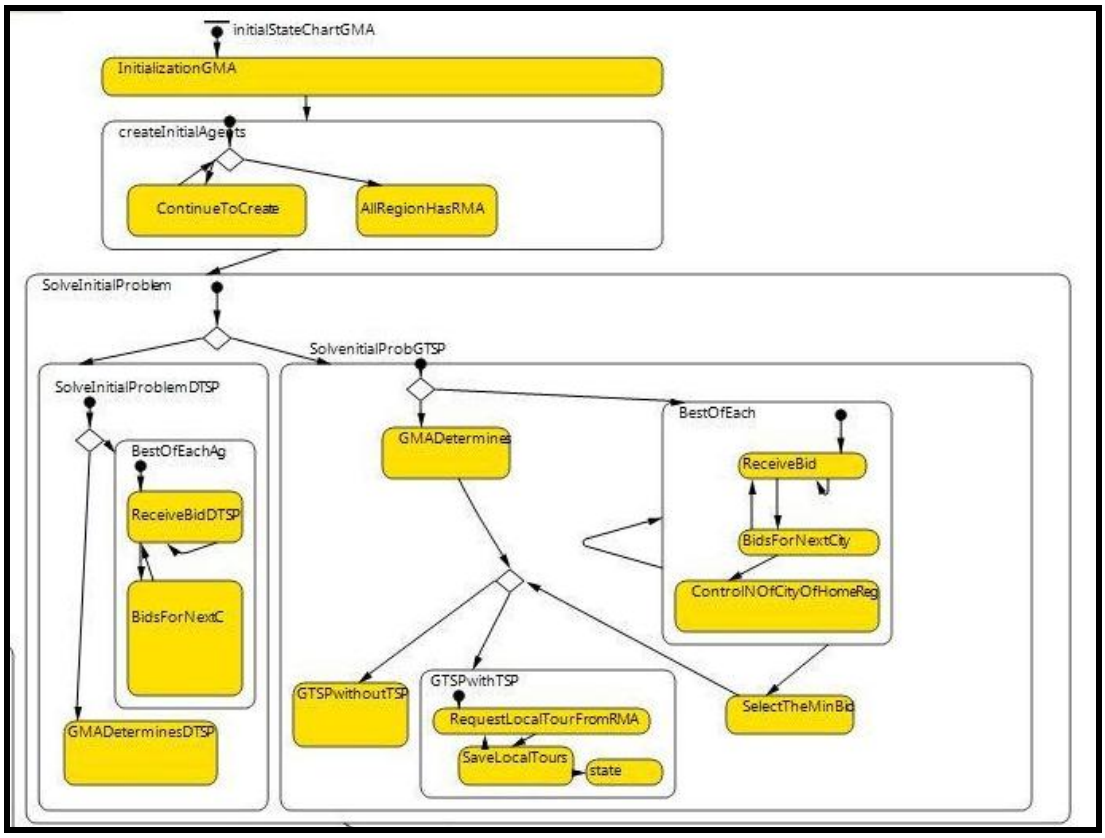

Figure 5.3 State chart of GMA defining the tasks to do

5.4 Problem Input Settings

There are some common and user defined initialization features and fundamental principles regarding the all problem kinds. These features and principles are very crucial to understand solution strategies. One is about naming and generating cities and regions. In all of problem types, initial number of cities is determined by the user. Cities are then created randomly in a user-defined environment (that has 1000 unit of width and 1000 unit of length maximum). Creation order of cities is used as the city id (the fifth randomly selected city is named as City 5).

For the problem types where regions are predetermined (type 2, type 4), regions are also named consecutively according to the row and column that they are located. If a city is in k^{th} row and j^{th} column then the cluster/region of that city is [(j-1)* number of rows +k] as illustrated in Figure 5.4.

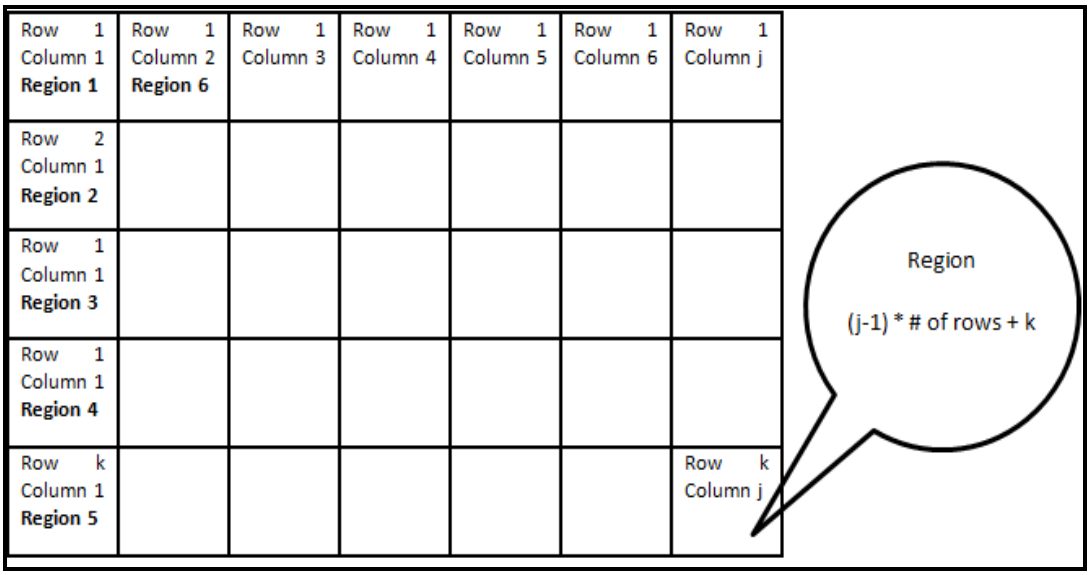

Figure 5.4 Determination of region number

If regions are not predetermined (problem type 3 and type 5) and then the region having the City 1 is named as Region 1 and then the new regions are named consecutively based on distance to center of Region 1.

For home-city and home-region selection, users have three options to select. Home city/region is determined by using one of the methods listed below:

1- The first city that is randomly created is accepted as the home city / region of the first randomly created city is accepted as the home region

2- User can select the home city/ user can select home region from the existing regions. Since user may not have idea about total number of regions that will be generated, he/she may select a non-existing region number. If that is the case, then the region having the biggest id is selected.

3- Any city can be home city that is decreasing the total cost; any region can be home region that is decreasing the total cost.

All of these initial settings are performed via a user-friendly interface with adjustable buttons as illustrated in Figure in 5.5.

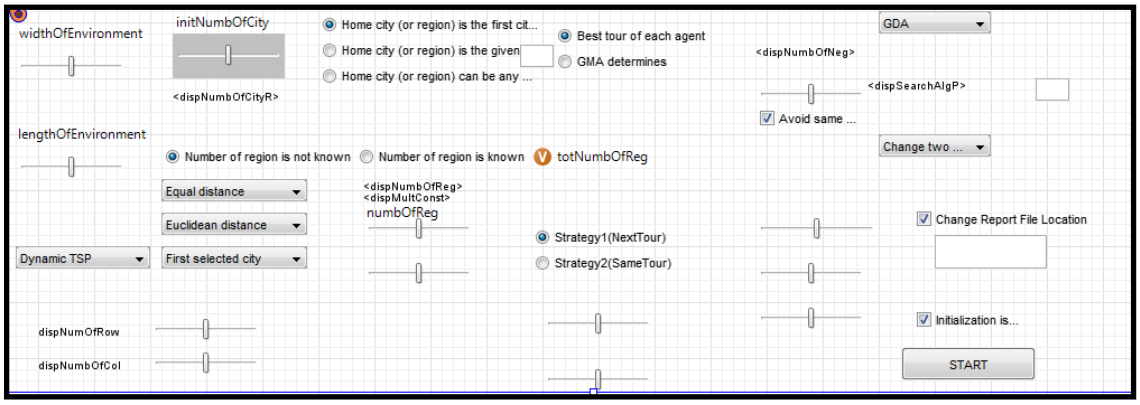

Figure 5.5 AnyLogic interface of the software for input settings

There are two main solution strategies employed for all of problem types that are defined for this section. One is competition of agents without a heuristic and the other is competition of agents using Great Deluge Algorithm (GDA).

5.5 Agent Competition Strategy

In agent competition strategy, where home city is defined, for dynamic TSP (Problem Type 1), each city agent delivers a bid to be connected to the home city. The agent offering the best bid (having the minimum cost) is then linked to the home city. After remaining agents delivers bid to be connected to the next agent (the agent that was connected to the home city). This process is followed until all cities are visited. Home city is automatically added to the last city of tour to obtain a completed tour. The agent competition strategy does not include the case where *"home city can be any city"* option.

In dynamic GTSP problems (in type 2 and type 3), if home region is determined but home city is not determined yet; then the first city of the region (having the smallest ID) is considered as the first candidate home city. General-manager agent asks all other region agents to offer their bids to be connected to candidate home city. Regional agents check all cities in their region to be connected to the candidate home city and they deliver their minimum bids to General Manager Agent (GMA). The region agent (RA) giving the best bid (suggesting the minimum cost to be connected) becomes winner agent. Suggested city of the winner agent is assumed as the city to be visited after the first candidate home city. Similarly, all unassigned remaining

RAs provide their bids to be added to the subsequent home city candidate (winner of the previous competition). Figure 5.6 represents the corresponding flow. This procedure is repeated until a complete tour is obtained. GMA keeps the best alternative for the state where the first city in the home region is candidate of being home-city.

Subsequently the second city in the home region is assumed as the second candidate of being home city and the same procedure is applied. After all those applied, at final step, GMA have different tour arrangement alternatives where each is the best for different home city alternatives. In this respect, GMA selects the alternative with minimum cost. Too many alternatives have been tested with the defined methodology. However, solution of the problem is expected to have less time compared to conventional methods since RAs evaluate their alternatives parallel to each other (simultaneously) and this helps to save time. The agent competition strategy does not include the case where *"home region can be any region"* option.

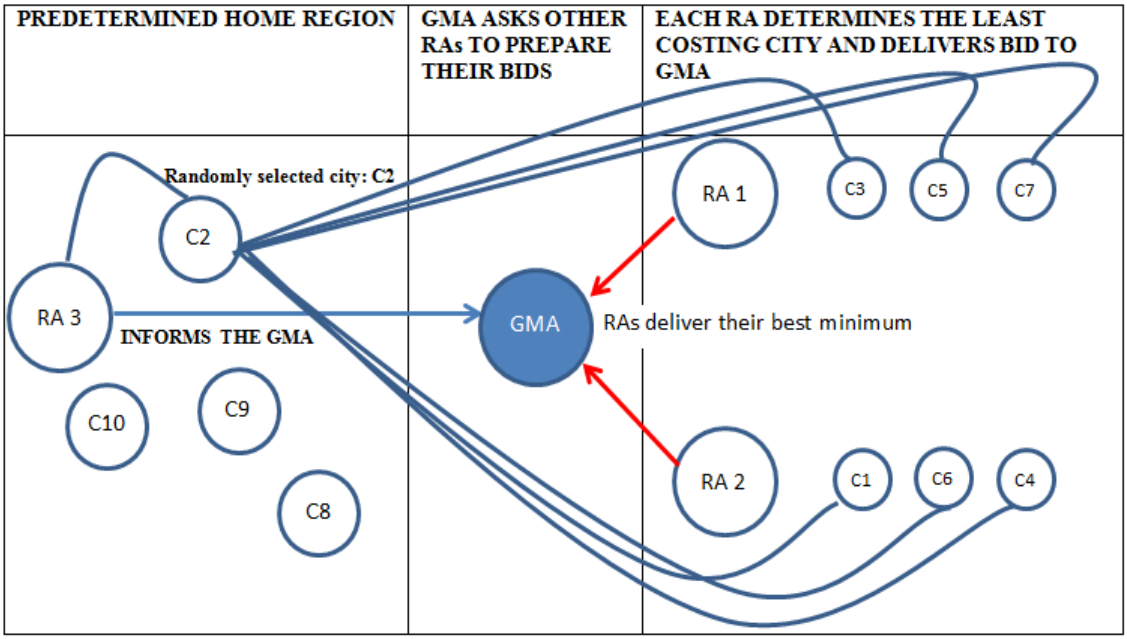

Figure 5.6 A figurative illustration of agent competition for GTSP

In problem type 4 and type 5, there are two tours to be performed. The first is the grand tour where exactly one city is visited from each region. The second is the regional tour where each city of the region is visited once and home city is visited at

the end. Here each RAs performs different regional tour alternatives using GDA and keep the best tours each starting with different cities of that region to report the GMA. If home region is determined but home city is not determined yet; then the first city of the region (having the smallest ID) is considered as the first candidate home city. Then, GMA asks all other region agents to offer their bids to be connected to candidate home city. Different from problem type 2 and problem type 3, each RA produce their alternative bids, including both total cost of their regional tours and the grant tours. However, RAs only deliver the alternative that has the smallest cost. The procedure is followed until several full-length tour alternatives are determined.

5.6 Heuristic-Based Agent Competition Strategy

Due to their characteristics, ABMs have been recently using as a promising heuristic techniques to solve problems whose domains are distributed, complex and heterogeneous (Barbati et. al, 2012). TSP has been a popular problem where various local search algorithms have been suggested for (Cowling and Keuthen, 2005). In this respect Great Deluge Algorithm has been adapted to solve the defined TSP variants. GDA is a heuristic first introduced by Dueck in 1993. It has been relatively a novel and simple algorithm applied to optimization problems. Since it needs only one basic parameter to setup, it is very attractive for solving optimization problems (Baykasoğlu, 2012). It has also several similarities with the simulated annealing algorithm (SA). As in case of SA; GDA may accept worse candidate solutions (than the current one) during its run (Burke et al., 2004). Great Deluge Algorithm (GDA) is employed for the searches of the difficult problems, where it can be too difficult to obtain exact and optimal solutions early such as dynamic problems like TSP.

The pseudo-code of a variant of GDA employed for this work is given in Figure 5.7. In implementation of the GDA, the algorithm is initialized with a random solution *s*. A numerical value of *initial cost/badness* is computed for *s* and thereby it is undesirability is measured. The higher the value of *cost/badness ((f(s))* the more undesirable is the initial random solution. Another numerical value called the *tolerance (B)* is included as being equal to the initial cost.

```
Set the initial solution s
Calculation initial cost function f(s)
Initial level B=f(s)
Specify input parameter ΔB
While further improvements is impossible
    Define neighborhood N(s)
    Randomly select the candidate solution s* ε N(s)
    Calculate f(s*)
    If  f(s*) ≤ f(s)
    Then accept s*
    Else if  f(s*) ≤ B
        Then accept s*
    Lower the level  B = B - ΔB
```

Figure 5.7 The pseudo code for the implemented GDA

The worse solutions are accepted if its fitness is less than or equal to some given upper limit *B* (in the paper by Dueck (1993) it was called a "level"). If *s** is worse than tolerance/upper limit *B*, a different neighbors* of *S* is chosen and the process repeated. If all the neighbors of *s* produce approximate solutions beyond *tolerance (B)*, then the algorithm is terminated and *s** is put forward as the best approximate solution obtained.

Initial tour determination strategy: For dynamic TSP (problem type 1) using Great Deluge Algorithm (GDA), is fully random. Random numbers are produced to determine the next city for a tour. Since double visiting is not allowed for TSP, random numbers that are corresponding to an existing city in the visit plan, it is reproduced until a new city is added.

If problem type is GTSP (type 2, type 3, type 4 and type 5) then a random floating number is produced such as 3.5. First part of this number (it is 3 in the given example) indicates the region to be selected. According to the number of cities in that region (assumed as 4 for the given example) a random city is selected by multiplying the floating part with the number of cities in that cluster/region (0.5*4=2 then City 2 is selected)

Neighborhood generation strategy: There are two strategies to generate neighborhood. One is swapping where two randomly selected cities are interchanged.

In this strategy, two random numbers are generated to determine the order of the cities to be changed.

In the second strategy, a random number is generated and all the cities that are following the selected city is taken from its location and added to home city.

Beside those two neighborhood generation strategies, user interface also presents an option to change of cities without changing the order of clusters/regions. There is also an option to avoid having same solutions in the solution set defined for GDA.

There are two options for stopping the iterations. One is defining the consecutive number of non-improving solutions and the other is maximum number of iterations.

Being different then the "agent competition strategy" here agents offer their bids according to the findings of GDA. Other mechanisms are very similar to the agent competition strategy.

5.7 New City Arrivals

In case of new city arrival for dynamic TSP (problem type 1), general manager agent (GMA) informs all city agents about the new city and asks them to possibility of adding that new city to their existing paths. As demonstrated in Figure 5.8., each city agent calculates the possible increase in the total cost of that tour if that new city is added to its route as the follower city. City agents then deliver the calculated cost value to the GMA. GMA selects the minimum bid, which is equivalent to selection of the agent giving the least cost increase.

Those calculated “cost differences” may be calculated for an already visited city or for an unvisited city. Thereby, two strategies exist here. If the option “next tour is selected” all agents deliver their bids but if “same tour option” is selected then only unvisited agents deliver a bid. If the winner of bids is an already visited agent for “next tour is selected” option, then the city is planned to be added in the next tour. Otherwise, city is added to existing tour. Thereby an agent is created and assigned for that city and the vehicle agent is informed about this newly added agent.

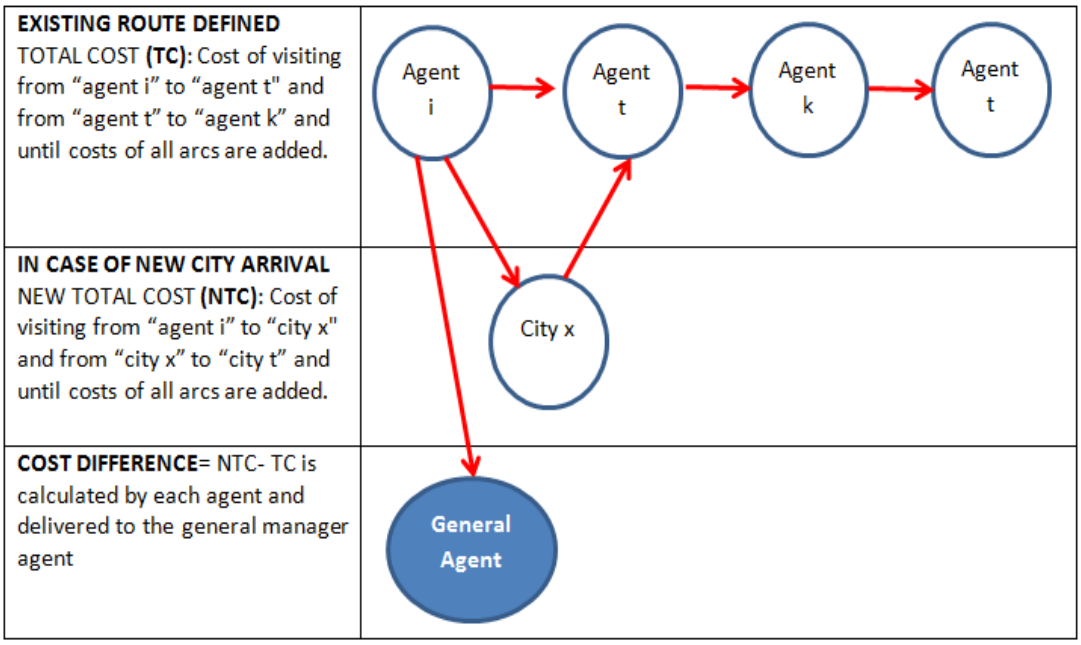

Figure 5.8 Competition of the agents when new city arrives to the system

In case of new city arrival, for dynamic GTSP where regions are predefined (problem type 2), then region of the city is found. If that city is the unique city in its region, then that city is added to visit plan and a region agent is created for that cluster. If that city is not the unique city for that region which has not been visited yet, then possibility of planning that new city to be visited is calculated. If total cost for the tour decreases then, new city is added as planned city, otherwise no change is required for the visit plan.

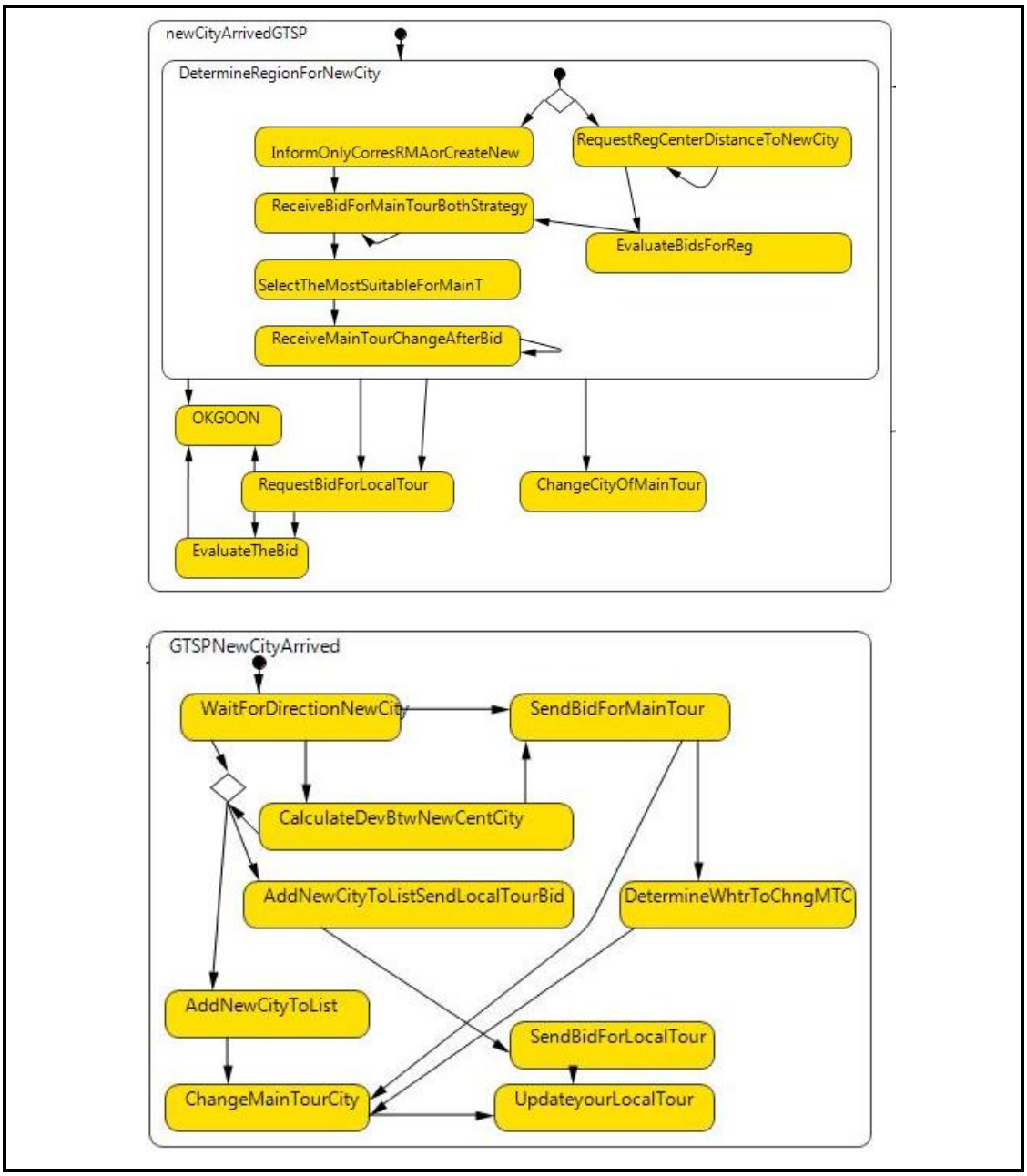

Figure 5.9 Statecharts defining the tasks to be performed for new city arrival to GTSP

If a new city appears in the case where, grand tours and regional tours are both considered and regions are predefined (problem type 4), then region of the city is found. If that city is the unique city in its region, then that city is added to visit plan and a region agent is created for that cluster.

If that city is not the unique city for that unvisited region, then firstly; home city of that region is kept constant and possibility of adding it behind the other cities is calculated. The minimum total cost found among those possible additions are saved

as COST A. Subsequently, possibility of assigning that new city as a home city (if existing home-city has not been visited yet) is searched. For that purpose, Great Deluge Algorithm (GDA) is asked to determine the regional tour cost if that new city becomes the home city for that region. This possibility also changes the grand tour cost. New total cost is saved as COST B. COST A and COST B are compared and the smaller one is selected as the new visit plan.

If regions are not known for the new coming city (problem types 3 and 5), all RAs are informed about it (coordinates of it is transmitted). All RAs check whether they can consider this new city in their region or not. This decision is taken upon two measures: average total distance of cities to region center (ATDC), and their standard deviation of distances of cities in that region (SDD). If new city's distance to a region agent is less than ATDC+SDD, then that city can be accepted for that region. Since having less deviation, from the average is preferable, if there are several regions that are able to accept that city to their region, then new city is added to the region where deviation is the least. If the city is added to one of the existing clusters then exactly same procedure is applied as described for problem type 4 and type 2.

5.8 City Deletion

In case of city deletion/disappearance, following procedures are applied with respect to the selected problem type.

If a city is already visitied, deletion of that city will have no influence in the system. If the problem type is 2 or 3, it is first checked if the deleted city is planned in the grant tour or not. If deleted city is a part of grant tour and if there is at least two cities remaining after deletion, RA delivers the second best alternative (having the same city order) in its list to be considered by GMA. If the deleted city is unique city in its region, then it is deleted and corresponding RA is destroyed (even if the region determination strategy is "intelligent clustering", no auction is performed for the mergergy of regions). If the deleted city is not in the visit plan of the grant tour, then the city is deleted only from the regional tour.

It should also be noticed that in TSP and its variants; if a planned delivery is cancelled and a vehicle is routed to that city, the already passed distance is also included in the total cost.

5.9 Region Determination Strategies

Cities in the problem domain are divided into regions (regions describe a part of physical environment where cities with similar coordinates are gathered) using a different algorithms described in this section. Each region is controlled by RAs and thereby solutions are searched simultaneously in each of regions.

There exist two strategies for region determination. One is the predetermined regions. In this type region determination (problem type 2 and problem type 4), user defines a unit area for a unit region. Subsequently, environment is divided into regions by that unit of area. For instance; if environment is 1000x1000 units and a region is defined, as 10x10 then, 100 regions is created. Thereby, it gets easier to determine a city's region just by checking its coordinates, corresponding rows and columns as illustrated in Figure 5.10.

In the second option, intelligent region determination policies are applied (problem type 3 and problem type 5). In those intelligent region determination policies, average distance (AD) has been used as the base measure. A city to be clustered is checked whether its distance is to region center is less or more then AD.

For calculation of AD, maximum and minimum values of X and Y should be found for the created cities. Difference between "maximum of X coordinate values" and "minimum of X coordinate values" determines the horizontal range of cities. Similarly, the difference between "maximum of Y coordinate values" and "minimum of Y coordinate values" determine the vertical range of cities (Figure 5.10).

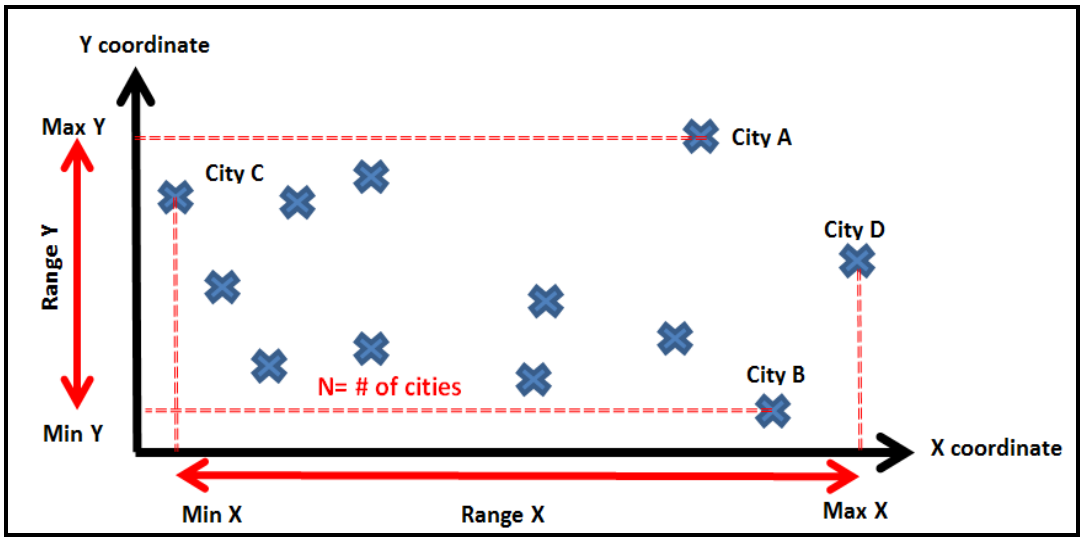

Figure 5.10 Figurative illustration of range calculation

Subsequently, AD is calculated upon the option selected by the user. Options are; Euclidian AD and circle diameter AD. Euclidian distance is calculated using Equation 5.1.

$$AD_{Euclidian} = \sqrt{\left(\frac{\text{Range of X}}{\text{number of cities}}\right)^2 + \left(\frac{\text{Range of Y}}{\text{number of cities}}\right)^2} \qquad \text{(Equation 5.1)}$$

If circular-diameter option is selected, following procedure is applied. If number of cities were and they would be homogenously distributed there should n circles with radios r where total area of them should be equal to $A = n\pi r^2$. From this equation, "ideal r value" is computed (Figure 5.11). Circle diameter AD is calculated using Equation 5.2.

$$AD_{Circle\ Diamater} = \sqrt{\frac{Range\ of\ X * Range\ of\ Y}{number\ of\ cities * \pi}} \qquad \text{(Equation 5.2)}$$

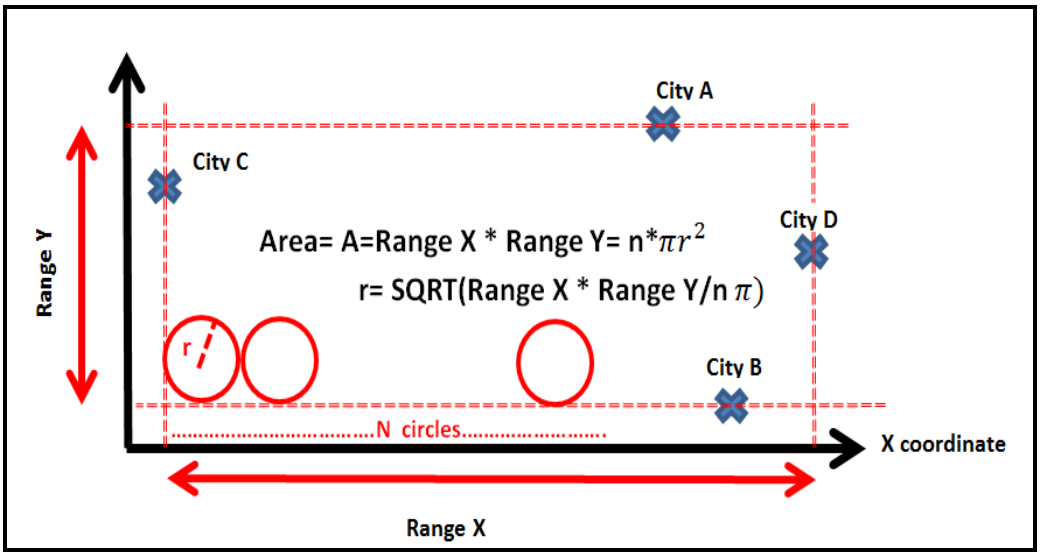

Figure 5.11 Circles in case of homogenous distribution of circles

After AD is calculated according to the user preference, following intelligent region determination options are presented:

1- **Equal Distance Option:** In this region determination option, City 1 is selected as the center of Region 1. Cities having at most "AD" much distance to City 1 is clustered in the same region (Region 1). After determination of all cities in region 1, the nearest city to City 1 (which has not been clustered yet) is found and accepted as the center of the second region. This policy is continued until all cities have been clustered.

2- **Modified Equal Distance Option:** In this option, equal distance option is performed first. The distance between the first city of Region 1 and the first city of Region 2 is calculated and called as "Modified Equal Distance (MED)". In this procedure cities having distance less than AD is clustered like "equal distance option". If there is only one city in region then the cities having less distance then MED is clustered at the same region (excluding the Region 1 and Region 2). After determination of all cities in a region, the nearest city to center city (which has not been clustered yet) is found and accepted as the center of the next region. There can be marginal cases for Modified Equal Distance Option. If some cities are accepted to a cluster due to having less distance then MED, next

region's center city may be closer to previous region as illustrated in Figure 5.12. In that case, that city is added to next region.

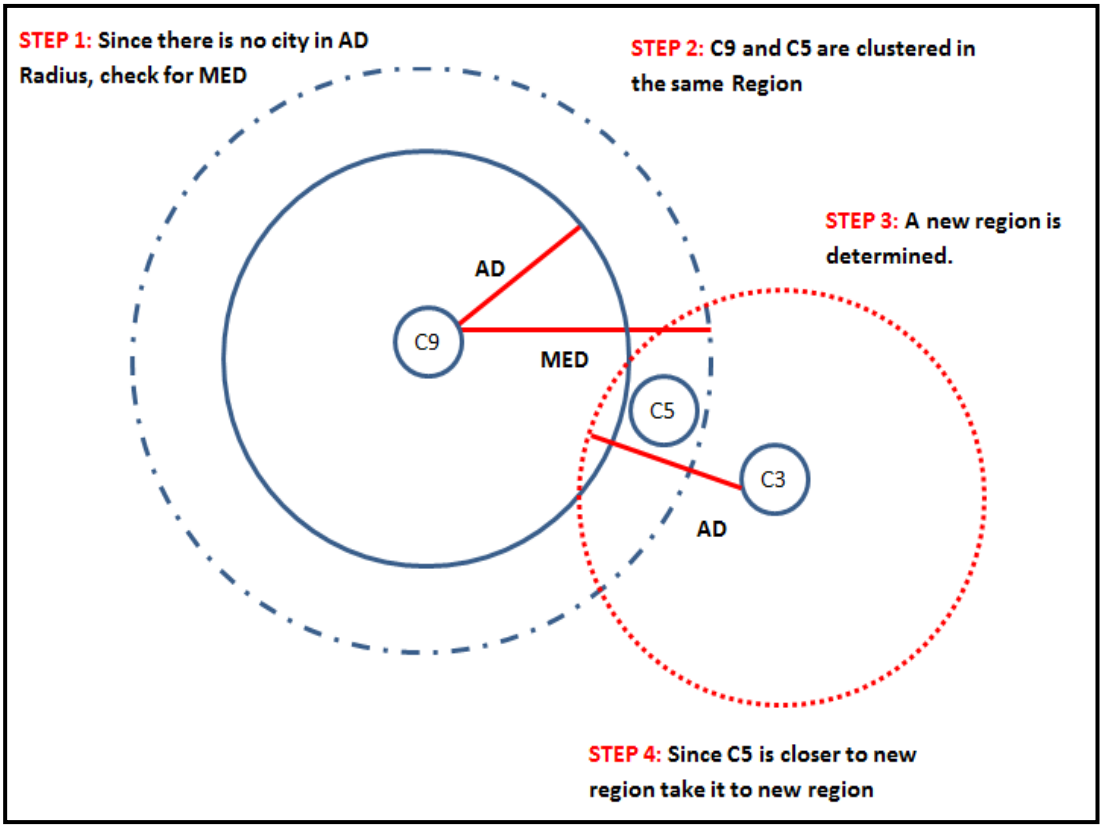

Figure 5.12 Marginal cases for modified distance option

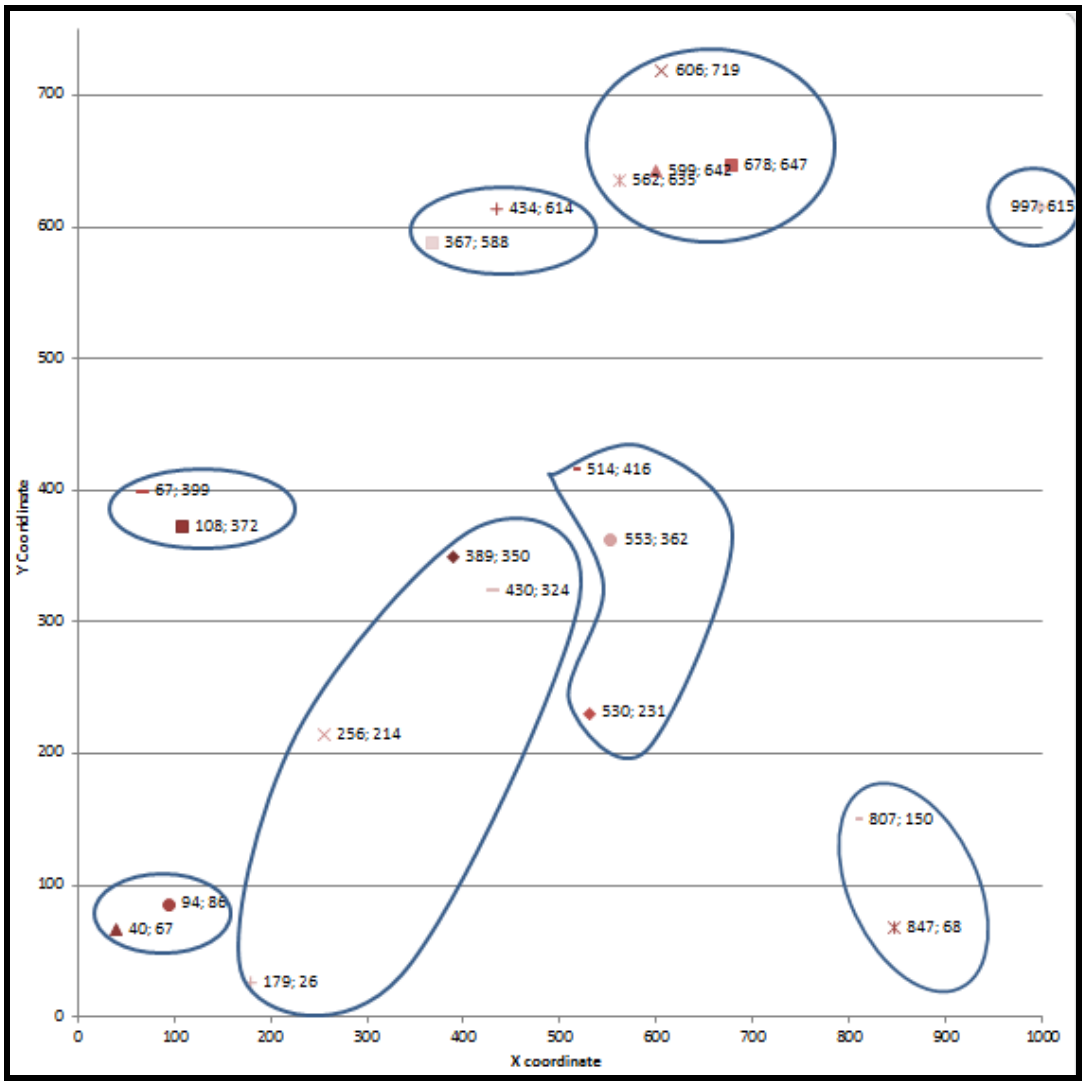

Figure 5.13 An example of clustering via modified distance option

5.10 Results and Findings

In this section, results of agent-based solution strategies that are developed to solve variants of dynamic TSP are discussed. These strategies allow local and parallel search, which actually helps to save from time and memory requirements that are source of effective solution of those in NP hard problem class. This section discusses findings of specific agent-based solution policies/strategies that are providing promising solutions dynamic TSP and its variants. It is remarkable to state one more time that, promising solutions mentioned here may not equally mean the optimal solutions, it is rather the feasible solutions obtained in affordable time with affordable costs and in an acceptable level. In addition to all of those remarks, all

combinations provided via the prepared software are not run, only the selected ones are run.

5.10.1 Basics and assumptions of the compared models

For comparison purposes, findings of the proposed agent-based strategies are compared with static heuristic solutions obtained via GDA. Those comparisons are performed when the total number of cities are (becomes) equal 0 in mode 5 by the time. GDA that has exactly the same parameters within agent-based GDA is used for solving above defined states and solutions are recorded. Since solutions obtained via GDA, are only valid for certain states, they will be named as "frozen solutions" through this chapter. Thereby solutions obtained with continuous agent-based system are compared with the frozen solutions.

It is remarkable to state here that, "frozen solutions" are the results obtained for hypothetic cases where change is not considered for awhile and the model is rebuilt and solved from the beginning like a new problem. It is also a known fact that, it is not an easy task to setup a model (like finding the suitable parameters) and resolve it using heuristics. Furthermore, obtaining solutions via heuristics in dynamic environment has no practical value since "*stopping delivery process until a good solution is obtained"* may be very difficult and costly in real life.

There is also one more crucial difference between the hypothetic case and the continous agent-based models. In the frozen solutions; all information is assumed to in hand (that is called as perfect information), on the other hand; in agent-based continuous systems, number of cities are dynamic and information is inperfect which is much realistic indeed.

Normally "frozen results" is expected to be better since GDA handle the problem from the beginning and looks for different combinations that are creating less cost. As Barbati et. al, (2012) state from the point of view of the *quality of solution*, since agent-based approaches are distributed, they do not have a global view of the state of the system, which is often necessary in order to find a truly good solution. Therefore, the quality of the solution provided by a classical heuristic could be better. On the

other hand, a heuristric like GDA may loose its capability to produce good results when parameters are kept constant as performed in this study. Total number of iterations terminating the search of a heuristic for a 50 city problem may not be adequate to search a 100 city problem. This is also an evidence of the certain advantage of ABM through negotiation. Providentially, in the proposed agent-based strategies, parameter setting have no considerable effect after the obtaining the first solution since other solutions are obtained via negotiation of agents. This feature can be accepted as the superiority of obtaining agent-based solutions to highly dynamic types of TSP. As Angelelli (2011) state, lack of information on future events creates the need for algorithms able to take quick decisions on the basis of current information possibly making some guess on future.

In this respect, comparisions given in the next sections can be read as follows. Heuristic based approaches is normally expected to produce promising results compared to agent-based negotiation strategies, while they may require unaffordable amount of resources. However, if the cost of parameter setting is waived/omitted (other costs remaining as they are) agent-based negotiation strategies even produce better results.
The results are presented in figures and tables in the subsequent sections, as averages from 100 runs of algorithms.

5.10.2 Findings of problem Type 1

Since there are several input setting that is possible to perform GDA, after several trials, it has been understood that, TSP perform better where beta=0.001 and neighborhood as 30 for 20 cities. Total number of 1000 iterations and 800 non-improving iterations have been considered as the terminating criteria. As it is mentioned in problem assumptions, these set of parameters are kept constant for 40 and 60 city problems. New event creation ratio is defined as 0.002.

Table 5.2 and Table 5.3 represent the comparison table for n=20 cities where cities lay in a wide environment (1000*1000). Frozen GDA, outperforms Agent-Based GDA until total number of cities are equal to 25. When total number of cities are equal to 25, Frozen GDA looses its cabability to produce promising results with the

given parameter set. Frozen GDA outperforms agent-competion in all of the given cases.

Table 5.2 Comparision of AB-GDA with frozen GDA for initial 20 cities

Number of City	Agent-Based GDA	Frozen Solutions of GDA
Initial (20)	3329.57	3329.57
20	3335.19	3318.49
25	4315.28	4722.51
30	4521.15	4743.18
30	4583.99	4765.45

Table 5.3 Comparision of AC with frozen GDA for initial 20 cities

Number of City	Agent Competition	Frozen Solutions of GDA
Initial (20)	3755.99	3329.57
20	3746.98	3678.97
25	5042.73	4354.83
30	5200.01	4783.33
30	5100.69	4788.61

Table 5.4 and Table 5.5 represent the comparison table for n=40 cities where cities lay in a large environment (1000*1000). Frozen GDA, outperforms Agent-Based GDA until total number of cities are equal to 45. When total number of cities are equal to 45, Frozen GDA looses its cabability to produce promising results with the given parameter set. Agent-competion strategy outperforms Frozen GDA for all of the given cases.

Table 5.4 Comparision of AB-GDA with frozen GDA for initial 40 cities

Number of City	Agent-Based GDA	Frozen Solutions of GDA
Initial (40)	6248.12	6248.12
40	6094.70	6077.82
45	6541.00	6803.76
50	6639.25	7497.71
55	6914.30	8071.90
60	6919.78	8690.65

Table 5.5 Comparision of AC with frozen GDA for initial 40 cities

Number of City	Agent Competition	Frozen Solutions of GDA
Initial (40)	5370.02	6248.12
40	5191.64	5972.94
45	5570.30	6883.92
50	5676.59	7908.72
55	6028.38	8071.96
60	6248.71	9061.53
60	6315.86	8307.60

Table 5.6 and Table 5.7 represent the comparison table for n=60 cities. Frozen GDA, outperforms Agent-Based GDA until total number of cities are equal to 75. When total number of cities are equal to 75, Frozen GDA looses its cabability to produce promising results with the given parameter set. Agent-competion strategy outperforms Frozen GDA for all of the given cases.

Table 5.6 Comparision of AB-GDA with frozen GDA- initial number of cities: 60

Number of City	Agent-Based GDA	Frozen Solutions of GDA
Initial (60)	9656.18	9656.18
60	9706.94	8918.42
65	9886.77	9805.00
70	10189.93	10182.00
75	10272.52	11084.79
75	10215.71	11447.47

Table 5.7 Comparision of AC with frozen GDA for initial 60 cities

Number of City	Agent Competition	Frozen Solutions of GDA
Initial (60)	7295.15	9656.18
60	7227.51	8861.89
65	7519.06	10032.27
70	7854.76	11167.52
75	8070.52	11529.68

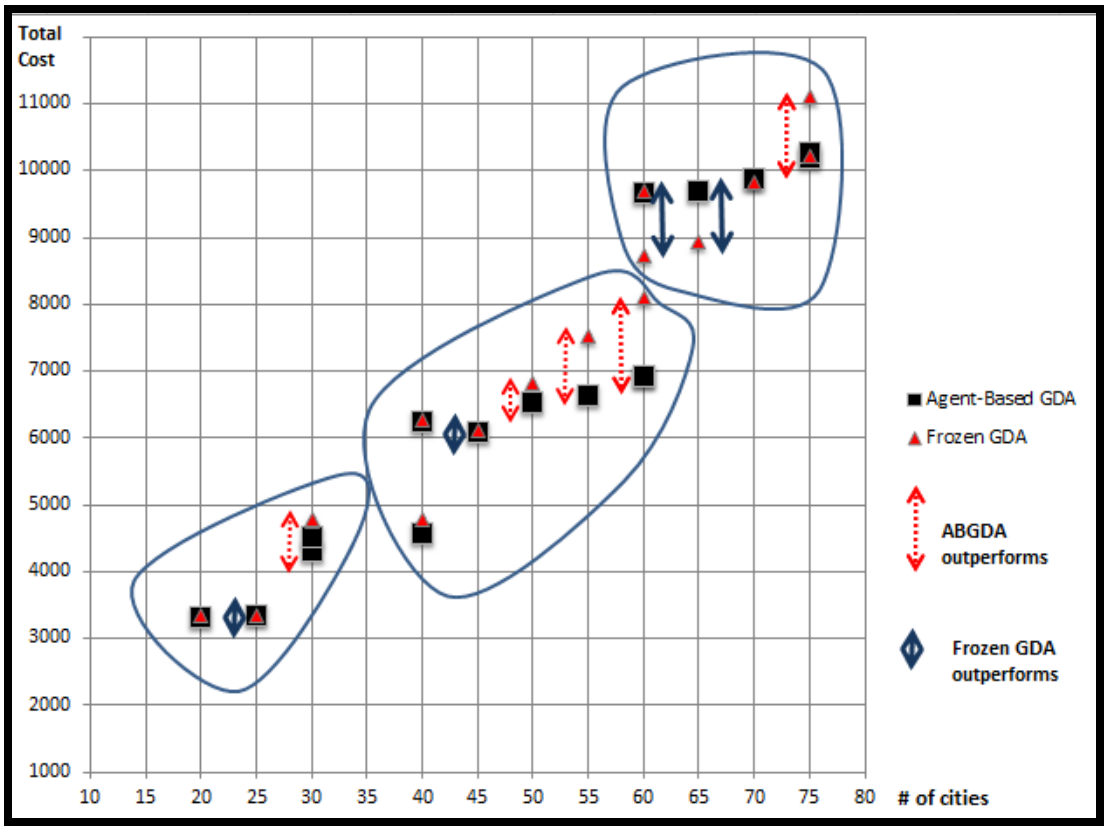

Figure 5.14 Comparision of agent-based model with ideal case

As it is demonstrated in Figure 5.14, increasing the number cities make the results better for agent-based strategies that are proposed in this thesis. This state can also be linked with the use of constant GDA parameters and adaptation ability of the agent-based solutions to the dynamism.

To test the significiance of the differences between agent-based GDA solution strategy and Frozen GDA results, Paired-T test is performed and the findings are as described in Table 5.8. The performances of the strategies are compared separately, as a paired t-test experiment. Since the *p*-value for the experiment is almost equal to zero (0.03), we can conclude that there is a statistically significant difference between the results of these strategies. Since the average of the total cost of the Agent-Based GDA is lower than the Frozen GDA, we can conclude that the Agent-Based GDA has a better performance compared to the Frozen GDA strategy.

Table 5.8 Results of paired T-test

Group	Agent Based GDA	Frozen GDA
Mean	7021.7871	7374.2953
SD	2521.9774	2631.7870
SEM	611.6694	638.3021
N	17	17

Paired *t* test results

P value and statistical significance:

The two-tailed P value equals 0.0332

By conventional criteria, this difference is considered to be statistically significant.

Confidence interval:

The mean of Agent Based GDA minus Frozen GDA equals -352.5082

95% confidence interval of this difference: From -673.1769 to -31.8395

5.10.3 Findings of problem Type 2 and Type 3

These problem types were defined in Section 5.2. In these defined types, target is finding sutiable the cities from each region to minimize the grant tour. Problems have high dynamism due random addition and deletion of cities. In this regard, total costs are computed and found as illustrated in Table 5.9 and Table 5.10. In the problem type 2,type 3, type 4 and type 5, agent-based competition has not been considered due to its relative failure to compete with frozen solutions for type 1 and type 2 problems.

Table 5.9 Total cost comparisions for GTSP with known regions initial number of cities: 20

Total # of cities	Cost of Agent-Based GDA	Cost of Frozen GDA
20	3552.49	3552.49
20	3538.13	3717.66
25	4695.97	4391.74
30	4711.85	5131.36
30	4755.15	5248.07
35	5122.64	5763.65
40	5719.49	6869.67
40	5578.37	6770.17

Table 5.10 Total cost comparisions for GTSP with known regions initial number of cities: 40

Total # of cities	Cost of Agent-Based GDA	Cost of Frozen GDA
40	7411.17	7411.17
40	7201.71	7043.80
45	7777.42	7452.53
50	7721.08	7627.40
55	8139.78	7917.90
60	8344.11	8320.25
60	8279.17	8543.28
60	8379.82	8725.38
60	8528.95	8751.76

Table 5.11 Total cost comparisions for GTSP with known regions initial number of cities: 60

Total # of cities	Cost of Agent-Based GDA	Cost of Frozen GDA
60	10371.31	10371.31
60	10118.48	9920.52
65	10192.72	10867.99
70	10353.81	11327.04
75	9984.31	11759.15
75	10064.20	12129.93
75	10094.63	11672.39

Table 5.12 Total cost comparisions for GTSP with region determination initial number of cities: 20

Total # of cities	Cost of Agent-Based GDA	Cost of Frozen GDA
20	3588.48	3588.48
20	3616.50	3291.10
25	4855.06	5378.51
30	5409.92	5487.89
30	5466.38	6605.12
35	5790.43	7118.21
40	6201.27	8601.07
40	6104.20	7743.94
40	5824.13	7166.22

Table 5.13 Total cost comparisions for GTSP with region determination initial number of cities: 40

Total # of cities	Cost of Agent-Based GDA	Cost of Frozen GDA
40	8250.17	8250.17
40	8103.70	8476.14
45	8699.32	9854.63
50	8966.07	10818.85
55	9372.36	11853.76
60	9653.06	12911.16
60	9437.73	13472.45
60	9644.53	13816.87

Table 5.14 Total cost comparisions for GTSP with region determination initial number of cities: 60

Total # of cities	Cost of Agent-Based GDA	Cost of Frozen GDA
60	13108.52	13108.52
60	13108.32	12868.68
65	13209.13	15471.39
70	13381.71	16835.49
75	13484.00	16535.39
75	13053.38	18278.45
75	13030.01	18339.81
80	13163.28	17969.45

5.10.4 Findings of problem Type 4 and Type 5

These problem types were defined with details in Section 5.2. In these defined types, target is finding sutiable the cities from each region to complete the grant tour and the local tours to be visited through the visit of each city of each region by another vehicle to minimize the total cost. Problems have high dynamism due random addition and deletion of cities. In this regard, total costs are computed and found as illustrated in Table 5.15 through Table 5.20.

Table 5.15 Total cost comparisions for GTSP+TSP with known regions initial number of cities: 20

Total # of cities	Cost of Agent-Based GDA	Local Tour	Grand Cost of Agent-Based GDA	Grand Tour Cost of Frozen GDA
20	4344.80	792.30	3552.50	3552.49
20	4273.28	792.30	3480.98	3470.24
25	5649.61	1093.72	4555.89	5154.03
30	5573.61	1093.72	4479.89	4764.19
35	6071.71	1282.46	4789.25	4505.00
35	6341.36	1282.46	5058.90	5197.44
40	7494.91	972.87	6522.04	5608.32
45	10262.7	972.87	9289.79	6861.55

Table 5.16 Total cost comparisions for GTSP+TSP with known regions initial number of cities: 40

Total # of cities	Cost of Agent-Based GDA	Local Tour	Grand Cost of Agent-Based GDA	Grand Tour Cost of Frozen GDA
40	23508.3	16097.1	7411.17	7411.17
40	23429.8	16162.6	7267.19	6454.43
45	24412.4	16378.4	8034.01	7361.59

Table 5.17 Total cost comparisions for GTSP+TSP with known regions initial number of cities: 60

Total # of cities	Cost of Agent-Based GDA	Local Tour	Grand Cost of Agent-Based GDA	Grand Tour Cost of Frozen GDA
60	45741.7	35370.4	10371.3	9728.42
60	45541.8	35208.1	10333.7	9278.14
65	45938.1	35371.7	10566.4	10559.7
65	45979.3	35549.9	10429.4	11424.8
70	45893.7	35447.6	10446.1	11702.4
70	46030.4	35619.3	10411.1	12145.5
75	46030.4	35619.3	10411.1	11552.5
75	46713.6	35924.1	10789.5	12721.2
80	46555.2	35840.8	10714.4	12260.5

Table 5.18 Total cost comparisions for GTSP+TSP with region determination initial number of cities: 20

Total # of cities	Cost of Agent-Based GDA	Local Tour	Grand Cost of Agent-Based GDA	Grand Tour Cost of Frozen GDA
20	6138.66	2550.18	3588.48	3588.48
20	6141.72	2550.18	3591.54	3540.15
25	6470.07	2474.87	3995.20	4013.17
30	6517.25	2474.87	4042.38	4002.16
35	6517.15	2474.87	4042.28	3953.89

Table 5.19 Total cost comparisions for GTSP+TSP with region determination initial number of cities: 40

Total # of cities	Cost of Agent-Based GDA	Local Tour	Grand Cost of Agent-Based GDA	Grand Tour Cost of Frozen GDA
40	10459.9	2209.72	8250.17	8250.17
45	10687.4	2209.72	8477.64	8506.86
50	10896.8	2209.72	8687.10	9965.62
55	10919.8	2209.72	8710.08	9310.65

Table 5.20 Total cost comparisions for GTSP+TSP with region determination initial number of cities: 60

Total # of cities	Cost of Agent-Based GDA	Local Tour	Grand Cost of Agent-Based GDA	Grand Tour Cost of Frozen GDA
60	17386.7	4278.14	13108.5	12494.4
60	17363.5	4278.14	13085.3	12781.9
65	17573.7	4278.14	13295.6	13292.5
65	17651.9	4278.14	13373.8	15087.2
70	17802.3	4252.07	13550.2	17241.3
70	17697.2	4252.07	13445.1	15385.2
75	17541.0	4252.07	13288.9	19411.5
75	17490.9	4252.07	13238.8	18220.9
80	17607.6	4252.07	13355.6	17874.1

As it is demonstrated through all of the given figures, increasing the number cities make the results better for agent-based strategies that are proposed in this thesis. This state is linked with the use of constant GDA parameters and adaptation ability of the agent-based solutions to the dynamism.

5.11 Concluding Remarks

A solution strategy has been developed that is allowing local and parallel search, which actually helps to save from time and memory requirements. Cities in the problem domain are divided into regions (regions describe a part of physical environment where cities with similar coordinates are gathered) using a special algorithm and thereby solutions are searched in each of regions parallel to each other and simultaneously. For this purpose, in addition to the vehicle agent, a manager agent and several region agents are created which can make autonomous decisions for their own regions and for the whole domain of the problem.

As it is demonstrated in all given results, increasing the number cities make the results better for agent-based strategies that are proposed in this thesis. This state can also be linked with the use of constant GDA parameters and adaptation ability of the agent-based solutions to the dynamism.

The performances of the strategies are compared separately, as a paired t-test experiment. Since the *p*-value for the experiment is almost equal to zero (0.03), we can conclude that there is a statistically significant difference between the results of these strategies. Since the average of the total cost of the Agent-Based GDA is lower than the Frozen GDA, we can conclude that the Agent-Based GDA has a better performance compared to the Frozen GDA strategy.

CHAPTER 6
CONCLUSION

6.1 Introduction

This PhD thesis, attempted to fulfill two vacancies in the area of agent-based modeling. One is about the all agent-based applications that have lied in a wide range of area. Although some dynamic optimization problems have common features, there is not a systematic scheme that represents similarities and differences of those applied solution strategies. Therefore, a researcher studying and focusing on a specific DOP may have many troubles to classify existing studies and their solution approaches. Systematic observations on several practical applications in the area have inspired a solution regarding the development of such a scheme that is capable to indicate features of a problem and solution strategy that is followed. Several sample problems were also used to test usability of this scheme.

The second vacancy that this PhD thesis focuses is application of agent-based modeling approaches to dynamic travelling salesman problem where number of cities can immediately change. In this respect several new variant of TSP are introduced and solved via different agent-based strategies. Promising solution have be obtained and statistically analyzed.

Details of the proposed solution approaches that are regarding both of the described vacancies in the area can be found in next section.

6.2 Thesis Findings

In the light of the adjustments discussed in the several chapter of thesis, this research study within this PhD thesis focuses on agent-based modeling and dynamic optimization problems.

In this respect, six chapters were presented. In Chapter 2, DOPs that are solved via agent-based modeling were discussed with details. The chapter attempted to be a warm-up chapter for the people who are not familiar with DOPs, agent-based modeling and the fundamental concepts of them.

In Chapter 3, a new representation scheme called ABDOPSS, which was constructed for standard representation and classification of agent-based solution approaches employed for DOPs, was presented. ABDOPSS was also exemplified to present different applications in the area.

Since there are numerous software packages that are widely used for agent-based modeling it has been a necessity to introduce them. Therefore, Chapter 4 was devoted to those software packages.

Chapter 5 presented five different types of Dynamic Travelling Salesman Problems (DTSP) that are highly dynamic due to random change in number of cities. In those defined problems, new cities arrive to the system and sometimes they disappear from it. In this respect, two main agent-based solution strategies were proposed for the solution of the defined problem. One is competition of agents without a heuristic and the other is competition of agents using Great Deluge Algorithm (GDA).

Finally, this chapter, Chapter 6 discusses the possible benefits and handicaps of the solutions provided by this PhD study. Future study extensions for new students and the people studying in the area, is also declared in the next section of this chapter.

6.3 Closure

The proposed solutions in this PhD thesis are all novel to the literature. They have been presented in several conferences and published in several different journals. These solutions are also expected to be useful in the industrial area. It is also anticipated that, some ideas and implementations presented in this thesis can create new research directions for other researchers.

The contribution of this thesis can be summarized in two folds:

1) The classification scheme (ABDOPPS: Agent Based Dynamic Optimization Problem Solution Strategy) for agent-based approaches is new to the literature and it is expected to be beneficial to researchers in many ways. Similarities of the features located in ABDOPPS can be used to define classes of solution strategies by their descriptions. In this regard, classes of the problems may orient researchers to focus on certain strategies. Using the dynamism related features of the corresponding DOPs presented in ABDOPPS, unpredictability levels of certain problems can be determined and be used to reclassify problems. These representation forms can also be used to discover the role of presented features and their importance for solution quality.

2) Thesis is the provision and extension of knowledge on how to analyze and design new computational models for travelling salesman integrating the city addition and deletion at any time during the visits. TSP variants defined through the thesis are also new to literature.

Several feature studies can also be performed as continuum of this PhD thesis. First, ABDOPSS can be enriched with the addition of several different factors and features. Some abstraction mechanisms can also be used make it much easier to be used. For extending the use of it, authors using agent-based approaches for DOPs can be asked to use it for their problems. Some experimental designs can be performed check whether similar problems use similar features or not.

Future extensions of DTSP can also be produced using several different agent-based heuristics. Those modeled TSP versions are unsolved problem from industry was investigated in this thesis. Those new variants of DTSP can also be search topic for many other researchers. Existing TSP models can also be converted into vehicle routing problems easily and some other conclusions can be extracted.

REFERENCES

Abbas, H. A., Bacardit, J., Butz, V. M., LLorà, X. (2004). Online adaptation in learning classifier systems: stream data mining. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IlliGAL Report No. 2004031.

Aggestam, L., Söderström, E. (2005). Managing critical success factors in a B2B Setting. *In: Proceedings of the IADIS International Conference e-Commerce*. 101-108.

Ahmad, R., Lee, Y. C., Rahimi, S., Gupta, B. (2007). A Multi-Agent Based Approach for Particle Swarm Optimization. *Integration of Knowledge Intensive Multi-Agent Systems, KIMAS 2007*. 267-271.

Allmendinger, R., Knowles, J. (2010). Evolutionary optimization on problems subject to changes of variables. *In: Schaefer RC, Kolodziej J, Rudolph G (eds) Parallel Problem Solving from Nature-PPSN XI, 6239*. Springer, Berlin, Heidelberg. 151-160.

Angelelli, E., Mansini, R., & Vindigni, M. (2011). Look-ahead heuristics for the dynamic traveling purchaser problem. *Computers & Operations Research*. **38**(12), 1867-1876.

Barbati, M., Bruno, G., Genovese, A. (2012). Applications of agent-based models for optimization problems: A literature review. *Expert Systems with Applications*. **39** (5), 6020-6028.

Baykasoğlu, A. (2012). Design optimization with chaos embedded great deluge algorithm. *Applied Soft Computing*. **12**(3), 1055-1067.

Baykasoğlu, A., U. Durmuşoğlu, Z. D. (2012). A classification scheme for agent based approaches to dynamic optimization. *Artificial Intelligence Review*, 1-26. doi:10.1007/s10462-011-9307-x (Article in Press).

Baykasoğlu, A., U. Durmuşoğlu, Z. D., Görkemli, L. (2011). Etmen tabanlı benzetim: ANYLOGIC™ yazılımı ve örnek bir uygulama. Endüstri Mühendisliği Yazılımları ve Uygulamaları Kongresi, İzmir, 30 Eylül-01/02 Ekim, TMMOB Makina Mühendisleri Odası Yayın No: E/2011/559, 197-204.

Baykasoğlu, A., U. Durmuşoğlu, Z. D. (2011). Dynamic optimization in a dynamic and unpredictable world. *In: Proceedings of Portland International Conference on Management of Technology (PICMET'11)*, Portland, Oregon, USA. 2312-2319.

Baykasoğlu, A., U. Durmuşoğlu, Z. D., Görkemli, L. (2011). Solving vehicle deployment planning problem by using agent based simulation modeling. *In: Proceedings of 2nd International Symposium on Computing in Science & Engineering*, Kuşadası, Aydın, Turkey. 338-340.

Berro, A., Duthen, Y. (2001). Search for optimum in dynamic environment: An efficient agent-based method. *In: Genetic and Evolutionary Computation Conference.* Workshop Program, San Francisco, California. 51-54.

Billiau, G., Ghose, A. K (2008). Robust, flexible multi-agent optimization using SBDO. Ins.l., Decision Systems Lab /Center for Software Engineering.

Bontoux, B., Artigues, C., & Feillet, D. (2010). A Memetic Algorithm with a large neighborhood crossover operator for the Generalized Traveling Salesman Problem. *Computers & Operations Research*. 37(11), 1844-1852.

Borst, S. C., Buvaneswari, A., Drabeck, L. M, Flanagan, J. M et al. (2005). Dynamic optimization in future cellular networks. *Bell Labs Technical Journal*. **10**(2), 99-119.

Boughaci, D., Drias, H. (2005). Taboo search as an intelligent agent for bid evaluation. *International Journal of Internet and Enterprise Management*. **3**, 170-186.

Bui, L. T, Michalewicz, Z., Parkinson, E., Abello, E. M. (2011). Adaptation in dynamic environments: a case study in mission planning. *IEEE Transactions on Evolutionary Computation* (Accepted Manuscript)

Burke, E. Bykov, Y. Newall, J., Petrovic, S. (2004). A time-predefined local search approach to exam timetabling problems. *IIE Transactions.* **36**, **(**6) 509-528.

Calégari, P., Coray, G., Hertz, A., Kobler, D., Kuonen, P. (1999). A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*. **5**(2), 145-158.

Campbell, A. M. (2006). Aggregation for the probabilistic traveling salesman problem. *Computers & Operations Research*. **33** (9), 2703-2724.

Chang, T.S. Wah Wan Y., OOI, W. T (2009). A stochastic dynamic traveling salesman problem with hard time Windows. *European Journal of Operational Research*. **198** (3), 748-759.

Corchado, J. M., Glez-Bedia, M., De Paz, Y., Bajo, J., De Paz, J. F. (2008). Replanning mechanism for deliberative agents in dynamic changing environments. *Computational Intelligence*. **24**(2), 77-107.

Cowling, P. I., Keuthen, R. (2005). Embedded local search approaches for routing optimization. *Computers & Operations Research*. **32**(3), 465–490.

Cruz, C., Gonzá J. R, Pelta, D. A., (2010). Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*. **15**(7), 1427-1448.

Dimitrijević, V., Šarić, Z. (1997). An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Information Sciences*. **102**(1-4), 105-110.

Dueck, G. (1993). New optimization heuristics. The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*. **104**, 86-92.

Eyckelhof, C. J., Snoek, M. (2002). Ant Systems for a Dynamic TSP-Ants caught in a traffic jam. In3rd International Workshop on Ant Algorithms.In M. Dorigo, G. Di Caro, & M. Sampels (Eds.), *Ant Algorithms*, Lecture Notes in Computer Science (**2463**, 88–99). Springer Berlin / Heidelberg.

Fisher, M., Bordini, R. H, Hirsch, B., Torroni, P. (2007). Computational logics and agents: a road map of current technologies and future trends. *Computational Intelligence*. **23**(1), 61-91.

Flood, M. M. (1995). The traveling salesman problem. *Operation Research*. **4**, 61-78.

Garcia, A. F., De Lucena, C. J. P., Cowan, D. D. (2004). Agents in object-oriented software engineering. *Software Practice and Experience*. **34**(5), 489-521.

García-Montoro, C., Vivancos, E., García-Fornes, A. and Botti, V. J. (2007). A Software Architecture-Based Taxonomy of Agent-Oriented Programming Languages. *Languages, Methodologies and Development Tools for Multi-Agent Systems*. 128-142.

Genesereth, M. R, Ketchel, S. P. (1994). Software agents. C*ommunication of the ACM*. **37**(7), 48-53.

González, J. R, Masegosa, A. D, García, I. (2010). A cooperative strategy for solving dynamic optimization problems. *Memetic Computing*. **3**(1), 3-14.

Guan, S. U, Chen, Q., Mo, W. (2005). Evolving dynamic multi-objective optimization problems with objective replacement. *Artificial Intelligence Review*. **23**, 267-293.

Hadeli, P., Valckenaers, P., Kollingbaum, M., Van Brussel, H. (2004). Multi-agent coordination and control using stigmergy. *Computers in Industry*. **53**(1), 75-96.

Hanna, L., Cagan, J. (2009). Evolutionary multi-agent systems: an adaptive and dynamic approach to optimization. *Journal Mechanical Design*. **131**(1), **011010**-1-011010-8.

Homayounfar, H., Areibi, S., Wang, F. (2003). An advanced island based GA for optimization problems. *In Proceedings of the International DCDIS Conference on Engineering Applications and Computations*. 46-51.

http://en.wikipedia.org/wiki/AnyLogic

http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software.

http://www.dynamic-optimization.org. Accessed 30 August 2011.

Huang, Z. C., Hu, X. L., Chen, S. D. (2001). Dynamic traveling salesman problem based on evolutionary computation. *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on.* **2**, 1283-1288.

Huhns, M. N, Stephens, L. M (1999). Multi-agent systems and societies of agents. G.Weiss (ed.), *Multi-Agent Systems*. MIT Press.

Jennings, N. R, Faratin, P., Lomuscio, A. R, Parsons, S., Sierra, C., Wooldridge, M. (2001). Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*. **10**(2), 199-215.

Jian, Y. (2003). Solving integer programming by evolutionary soft agent. *Wuhan University Journal of Natural Sciences*. **8**, 283-286.

Jiang, D., Han, J. (2008). Real time multi-agent decision making by simulated annealing. In TechOpen, Simulated Annealing.

Jin, Y. (2004). A tutorial on evolutionary computation in dynamic and uncertain environments. In CEC'04, Portland, USA.

Jou, S. H, Kao, S. J (2002). Agent-based infrastructure and an application to internet information gathering. *Knowledge and Information Systems*. **4**(1), 80-95.

Jung, Y., Kim, M., Masoumzadeh, A., Joshi, J. B. D. (2011). A survey of security issue in multi-agent systems. Artificial Intelligence Review. doi:10.1007/s10462-011-9228-8.

Kang, L., Zhou, A., McKay, B., Li, Y., Kang, Z. (2004). Benchmarking algorithms for dynamic travelling salesman problems. *Evolutionary Computation, CEC2004.* **2**, 1286 -1292.

Karlsson, M., Ygge, F., Andersson, A. (2007). Market-based approaches to optimization. *Computational Intelligence*. **23**(1), 92-109.

Kulkarni, A. J, Tai, K. (2010). Probability Collectives: A multi-agent approach for solving combinatorial optimization problems. *Applied Soft Computing*. **10**(3), 759-771.

Lepagnot, J., Nakib, A., OulHadj, H., Siarry, P. (2010). A new multi-agent algorithm for dynamic continuous optimization. International Journal of Applied Metaheuristic Computing. **1**(1), 16-38.

Li, C., Yang, M., Kang, L. (2006). A New Approach to Solving Dynamic Traveling Salesman Problems. In T. D. Wang, X. Li, S.-H. Chen, X. Wang, H. Abbass, H. Iba, G.-L. Chen, et al. (Eds.), *Simulated Evolution and Learning*, Lecture Notes in Computer Science (4247, 236-243). Springer Berlin / Heidelberg.

Li, S., Li, J. Z. (2009). A multi-agent-based hybrid framework for international marketing planning under uncertainty. *International Journal of Intelligent Systems in Account and Finance Management*. **16**, 231-254.

Liu, J. Zhong, W., Jiao, L. (2006). A multiagent evolutionary algorithm for constraint satisfaction problems, *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics: A Publication of the IEEE Systems, Man, and Cybernetics Society*. **36**, 54-73.

Liu, J., Zhong, W. and Jiao, L. (2010). A multi-agent Evolutionary Algorithm for Combinatorial Optimization Problems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*. **40**, 229-240.

Liu, X., Xu. K., Liu, H. (2006). A multi-agent particle swarm optimization framework with applications. *In Proceeding of 1st International Symposium on Pervasive Computing and Applications*. 1-6.

Lung, R. I, Dumitrescu, D. (2009). Evolutionary swarm cooperative optimization in dynamic environments. *Natural Computing*. **9**(1), 83-94.

Macal, C. M., North, M. J. (2010). Tutorial on agent-based modelling and Simulation. *Journal of Simulation*. **4**, 151-162.

Máhr, T., Srour, J., De Weerdt, M., Zuidwijk, R. (2010). Can agents measure up? A comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty. *Transportation Research Part C: Emerging Technology*. **18**(1), 99-119.

Mataric, M. J. (1995). Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous System*. **16** (2-4), 321-331.

Mavrovouniotis, M., Yang, S. (2010). A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Computing*. ***15***(7), 1405-1425.

Neches, R., Fikes R., Finin, T., Gruber, T., Patil, R., Senator, T., Swartout, W. R. (1991). Enabling technology for knowledge sharing. *Artificial Magazine*. **12**(3), 36–56.

Newkirk, H. E., Lederer, A. L. (2006). Incremental and comprehensive strategic information systems planning in an uncertain environment. *IEEE Trans on Engineering Management*. **53**(3), 380-394.

O'Hare, G. M. P., O'Grady, M. J, Tynan, R., Muldoon, C., Kolar, H. R., Ruzzelli, A. g., Diamond, D., Sweeney, E. (2007). Embedding intelligent decision making within complex dynamic environments. *Artificial Intelligence Review*. **27**, 189-201.

Panzarasa, P., Jennings, N. R., Norman, T. J. (2001). Social mental shaping: modeling the impact of sociality on the mental states of autonomous agents. *Computational Intelligence*. **17**(4), 738-782.

Parunak, H. V. D. (1997). Go to the ant: Engineering principles from natural multi-agent systems. *Annals of Operations Research*. **75**, 69-101.

Pelta, D., Cruz, C., Verdegay, J. L. (2009). Simple control rules in a cooperative system for dynamic optimization problems. *International Journal Genetic Systems*. **38**, 701-717.

Pelta, D., Cruz, C., González, J. R. (2009). A study on diversity and cooperation in a multi-agent strategy for dynamic optimization problems. *International Journal Intelligent Systems.* **24**(7), 844-861.

Persson, J. A., Davidsson, P., Johansson, S. J., Wernstedt, F., Center, S., Ronneby, S. (2005). Combining agent-based approaches and classical optimization techniques. *Third European Workshop on Multi-Agent Systems*.

Psaraftis, H.N. (1988). Dynamic vehicle routing problems. In Vehicles Routing: Methods and Studies, B. L. Golden and A. A. Assad (eds), Elsevier Science Publishers.

Raman, N., Talbot, F. B. (1993). The job shop tardiness problem: a decomposition approach. *European Journal of Operations Research*. **69**(2), 187-199.

Razavi, S. N., Gaud, N., Mozayani, N., Koukam, A. (2001). Multi-agent based simulations using fast multipole method: application to large scale simulations of flocking dynamical systems. *Artificial Intelligence Review*. **35**, 53-72.

Sanchez, R. (1997). Preparing for an Uncertain Future: Managing Organizations for Strategic Flexibility. *International Studies of Management & Organization*. **27**, 71-94.

Satoh, K., Inoue, K., Iwanuma, K., Sakama, C. (2000). Speculative computation by abduction under incomplete communication environments. *In: Proceedings of Fourth International Conference on Multi-Agent Systems*. 263-270.

Savelsbergh, M. (1984). Local Search in Routing Problems with Time Windows. Report OS-R8409, Centre for Mathematics and Computer Science.

Shigehiro, Y., Kumura, N., Masuda, T. (2002). An agent-based method for combinatorial optimization problems. *SICE 2002. Proceedings of the 41st SICE Annual Conference*. **2,** 1309-1312.

Takahashi, Y. (1998). A mathematical framework for solving dynamic optimization problems with adaptive networks. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*. ***28***(3), 404-416.

Tan, M. (1993). Multi-agent reinforcement learning: independent vs. cooperative agents. *In: Proceedings of the Tenth International Conference on Machine Learning*. 330-337.

Tang, K., Kumara, S. R. T, Yee, S. T, Tes, J. (2004). Wireless-based dynamic optimization for load makeup using auction mechanism. *Industrial Engineering Research Conference (IERC)*.

Teo, T. S. H, King, W. R. (1997). Integration between business planning and information systems planning: an evolutionary-contingency perspective. *Journal of Management Information System*. **14**(1), 185-214.

Tsui, K. C., Liu, J. (2003). Multi-agent diffusion and distributed optimization. *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, New York, NY, USA: ACM, 169–176.

Vanden Bergh, F., Engelbrecht, P. A. (2004). A Cooperative Approach to Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation*. **8**, 225-239.

Volberda, H. W. (1997). Building flexible organizations for fast-moving markets. *Long Range Planning*. **30**, 169-183.

Voos, H. (2009). Agent-based distributed resource allocation in continuous dynamic systems. *InTechOpen*. Multi-agent Systems.

Wagner, S., Affenzeller, M., Ibrahim, I. K. (2003). Agent-based problem solving: the ant colonies metaphor. In Proceedings of the Fifth International Conference on Information Integration and Web-Based applications & Services. 317-323.

Wang, D., Shixin, L. (2010). An agent-based evolutionary search for dynamic travelling salesman problem. In: Proceedings of WASE International Conference on Information Engineering. 111-114.

Wang, S., Xi, L., Zhou, B. (2008). FBS-enhanced agent-based dynamic scheduling in FMS. Engineering Applications Artificial Intelligence. **21**(4), 644–657.

Wang, Y. C, Usher, J. M. (2002). An agent-based approach for flexible routing in dynamic job shop scheduling. In: Proceedings of the 11th Industrial Engineering Research Conference.

Wangermann, J. P., Stengel, R. F. (1999). Optimization and coordination of multiagent systems using principled negotiation. *Journal of guidance control and dynamics*. **22**, 43-50.

Wilensky, U. (1999), NetLogo, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. http://ccl.northwestern.edu/netlogo.

Xiang, W., Lee, H. P. (2008). Ant colony intelligence in multi-agent dynamic manufacturing scheduling. *Journal Engineering Applications of Artificial Intelligent*. **21**(1), 73–85.

XJ Technologies (2012). AnyLogic home page. http://www.xjtek.com/.

Yan, Y., Yang, S., Wang, D., Wang, D. (2010). Agent based evolutionary dynamic optimization. In: Sarker, RA, Ray T (Eds) Agent-Based Evolutionary Search, Chapter 5, Springer, Heidelberg. 97-116.

Yang, S. (2007). Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Yang S, Ong YS, Jin Y (eds), Evolutionary Computation in Dynamic and Uncertain Environments, Studies in Computational Intelligence, 51, Springer, Heidelberg, 3-28.

Zhong, W., Liu, J., Xue, M., Jiao, L. (2004). A multiagent genetic algorithm for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics*. **34**, 1128-1141.

Zhou, R., Lee, H. P., Nee, A. Y. C. (2008). Simulating the generic job shop as a multi-agent system. *International Journal of Intelligent System Technology and Applications*. **4**, 5-33.

Zhou, Z., Chan, W. K., Chow, J. H. (2009). Agent-based simulation of electricity markets: a survey of tools. *Artificial Intelligence Review*. **28** (4), 305-342.

APPENDIX A

```
package dtsp;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.Currency;
import java.util.Date;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.util.Locale;
import java.util.Map;
import java.util.Random;
import java.util.Set;
import java.util.SortedMap;
import java.util.SortedSet;
import java.util.Stack;
import java.util.Timer;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.Vector;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import static java.lang.Math.*;
import static com.xj.anylogic.engine.presentation.UtilitiesColor.*;
import static com.xj.anylogic.engine.presentation.UtilitiesDrawing.*;
import static com.xj.anylogic.engine.HyperArray.*;
import com.xj.anylogic.engine.*;
import com.xj.anylogic.engine.analysis.*;
import com.xj.anylogic.engine.connectivity.*;
import com.xj.anylogic.engine.connectivity.ResultSet;
import com.xj.anylogic.engine.connectivity.Statement;
import com.xj.anylogic.engine.presentation.*;
import java.awt.geom.Arc2D;
public class Main extends ActiveObject{
  // Plain Variables
  /**
   * Maximum of   X coordinates of initial cities
   */
public double maxX;
  /**
   * Maximum of Y coordinates of initial cities
   */
public double maxY;
  /**
   * Range of X coordinates of the cities at the beginning<br>
   * If regions are known it represents the width of the each cell adjusted
to presentation area
   */
  public double rangeX;
  /**
```

```
   * Range of Y coordinates of the cities at the beginning<br>
   * If regions are known it represents the height of the each cell adjusted
to presentation area
   */
  public double rangeY;
  /**
   * City id is written. If you call this element from the linked list make
necessary calculations
   */
  public String maxXCity;
  /**
   * City id is written. If you call this element from the linked list make
necessary calculations
   */
  Public String maxYCity;
  /**
   * City id is written. If you call this element from the linked list make
necessary calculations
   */
  public String minXCity;
  /**
   * City id is written. If you call this element from the linked list make
necessary calculations
   */
  public String minYCity;
  /**
   * Calculated after initial cities are created. It is used to determine
region elements when region numbers are not known in advance.
   */
  public double avgDistBtwCities;
  /**
   * Do not delete but you can make it invisible. It is used to initialize
cities' locations
   */
  public double rndXCord;
  /**
   * Do not delete but you can make it invisible. It is used to initialize
cities' locations
   */
  public double rndYCord;
  /**
   * A temporary integer variable used for intermediate assigments or
calculations
   */
  public int tempInt;
  /**
   * A temporary double variable used for intermediate assigments or
calculations
   */
 public double tempDouble;
  /**
   * Sum of X coordinates of the cities at the beginning
   */
 public double sumX;
  /**
   * Sum of Y coordinates of the cities at the beginning
   */
  public double sumY;
  /**
   * Stores x coordinate of the home city
   */
  public double xCordOfHomeCity;
  /**
   * Stores Y coordinate of the home city
   */
  public double yCordOfHomeCity;
  /**
   * A temporary string variable used for intermediate assigments
```

```
   */
  public String tempString;
  public double deltaB;
  static public double simStartTime;
  static public int totNumbOfCityInTour;
  /**
   * When a new city arrives, this variable temporary stores its X
coordinate
   */
  static public double newCityXCord;
  /**
   * When a new city arrives, this variable temporary stores its Y
coordinate
   */
  static public double newCityYCord;
  /**
   * When a new city arrives, this variable temporary stores its name
   */
static public String newCityName;
public double tempX;public
double tempY;
public int dispIndex;
public int neighType;
static public double regCenterPInitialX;
static public double regCenterPInitialY;
public int allCitiesIndex;
public int cellXCord;
public int cellYCord;
static public String cityOrRegion;
static public int allRegionsIndex;
  /**
   * Used to change the text "Initial number of city" to "Number of city: ".
   */
public String infCity;
  /**
   * Minimum of   X coordinates of initial cities<br>
   * If regions are known it represents the origin of the X cord for
presentation window
   */
public double minX;
  /**
   * Minimum of   Y coordinates of initial cities<br>
   * If regions are known it represents the origin of the X cord for
presentation window
   */
public double minY;
  /**
   * Total number of regions currently
   */
public int totNumbOfReg;
  /**
   * Represents the number of current tour
   */
static public int tourCounter = 1 ;
  /**
   * Simplify to get X coordinate of the necessary city. If any element is
added between city and x coordinate in the linked list, do not forget to<br>
   * adjust value of this variable
   */
static public final int xCordIndexL = 1 ;
  /**
   * Simplify to get Y coordinate of the necessary city
   */
static public final int yCordIndexL = xCordIndexL+1 ;
  /**
   * Represents the current number of city .
   */
public int numbOfCity;
```

```
  /**
   * This variable is used to separete initial  X cord value and initial Y
cord value in the linked list.
   */
static public final int increaseFac = 2 ;
  /**
   * Used to write city id in the linked lists
   */
static public final String cityIdString = "City " ;
  /**
   * Used to write region id in the linked lists
   */
static public finalStringregionIdString = "Region " ;
  /**
   * Used to write tour id in the linked lists
   */
static public final String tourIdString = "Tour " ;
  /**
   * This is used to represent unvisited cities
   */
static public final String unVisitedString = "Unvisited" ;
  /**
   * This is used to representvisited cities
   */
static public final String visitedString = "Visited" ;
  /**
   * This is used to deleted cities
   */
static public final String deletedString = "Deleted" ;
  /**
   * Stores the home city of the dynamic TSP
   */
public int homeCity;
public boolean startToCreateEvent;
static public int neighborType = 1 ;
static public final int numbOfAttributes = 3 ;
  /**
   * =0 if dtsp is selected<br>
   * =1 other problem type
   */
static public int constant1Or0 = 0 ;
  /**
   * = true for DTSP<br>
   * = false for others
   */
static public boolean partial = false ;
static public int count = 0 ;
public String numbOfRegOptString;
public String initialSolStrategyGMAString;
static public final String costString = "Cost " ;
  // Collection Variables
  /**
   * Sequence of elements finally<br>
   * First element : region ID<br>
   * Second :  City id of region center (If city center is calculated after
cities are regioned and there is no city at this point,City -1 is written as
center city id)<br>
   * Third:  Xcord of region center<br>
   * Fourth:Y cord of region center<br>
   * Fifth:City id of following city <br>
   * Sixth: Xcord of following city <br>
   * Seventh: Y cord of following city
   */
public java.util.LinkedList < Object > RegionsAndCities = new
java.util.LinkedList<Object>();
  /**
   * Stores all cities up to know including deleted cities. <br>
```

```
   * All distance calculations are made using the lements of this linked
list<br>
   * <br>
   * First elment:City ID<br>
   * Second: City X cord<br>
   * Third: City Y Cord<br>
   * ...
   */
public java.util.LinkedList <Object > CitiesDataBase = new
java.util.LinkedList<Object>();
  /**
   * Region name<br>
   * # of city in this region<br>
   * First city of this region<br>
   * Second city of this region<br>
   * ....
   */
public java.util.LinkedList <Object > RegionsSequence = new
java.util.LinkedList<Object>();
public java.util.LinkedList <
String > InstCityNames = new java.util.LinkedList<String>();
// Events
public EventRate newEvent = new EventRate(this);
@Override
public String getNameOf( EventRate _e ) {
     if ( _e == newEvent) return "newEvent";
     return super.getNameOf( _e );}
  @Override
  public double evaluateRateOf( EventRate _e ) {
    if ( _e == newEvent) return 0.05 ;
    return super.evaluateRateOf( _e );}
@Override
public void executeActionOf( EventRate _e ) {
if ( _e == newEvent) {
int randIntNumb=0;
double randDouble=0;
String sendThis="";
boolean acceptNewCord=true;
if(startToCreateEvent==true)
{randDouble=(2*rndm.nextDouble()-1);
      if(randDouble<0){//Delete city
            do{randIntNumb=rndm.nextInt(InstCityNames.size());}
      while((generalManager.BestFeasibleTourGMA.indexOf(InstCityNames.get(r
andIntNumb))==1)||(InstCityNames.get(randIntNumb).toString()==generalManager
.BestFeasibleTourGMA.get(constant1Or0).toString()));
            newCityName=InstCityNames.get(randIntNumb);
       System.out.println("aaa"+InstCityNames.get(randIntNumb));
System.out.println("bbb"+generalManager.BestFeasibleTourGMA.get(constant1Or0
));
            sendThis="Delete city ";}
else{
do{
                  rndXCord=rndm.nextInt(((int)
widthOfEnvironment.getValue()));
                  rndYCord=rndm.nextInt(((int)
lengthOfEnvironment.getValue()));
            if(cityP.contains((int)rndXCord,(int)rndYCord)){
      if(cityP.get((Integer.parseInt(CitiesDataBase.get(CitiesDataBase.inde
xOf((int)rndXCord)-1).toString().substring(cityIdString.length()))))-
1).getFillColor()!=darkTurquoise){acceptNewCord=false;}}}
            while(acceptNewCord=false);
//Addassign x cord, y cord and name of the newcity to corresponding
variables
            allCitiesIndex++;
            newCityXCord=rndXCord;
            newCityYCord=rndYCord;
            newCityName=cityIdString+(allCitiesIndex);
//Presentation of the new city and its attributes
```

```
cityP.get(allCitiesIndex1).setX((setPresent.setXYCordOfRMAPres((int)newCityX
Cord,cityArea.getWidth(),((int)
widthOfEnvironment.getValue()),cityArea.getX())));
              cityP.get(allCitiesIndex-
1).setY((setPresent.setXYCordOfRMAPres((int)newCityYCord,cityArea.getHeight(
),((int) lengthOfEnvironment.getValue()),cityArea.getY())));
              cityP.get(allCitiesIndex-1).setLineColor(black);
              cityP.get(allCitiesIndex-1).setFillColor(black);
       dispAgentNumber.get(allCitiesIndex-1).setX(cityP.get(allCitiesIndex-
1).getX());
dispAgentNumber.get(allCitiesIndex-1).setY(cityP.get(allCitiesIndex-
1).getY());
dispAgentNumber.get(allCitiesIndex-1).setText(newCityName);
              for(int i=0; i<CitiesDataBase.size();i=i+numbOfAttributes){

       generalManager.DistanceLinkedList.add(CitiesDataBase.get(i).toString(
)+newCityName);
       generalManager.DistanceLinkedList.add(necessaryCalculations.calculate
EucDist(Double.parseDouble(CitiesDataBase.get((i+xCordIndexL)).toString()),n
ewCityXCord,Double.parseDouble(CitiesDataBase.get((i+yCordIndexL)).toString(
)),newCityYCord));//In order to simplify readability}
              print.writeToTxtFile("X cord of new city "+newCityXCord+"Y
cord of new city "+newCityYCord);
              print.writeToTxtFile("Distance between new city and others are
given below:");
       print.printLinkedListToTxt(generalManager.DistanceLinkedList.subList(
(CitiesDataBase.size()/numbOfAttributes),generalManager.DistanceLinkedList.s
ize()));
       //************************************************************
              //Add new city to the cities database
              CitiesDataBase.add(newCityName);
              InstCityNames.add(newCityName);
              CitiesDataBase.add((int)newCityXCord);
              CitiesDataBase.add((int)newCityYCord);
       //************************************************************
sendThis="New city ";}
print.writeToTxtFile("New event ("+sendThis+")"+newCityName);
print.writeToTxtFile("Received at  "+time());
startToCreateEvent=false;
generalManager.newEventReceived=true;
generalManager.send((sendThis+ newCityName),generalManager);};
return ;}
super.executeActionOf( _e );}
  // Embedded Objects
  public GeneralManagerAgent generalManager;
  public VehicleAgent vehicle;
  public String getNameOf( ActiveObject ao ) {
    if ( ao == generalManager ) return "generalManager";
    if ( ao == vehicle ) return "vehicle";
    return null;}
  public ActiveObjectArrayList<RegionManagerAgents> regionManagerAgents =
new ActiveObjectArrayList<RegionManagerAgents>();
  public ActiveObjectArrayList<LocalVehicle> localVehicle = new
ActiveObjectArrayList<LocalVehicle>();
  public String getNameOf( ActiveObjectCollection<?> aolist ) {
    if( aolist == regionManagerAgents ) return "regionManagerAgents";
    if( aolist == localVehicle ) return "localVehicle";
    return null;}
  /**
   * This method creates and adds new embedded object in the replicated
embedded object collection regionManagerAgents<br>
   * @return newly created embedded object
   */
  public RegionManagerAgents add_regionManagerAgents() {
    int index = regionManagerAgents.size();
    RegionManagerAgents object = instantiate_regionManagerAgents_xjal( index
);
    setupParameters_regionManagerAgents_xjal( object, index );
```

```
    create_regionManagerAgents_xjal( object, index );
    object.start();
    return object;}
  /**
   * This method removes the given embedded object from the replicated
embedded object collection regionManagerAgents<br>
   * The given object is destroyed, but not immediately in common case.
   * @param object the active object - element of replicated embedded object
regionManagerAgents - which should be removed
   * @return <code>true</code> if object was removed successfully,
<code>false</code> if it doesn't belong to regionManagerAgents
   */
  public boolean remove_regionManagerAgents( RegionManagerAgents object ) {
    if( ! regionManagerAgents._remove( object ) ){
      return false;}
    object.setDestroyed();
    return true;}
  /**
   * This method creates and adds new embedded object in the replicated
embedded object collection localVehicle<br>
   * @return newly created embedded object
   */
  public LocalVehicle add_localVehicle() {
    int index = localVehicle.size();
    LocalVehicle object = instantiate_localVehicle_xjal( index );
    setupParameters_localVehicle_xjal( object, index );
    create_localVehicle_xjal( object, index );
    object.start();
    return object;}
  /**
   * This method removes the given embedded object from the replicated
embedded object collection localVehicle<br>
   * The given object is destroyed, but not immediately in common case.
   * @param object the active object - element of replicated embedded object
localVehicle - which should be removed
   * @return <code>true</code> if object was removed successfully,
<code>false</code> if it doesn't belong to localVehicle
   */
  public boolean remove_localVehicle( LocalVehicle object ) {
    if( ! localVehicle._remove( object ) ){
      return false;}
    object.setDestroyed();
    return true;}
  /**
   * Creates an embedded object instance<br>
   * <i>This method should not be called by user</i>
   */
  private GeneralManagerAgent instantiate_generalManager_xjal() {
    GeneralManagerAgent object = new GeneralManagerAgent( getEngine(), this,
null );
    return object;}
  /**
   * Setups parameters of an embedded object instance<br>
   * This method should not be called by user
   */
  private void setupParameters_generalManager_xjal(GeneralManagerAgent
object ) {}
  /**
   * Setups an embedded object instance<br>
   * This method should not be called by user
   */
private void create_generalManager_xjal(GeneralManagerAgent object){
object.setEnvironment( negotiationEnvCont);object.create();}
  /**
   * Creates an embedded object instance and adds it to the end of
replicated embedded object list<br>
   * <i>This method should not be called by user</i>
   */
```

```
  private RegionManagerAgents instantiate_regionManagerAgents_xjal( final
int index ) {
    RegionManagerAgents object = new RegionManagerAgents( getEngine(), this,
regionManagerAgents );
     regionManagerAgents._add(object);
    return object;}
  /**
   * Setups parameters of an embedded object instance<br>
   * This method should not be called by user
   */
  private void setupParameters_regionManagerAgents_xjal(RegionManagerAgents
object, final int index ) {
  }
  /**
   * Setups an embedded object instance<br>
   * This method should not be called by user
   */
  private void create_regionManagerAgents_xjal(RegionManagerAgents object,
final int index ) {
    object.setEnvironment(
negotiationEnvCont );
    object.create();
    // Port connections
  }
  /**
   * Creates an embedded object instance<br>
   * <i>This method should not be called by user</i>
   */
  private VehicleAgent instantiate_vehicle_xjal() {
    VehicleAgent object = new VehicleAgent( getEngine(),this,null)
    return object;}
  /**
   * Setups parameters of an embedded object instance<br>
   * This method should not be called by user
   */
  private void setupParameters_vehicle_xjal(VehicleAgent object ) {}
  /**
   * Setups an embedded object instance<br>
   * This method should not be called by user
   */
  private void create_vehicle_xjal(VehicleAgentobject){object.create(); }
  /**
   * Creates an embedded object instance and adds it to the end of
replicated embedded object list<br>
   * <i>This method should not be called by user</i>
   */
  private LocalVehicle instantiate_localVehicle_xjal( final int index) {
    LocalVehicle object = new LocalVehicle( getEngine(), this, localVehicle
);
    localVehicle._add(object);
    return object; }
  /**
   * Setups parameters of an embedded object instance<br>
   * This method should not be called by user
   */
  private void setupParameters_localVehicle_xjal(LocalVehicle object, final
int index ) {}
  /**
   * Setups an embedded object instance<br>
   * This method should not be called by user
   */
  private void create_localVehicle_xjal(LocalVehicle object, final int index
) {
    object.create();
    // Port connections
}void  generate() {
    for ( int i = 1 ; i <= numbOfCity ; i++  ) { // generateCities
{ // initialLocOfCities
```

```
// InitializationOfEnvironment: Generate location for predetermined
numbOfCity
rndXCord= rndm.nextInt((int) widthOfEnvironment.getValue());
rndYCord= rndm.nextInt((int) lengthOfEnvironment.getValue()) ;
//rndXCord=necessaryCalculations.roundTwoDecimals((rndm.nextDouble()*((int)
widthOfEnvironment.getValue())));
//rndYCord=necessaryCalculations.roundTwoDecimals((rndm.nextDouble()*((int)
lengthOfEnvironment.getValue())));
RegionsAndCities.add((cityIdString+i));
RegionsAndCities.add((int)rndXCord);
RegionsAndCities.add((int)rndYCord);
CitiesDataBase.add(cityIdString+i);
InstCityNames.add(cityIdString+i);
CitiesDataBase.add((int)rndXCord);
CitiesDataBase.add((int)rndYCord);
cityP.get(I1).setX((setPresent.setXYCordOfRMAPres((int)rndXCord,cityArea.
getWidth(),((int) widthOfEnvironment.getValue()),cityArea.getX())));
cityP.get(i-1).
setY((setPresent.setXYCordOfRMAPres((int)rndYCord,cityArea.getHeight(),((int
) lengthOfEnvironment.getValue()),cityArea.getY())));
cityP.get(i-1).setLineColor(red);
      } // initialLocOfCities
     if (problemType.getValue().toString().contains("Regions are
known")==true ) { // DecRegionsAreKnown
        { // RegionsAreKnown
          // Regions are known
cellXCord=(int)ceil((Double.parseDouble(RegionsAndCities.get(RegionsAndCitie
s.size()-2).toString()))/rangeX);
cellYCord=(int)ceil((Double.parseDouble(RegionsAndCities.getLast().toString(
))/rangeY));
tempInt=((cellXCord-1)*((int)numbOfRow.getValue()))+cellYCord;
tempString=regionIdString+tempInt;
if(RegionsAndCities.indexOf(tempString)==-1){
      totNumbOfReg++;
      RegionsAndCities.add((RegionsAndCities.size()-
numbOfAttributes),(tempString));
      RegionsSequence.add(tempString);
      RegionsSequence.add(cityIdString+i);}
else{
      tempInt=necessaryCalculations.findStOccurenceOfSubStringFirstToLast(R
egionsSequence,(RegionsSequence.indexOf(tempString)+1),RegionsSequence.size(
),1,1,regionIdString);
      tempInt=(tempInt==-1?RegionsSequence.size():tempInt);
      RegionsSequence.add(tempInt,cityIdString+i);
      tempInt=necessaryCalculations.findIndexOfSubStringFirstToLast(Regions
AndCities,(RegionsAndCities.indexOf(tempString)+1),RegionsAndCities.size(),n
umbOfAttributes,regionIdString);
      if(tempInt!=-1){
      RegionsAndCities.addAll(tempInt,RegionsAndCities.subList((RegionsAndC
ities.size()-numbOfAttributes),RegionsAndCities.size()));
            RegionsAndCities.subList((RegionsAndCities.size()-
numbOfAttributes),RegionsAndCities.size()).clear();}}
        } // RegionsAreKnown
      } // DecRegionsAreKnown
    } // generateCities
    { // code1
InstCityNames.removeFirst();
    } // code1
    if (
problemType.getValue().toString().contains("Regions are known")==true ) { //
DecRegionsAreKnown2
      { // citiesWereAssignedToPredeterminedReg
        // Print Regions and Cities
allCitiesIndex=numbOfCity;
if(homeCityOpt.getValue()==1){
 print.writeToTxtFile("Home region is region"
+(Integer.parseInt(homeCityVal.getText()))+"and this region with its
elements will be written at the begining of the region sequence");
```

```
 int
frstIn=necessaryCalculations.findStOccurenceOfSubStringFirstToLast(RegionsSe
quence,0,RegionsSequence.size(),1,(Integer.parseInt(homeCityVal.getText())),
regionIdString);
 int
lst=(necessaryCalculations.findStOccurenceOfSubStringFirstToLast(RegionsSequ
ence,0,RegionsSequence.size(),1,(Integer.parseInt(homeCityVal.getText())+1),
regionIdString)==-1
?RegionsSequence.size():(necessaryCalculations.findStOccurenceOfSubStringFir
stToLast(RegionsSequence,0,RegionsSequence.size(),1,(Integer.parseInt(homeCi
tyVal.getText())+1),regionIdString)));
  RegionsSequence.addAll(0,RegionsSequence.subList(frstIn,lst));
  RegionsSequence.subList(frstIn+(lst-frstIn),lst+(lst-frstIn)).clear();}
print.writeToTxtFile("Regions and Cities linked list for GTSP (Regions are
known)");
print.writeToTxtFile("Region centers are calculated by using center of
gravity.If there is not any city at the region center, City -1 is written");
print.printLinkedListToTxt(RegionsAndCities);
necessaryCalculations.findCenterByCG(RegionsAndCities,numbOfAttributes);
print.printLinkedListToTxt(RegionsAndCities);
print.writeToTxtFile("Regions Sequences");
print.printLinkedListToTxt(RegionsSequence);
      } // citiesWereAssignedToPredeterminedReg
    } else { // DecRegionsAreKnown2
      { // ifNecChangeHomeCity
        // Make Initial Calculations
allCitiesIndex=numbOfCity;
switch (homeCityOpt.getValue())
{case  0: xCordOfHomeCity= (Double.parseDouble
((RegionsAndCities.get(xCordIndexL)).toString()) );
      yCordOfHomeCity=(Double.parseDouble((RegionsAndCities.get(yCordIndexL
)).toString()) );
      break;
case 1:tempInt=(homeCity-1)*(numbOfAttributes);
      RegionsAndCities.addAll(0,RegionsAndCities.subList(tempInt,tempInt+(n
umbOfAttributes)));
      RegionsAndCities.subList(tempInt+numbOfAttributes,tempInt+2*numbOfAtt
ributes).clear();
      xCordOfHomeCity=(Double.parseDouble((RegionsAndCities.get(xCordIndexL
)).toString()) );
      yCordOfHomeCity=(Double.parseDouble((RegionsAndCities.get(yCordIndexL
)).toString()) );
      break;
      default:
          xCordOfHomeCity=0;
             yCordOfHomeCity=0;
      break; }
      } // ifNecChangeHomeCity
     if (problemType.getValue().contains("Dynamic TSP") ) { // ProblemType
        { // DynamicTSPbranch
          // Dynamic TSP
print.writeToTxtFile("Cities linked list");
print.printLinkedListToTxt(RegionsAndCities);
        } // DynamicTSPbranch
      } else { // ProblemType
        { // makeNecCalculations
          // Calculations for GTSP
int frstCity=1;
//totNumbOfCityInTour=numbOfCity;
 // Max and max of X and Y cord**********0 x cord, 1 y cord.
RegionsAndCities=necessaryCalculations.findMaxElOfLinkedList(RegionsAndCitie
s,frstCity,numbOfAttributes);
maxXCity=RegionsAndCities.get((Integer.parseInt(RegionsAndCities.getLast().t
oString())-frstCity)).toString();
RegionsAndCities.removeLast();
maxX=(Double.parseDouble((RegionsAndCities.getLast()).toString()) );
RegionsAndCities.removeLast();
```

```
RegionsAndCities=necessaryCalculations.findMinElOfLinkedList(RegionsAndCitie
s,frstCity,numbOfAttributes);
minXCity=RegionsAndCities.get((Integer.parseInt(RegionsAndCities.getLast().t
oString())-(frstCity))).toString();
RegionsAndCities.removeLast();
minX=(Double.parseDouble((RegionsAndCities.getLast()).toString()) );
RegionsAndCities.removeLast();
frstCity=1;
RegionsAndCities=necessaryCalculations.findMaxElOfLinkedList(RegionsAndCitie
s,frstCity+1,numbOfAttributes);
maxYCity=RegionsAndCities.get((Integer.parseInt(RegionsAndCities.getLast().t
oString())-(frstCity+1))).toString();
RegionsAndCities.removeLast();
maxY=(Double.parseDouble((RegionsAndCities.getLast()).toString())) ;
RegionsAndCities.removeLast();
RegionsAndCities=necessaryCalculations.findMinElOfLinkedList(RegionsAndCitie
s,frstCity+1,numbOfAttributes);
minYCity=RegionsAndCities.get((Integer.parseInt(RegionsAndCities.getLast().t
oString())-(frstCity+1))).toString();
RegionsAndCities.removeLast();
minY=Double.parseDouble((RegionsAndCities.getLast()).toString()) ;
RegionsAndCities.removeLast();
//Range of x coordinate and y coordinate
sumX=necessaryCalculations.sumOfLinkedList(RegionsAndCities,1,numbOfAttribut
es);
sumY=necessaryCalculations.sumOfLinkedList(RegionsAndCities,2,numbOfAttribut
es);
//Range of x coordinate and y coordinate
rangeX=abs(maxX -minX);
rangeY=abs(maxY -minY);
        } // makeNecCalculations
        if (numbOfRegOpt.getValue()==1 ) { // GTSPType
          { // GTSPNumberOfRegionsAreKnown
            // GTSP Number of Region is Known
          } // GTSPNumberOfRegionsAreKnown
        } else { // GTSPType
          { // code5
          } // code5
          if (avgDistOpt.getValue()=="User defined" ) { // avgDistUserDefDec
            { // avgDistUser
              // User defined average distance
avgDistBtwCities=avgDistInput.getValue();
print.writeToTxtFile("Average distance: "+avgDistBtwCities);
            } // avgDistUser
          } else { // avgDistUserDefDec
            { // code4
            } // code4
            if (avgDistOpt.getValue()=="Circle diameter" ) { // avgDistDec
              { // circleDia
                // Circle diameter distance
necessaryCalculations.updateConstructor(rangeX,rangeY,numbOfCity);
avgDistBtwCities=necessaryCalculations.findAvgDist(1);
              } // circleDia
            } else { // avgDistDec
              { // code
                // Euclidean distance
necessaryCalculations.updateConstructor(rangeX,rangeY,numbOfCity);
avgDistBtwCities=necessaryCalculations.findAvgDist();
              } // code
            } // avgDistDec
            { // code2
print.writeToTxtFile("Average distance: "+avgDistBtwCities);
            } // code2
            if (regDistOpt.getValue()=="Modified equal distance (Next
minimum)"||regDistOpt.getValue()=="Modified equal distance (Multiply
constant)" ) { // modOrEqDec
              { // code3
              } // code3
```

```
              if (
regDistOpt.getValue()=="Modified equal distance (Next minimum)" ) { //
nextMinOrMultConstDec
                { // nextMinDist
                  // Modified equal distance (Next minimum)
FindRegionsOfAgents findRegions = new
FindRegionsOfAgents(numbOfCity,avgDistBtwCities,RegionsAndCities);
if(changeReportFileLocation.isSelected()==true){
      findRegions.print.updatePath(fileLocation.getText());}
//CalculateRegionCenter calculateReg= new
CalculateRegionCenter(RegionsAndCities);
RegionsAndCities=findRegions.findInitialRegModifiedEqNextMin();
//Convert number of region to integer
totNumbOfReg=(Integer.parseInt((RegionsAndCities.getLast()).toString()) );
RegionsAndCities.removeLast();
                } // nextMinDist
              } else { // nextMinOrMultConstDec
                { // multConst
                  // Modified equal distance (Multiply constant)
FindRegionsOfAgents findRegions = new
FindRegionsOfAgents(numbOfCity,avgDistBtwCities,RegionsAndCities);
if(changeReportFileLocation.isSelected()==true){
      findRegions.print.updatePath(fileLocation.getText());}
//CalculateRegionCenter calculateReg= new
CalculateRegionCenter(RegionsAndCities);
RegionsAndCities=findRegions.findInitialRegModifiedEqMultConst(numbOfReg.get
Value());
//Convert number of region to integer
totNumbOfReg=(Integer.parseInt((RegionsAndCities.getLast()).toString()) );
RegionsAndCities.removeLast();
                } // multConst
              } // nextMinOrMultConstDec
            } else { // modOrEqDec
              { // eqDist
                // Equal distance
FindRegionsOfAgents findRegions = new
FindRegionsOfAgents(numbOfCity,avgDistBtwCities,RegionsAndCities);
if(changeReportFileLocation.isSelected()==true){
      findRegions.print.updatePath(fileLocation.getText());}
RegionsAndCities=findRegions.findInitialRegEqDist();
//Convert number of region to integer
totNumbOfReg=(Integer.parseInt((RegionsAndCities.getLast()).toString()) );
RegionsAndCities.removeLast();
              } // eqDist
            } // modOrEqDec
          } // avgDistUserDefDec
        } // GTSPType
        if (
regCenterOpt.getValue()=="Center of gravity" ) { // regCenterDec
          { // regCenterCOfGrav
            // Region Center (Center of Gravity)
necessaryCalculations.findCenterByCG(RegionsAndCities,numbOfAttributes);
print.writeToTxtFile("Regions and cities linked list.If there is not ant
city at the region center, City -1 is written");
print.printLinkedListToTxt(RegionsAndCities);
          } // regCenterCOfGrav
        } // regCenterDec
      } // ProblemType
    } // DecRegionsAreKnown2
    return; // returnStatement}
  // View areas
  public ViewArea createInitialCities = new ViewArea( this, "Create Initial
Cities", 1080, 10, ViewArea.TOP_LEFT, ViewArea.NONE, 1.0, 1480, 100 );
  public ViewArea inputFromUser = new ViewArea( this, "Input from user", 20,
30, ViewArea.TOP_LEFT, ViewArea.SHRINK_TO_FIT, 1.0, 1050, 350 );
  public ViewArea cityArea = new ViewArea( this, "Cities", -3200, 2,
ViewArea.TOP_LEFT, ViewArea.SHRINK_TO_FIT, 1.0, 1320, 1000 );
```

```
  public ViewArea agentsObjectsView = new ViewArea( this, "Agents objects
view", 0, -400, ViewArea.TOP_LEFT, ViewArea.NONE,1.0,100,100);
  static final Color _regCenterP_FillColor = purple;
  static final int _determineRegions = 1;
  static final int _initNumbOfCity = 2;
  static final int _lengthOfEnvironment = 3;
  static final int _widthOfEnvironment = 4;
  static final int _numbOfReg = 5;
  static final int _convertCord = 6;
  static final int _createAgentsButton = 7;
  static final int _numbOfRegOpt = 8;
  static final int _avgDistOpt = 9;
  static final int _regDistOpt = 10;
  static final int _regionDetAlg = 11;
  static final int _homeCityOpt = 12;
  static final int _homeCityVal = 13;
  static final int _setInitialParameters = 14;
  static final int _searchAlgorithms = 15;
  static final int _searchAlgP = 16;
  static final int _maxNumbOfIter = 17;
  static final int _maxNumbOfNonImp = 18;
  static final int _numbOfBestCan = 19;
  static final int _neighOpt = 20;
  static final int _regCenterOpt = 21;
  static final int _avgDistInput = 22;
  static final int _numbOfNeg = 23;
  static final int _initialSolStrategyGMA = 24;
  static final int _problemType = 25;
  static final int _numbOfRow = 26;
  static final int _numbOfCol = 27;
  static final int _optStrategies = 28;
  static final int _changeReportFileLocation = 29;
  static final int _fileLocation = 30;
  static final int _avoidSameMembers = 31;
  static final int _maxNumbOfIterSGS = 32;
  static final int _maxNumbOfNonImpSGS = 33;
  static final int _dispLOfEnv = 34;
  static final int _dispWOfEnv = 35;
  static final int _dispNumbOfCityR = 36;
  static final int _dispNumbOfReg = 37;
  static final int regionManagerAgents_Presentation = 38;
  static final int vehicle_presentation = 39;
  static final int _cityP = 40;
  static final int _xCordCity = 41;
  static final int _xCordLabel = 42;
  static final int _yCordLabel = 43;
  static final int _yCordCity = 44;
  static final int _dispSearchAlgP = 45;
  static final int _dispMaxNOfIter = 46;
  static final int _dispMaxNOfNonImp = 47;
  static final int _dispMaxNOfBestCanSol = 48;
  static final int _dispMultConst = 49;
  static final int _dispAvgDist = 50;
  static final int _dispNumbOfNeg = 51;
  static final int _dispNumbOfRow = 52;
  static final int _dispNumbOfCol = 53;
  static final int _dispAgentNumber = 54;
  static final int _dispRegName = 55;
  static final int _dispMaxNumbOfIterSGS = 56;
  static final int _dispMaxNumbOfNonImpSGS = 57;
  static final int localVehicle_presentation = 58;
  static final int _regCenterP = 59;
  /**
   * Top-level presentation group id
   */
  static final int _presentation = 0;
  /**
   * Top-level icon group id
```

```
   */
  static final int _icon = -1;
  @Override
  public boolean onShapeClick( int _shape, int index, double clickx, double
clicky ){
    switch( _shape ){
      case _cityP:
        if (true) {for(int i=1; i<=numbOfCity;i++){
       xCordLabel.get(i).setX(cityP.get(i).getX()+12);
       xCordLabel.get(i).setY(cityP.get(i).getY()-20);
       yCordLabel.get(i).setX(cityP.get(i).getX()+12);
       yCordLabel.get(i).setY(cityP.get(i).getY()-3);
       xCordLabel.get(i).setText("Modified X");
       yCordLabel.get(i).setText("Modified Y");
       xCordCity.get(i).setX(cityP.get(i).getX()+12);
       xCordCity.get(i).setY(cityP.get(i).getY()-12);
       yCordCity.get(i).setX(cityP.get(i).getX()+12);
       yCordCity.get(i).setY(cityP.get(i).getY()+5);
       xCordCity.get(i).setText(cityP.get(i).getX());
       yCordCity.get(i).setText(cityP.get(i).getY());} }
        break;
      default: return super.onShapeClick(_shape,index,clickx, clicky);
       }return false;}
  @Override
  public void executeShapeControlAction( int _shape, int index ) {
    switch( _shape ) {
      case _determineRegions: {
cityOrRegion=regionIdString;
if(changeReportFileLocation.isSelected()==true){
       print.updatePath(fileLocation.getText());
       generalManager.print.updatePath(fileLocation.getText());
       generalManager.searchAlg.printTxt.updatePath(fileLocation.getText());
       generalManager.print.updatePath(fileLocation.getText());
       vehicle.print.updatePath(fileLocation.getText());}
determineRegions.setEnabled(false);
infCity = "Number of city: ";
if(problemType.getValue().toString().contains("Regions are known")==true){
       rangeX=cityArea.getWidth()/((int)numbOfCol.getValue());
       rangeY=cityArea.getHeight()/((int)numbOfRow.getValue());}
if(problemType.getValue()=="Dynamic GTSP (Regions are not
known"||problemType.getValue()=="Dynamic GTSP with TSP (Regions are not
known)"){
              regCenterPInitialX=regCenterP.get(0).getX();
       regCenterPInitialY=regCenterP.get(0).getY();}
if(problemType.getValue()=="Dynamic TSP"){
       partial=true;
       totNumbOfReg=numbOfCity;
       cityOrRegion=cityIdString;}
//About region selection algorithm
print.writeToTxtFileUsedElements(regDistOpt.isVisible(),("Region distance
option: "+regDistOpt.getValue()));
print.writeToTxtFileUsedElements(dispMultConst.isVisible(),dispMultConst.get
Text());
print.writeToTxtFileUsedElements(avgDistOpt.isVisible(),("Average distance
option: "+avgDistOpt.getValue()));
print.writeToTxtFileUsedElements(dispAvgDist.isVisible(),dispAvgDist.getText
());
print.writeToTxtFileUsedElements(regCenterOpt.isVisible(),("Region center:
"+regCenterOpt.getValue()));
//About algorithm
print.writeToTxtFileUsedElements(searchAlgorithms.isVisible(),("Search
algorithm: "+searchAlgorithms.getValue()));
print.writeToTxtFileUsedElements(dispSearchAlgP.isVisible(),(dispSearchAlgP.
getText()+searchAlgP.getText()));
print.writeToTxtFileUsedElements(dispMaxNumbOfIterSGS.isVisible(),dispMaxNum
bOfIterSGS.getText());
print.writeToTxtFileUsedElements(dispMaxNumbOfNonImpSGS.isVisible(),dispMaxN
umbOfNonImpSGS.getText());
```

```
print.writeToTxtFileUsedElements(dispMaxNOfIter.isVisible(),dispMaxNOfIter.g
etText());
print.writeToTxtFileUsedElements(dispMaxNOfNonImp.isVisible(),dispMaxNOfNonI
mp.getText());
print.writeToTxtFileUsedElements(dispNumbOfNeg.isVisible(),dispNumbOfNeg.get
Text());
print.writeToTxtFileUsedElements(avoidSameMembers.isSelected(),"Same members
in the population is not allowed");
print.writeToTxtFileUsedElements(neighOpt.isVisible(),("Neighborhood option:
"+neighOpt.getValue()));
print.writeToTxtFileUsedElements(dispMaxNOfBestCanSol.isVisible(),dispMaxNOf
BestCanSol.getText());
 if(searchAlgP.isVisible()==false){
 searchAlgP.setText("1");}
generate();;}
        break;
      case _convertCord: {
double doubleVal=0;
String convert;
String befComma;
String afterComma;
int j=0;
for(int i=1; i<=numbOfCity;i++){
       if(xCordLabel.get(i).getText()=="Modified X"){
       xCordLabel.get(i).setText("Original X");
       yCordLabel.get(i).setText("Original Y");
       convert=xCordCity.get(i).getText();
       j=convert.indexOf(",");
       System.out.println("j " +j);
       befComma=convert.substring(0,j);
       afterComma=convert.substring(j+1);
       convert=befComma+afterComma;
       System.out.println("Burda ne yazıyor " +befComma + "afer "
+afterComma);
       doubleVal=Double.parseDouble(convert);
       doubleVal=setPresent.setXYCordOfRMAOrg(doubleVal,cityArea.getWidth(),
((int) widthOfEnvironment.getValue()),cityArea.getX());
       convert=Double.toString(doubleVal);
       xCordCity.get(i).setText(convert);
       convert=yCordCity.get(i).getText();
       doubleVal=Double.parseDouble(convert);
       doubleVal=setPresent.setXYCordOfRMAOrg(doubleVal,cityArea.getHeight()
,((int) lengthOfEnvironment.getValue()),cityArea.getY());
       convert=Double.toString(doubleVal);
       yCordCity.get(i).setText(convert);}};}
        break;
      case _createAgentsButton: {
//Agent part
generalManager.send("Create initial region agents",generalManager);//Manager
agent is created;}
        break;
      default:
        super.executeShapeControlAction( _shape, index );
        break;}}
  @Override
  public void executeShapeControlAction( int _shape, int index, boolean
value ) {
    switch( _shape ) {
      case _setInitialParameters: {
determineRegions.setEnabled(true); ;}
        break;
      case _changeReportFileLocation: {
if (changeReportFileLocation.isSelected()==true){
       fileLocation.setVisible(true)}
else{fileLocation.setVisible(false);} ;}
        break;
      default:
        super.executeShapeControlAction( _shape, index, value );
```

```
        break;}}
  @Override
  public void executeShapeControlAction( int _shape, int index, int value )
{
    switch( _shape ) {
      case _numbOfRegOpt: {
if(numbOfRegOpt.getValue()==0){
       totNumbOfReg=0;
       numbOfRegOptString="Number of region is known";
       numbOfReg.setVisible(false);
       dispNumbOfReg.setVisible(false);
       avgDistOpt.setVisible(true);
       regDistOpt.setVisible(true);
       regionDetAlg.setVisible(false);}
if(numbOfRegOpt.getValue()==1){
       numbOfRegOptString="Number of region is not known";
       numbOfReg.setVisible(true);
       dispNumbOfReg.setText((int)numbOfReg.getValue());
       dispNumbOfReg.setVisible(true);
       avgDistOpt.setVisible(false);
       regDistOpt.setVisible(false);
       regionDetAlg.setVisible(true);};}
      break;
      case _homeCityOpt: {
if(homeCityOpt.getValue()==0){
       homeCityVal.setVisible(false);
       homeCityVal.setText("1");
       homeCity=1;}
else if(homeCityOpt.getValue()==1){
       homeCityVal.setVisible(true);}
else{homeCityVal.setVisible(false);
       homeCity=0;} ;}
      break;
      case _initialSolStrategyGMA: {
if(initialSolStrategyGMA.getValue()==0){
       initialSolStrategyGMAString="Best tour or each agent";
       if(problemType.getValue().contains("Dynamic TSP")){
       searchAlgorithms.setVisible(false);
       dispNumbOfNeg.setVisible(false);
       maxNumbOfNonImp.setVisible(false);
       maxNumbOfIter.setVisible(false);
       dispMaxNOfNonImp.setVisible(false);
       dispMaxNOfIter.setVisible(false);
       dispMaxNOfBestCanSol.setVisible(false);
       numbOfBestCan.setVisible(false);
       neighOpt.setVisible(false);
       dispSearchAlgP.setVisible(false);
       searchAlgP.setVisible(false);
numbOfNeg.setVisible(false);
avoidSameMembers.setVisible(false);
searchAlgP.setText("1");}
       else{
              searchAlgorithms.setVisible(true);
              avoidSameMembers.setVisible(true);
       dispNumbOfNeg.setVisible(true);
       maxNumbOfNonImp.setVisible(true);
       maxNumbOfIter.setVisible(true);
       dispMaxNOfNonImp.setVisible(true);
       dispMaxNOfIter.setVisible(true);
       dispMaxNOfBestCanSol.setVisible(true);
       numbOfBestCan.setVisible(true);
       neighOpt.setVisible(true);
       dispSearchAlgP.setVisible(true);
       searchAlgP.setVisible(true);
numbOfNeg.setVisible(true);
       }}
if (initialSolStrategyGMA.getValue()==1){
       initialSolStrategyGMAString="GMA determines";
```

```
              searchAlgorithms.setVisible(true);
       dispNumbOfNeg.setVisible(true);
       maxNumbOfNonImp.setVisible(true);
       maxNumbOfIter.setVisible(true);
       dispMaxNOfNonImp.setVisible(true);
       dispMaxNOfIter.setVisible(true);
       dispMaxNOfBestCanSol.setVisible(true);
       numbOfBestCan.setVisible(true);
       neighOpt.setVisible(true);
       dispSearchAlgP.setVisible(true);
       searchAlgP.setVisible(true);
numbOfNeg.setVisible(true);
avoidSameMembers.setVisible(true);} ;}
        break;
      default:
        super.executeShapeControlAction( _shape, index, value );
        break;}}
  @Override
  public void executeShapeControlAction( int _shape, int index, double value
) {
    switch( _shape ) {
      case _initNumbOfCity:
        numbOfCity = (int) value;
        break;
      case _numbOfReg:
        totNumbOfReg = (int) value;
        break;
      default:
        super.executeShapeControlAction( _shape, index, value );
        break;}}
  @Override
  public void executeShapeControlAction( int _shape, int index, String value
) {
    switch( _shape ) {
      case _avgDistOpt: {
regCenterOpt.setVisible(true);
if (avgDistOpt.getValue()=="Euclidean distance"){
       avgDistInput.setVisible(false);
       dispAvgDist.setVisible(false);}
if (avgDistOpt.getValue()=="Circle diameter"){
       avgDistInput.setVisible(false);
       dispAvgDist.setVisible(false)}
if(avgDistOpt.getValue()=="User defined"){
avgDistInput.setVisible(true);
dispAvgDist.setVisible(true);} ;}
        break;
      case _regDistOpt: {
if(regDistOpt.getValue()=="Equal distance"){
       avgDistOpt.setVisible(true);
       regionDetAlg.setVisible(false);
       numbOfReg.setVisible(false);
       dispMultConst.setVisible(false);}
if(regDistOpt.getValue()=="Modified equal distance (Next minimum)"){
       avgDistOpt.setVisible(true);
       regionDetAlg.setVisible(false);
       numbOfReg.setVisible(false);
       dispMultConst.setVisible(false);}
if(regDistOpt.getValue()=="Modified equal distance (Multiply constant)"){
       avgDistOpt.setVisible(true);
       regionDetAlg.setVisible(false);
       dispMultConst.setVisible(true);
       numbOfReg.setVisible(true);};}
        break;
      case _homeCityVal: {
 homeCity=Integer.parseInt(homeCityVal.getText()); ;}
        break;
      case _searchAlgorithms: {
dispMaxNOfIter.setVisible(true);
```

```
maxNumbOfIter.setVisible(true);
maxNumbOfNonImp.setVisible(true);
dispMaxNOfNonImp.setVisible(true);
numbOfBestCan.setVisible(true);
dispMaxNOfBestCanSol.setVisible(true);
neighOpt.setVisible(true);
if (searchAlgorithms.getValue()=="GDA"){
       dispSearchAlgP.setText("Write initial Delta B value (-1 if formula is
used)");
       dispSearchAlgP.setVisible(true);
       searchAlgP.setVisible(true);
       dispNumbOfNeg.setVisible(true);
       avoidSameMembers.setVisible(true);
       numbOfNeg.setVisible(true);} ;}
        break;
      case _searchAlgP: {
if (searchAlgorithms.getValue()=="GDA"){
       if (dispSearchAlgP.getText()=="Write initial Delta B value (-1 if
formula is used)"){
       deltaB=Double.parseDouble(searchAlgP.getText());}} ;}
        break;
      case _neighOpt: {
if(neighOpt.getValue()=="Change two cities"){
       neighborType=1;}
else if (neighOpt.getValue()=="CrossOver(Point)"){
       neighborType=2;} ;}
      break;
      case _problemType: {
if(problemType.getValue()=="Dynamic TSP"){
       numbOfRegOpt.setVisible(false);
       regDistOpt.setVisible(false);
       avgDistOpt.setVisible(false);
       regCenterOpt.setVisible(false);
       dispMultConst.setVisible(false);
       dispNumbOfReg.setVisible(false);
       numbOfReg.setVisible(false);
       avgDistInput.setVisible(false);
       regionDetAlg.setVisible(false);
       optStrategies.setVisible(true);
       numbOfRow.setVisible(false);
       dispNumbOfRow.setVisible(false);
       numbOfCol.setVisible(false);
       dispNumbOfCol.setVisible(false);
       constant1Or0=0;}
else if(problemType.getValue().toString().contains("Regions are
known")==true){
       numbOfRegOpt.setVisible(false);
       regDistOpt.setVisible(false);
       avgDistOpt.setVisible(false);
       regCenterOpt.setVisible(false);
       dispMultConst.setVisible(false);
       dispNumbOfReg.setVisible(false);
       numbOfReg.setVisible(false);
       avgDistInput.setVisible(false);
       regionDetAlg.setVisible(false);
       optStrategies.setVisible(true);
       numbOfRow.setVisible(true);
       dispNumbOfRow.setVisible(true);
       numbOfCol.setVisible(true);
       dispNumbOfCol.setVisible(true);
       constant1Or0=1;}
else if(problemType.getValue().toString().contains("are not known")){
       numbOfRegOpt.setVisible(true);
       regDistOpt.setVisible(true);
       avgDistOpt.setVisible(true);
       regCenterOpt.setVisible(true);
       dispMultConst.setVisible(true);
       dispNumbOfReg.setVisible(true);
```

```
        numbOfReg.setVisible(false);
        avgDistInput.setVisible(true);
        regionDetAlg.setVisible(false);
        optStrategies.setVisible(true);
        numbOfRow.setVisible(false);
        dispNumbOfRow.setVisible(false);
        numbOfCol.setVisible(false);
        dispNumbOfCol.setVisible(false);
        constant1Or0=1;}
else{constant1Or0=1;}
if(problemType.getValue().contains("GTSP")){
        searchAlgorithms.setVisible(true);
        dispMaxNumbOfIterSGS.setVisible(true);
        maxNumbOfIterSGS.setVisible(true);
        maxNumbOfNonImpSGS.setVisible(true);
        dispMaxNumbOfNonImpSGS.setVisible(true);}
else{dispMaxNumbOfIterSGS.setVisible(false);
        maxNumbOfIterSGS.setVisible(false);
        maxNumbOfNonImpSGS.setVisible(false);
        dispMaxNumbOfNonImpSGS.setVisible(false);} ;}
       break;
       default:
       super.executeShapeControlAction( _shape, index, value );
         break;}}
  @Override
  public double getShapeControlMinimum( int _shape, int index ) {
    switch( _shape ) {
      case _initNumbOfCity: return 0 ;
      case _lengthOfEnvironment: return 0 ;
      case _widthOfEnvironment: return 0 ;
      case _numbOfReg: return 0 ;
      case _maxNumbOfIter: return 0 ;
      case _maxNumbOfNonImp: return 0 ;
      case _numbOfBestCan: return 0 ;
      case _avgDistInput: return 0 ;
      case _numbOfNeg: return 0 ;
      case _numbOfRow: return 0 ;
      case _numbOfCol: return 0 ;
      case _maxNumbOfIterSGS: return 0 ;
      case _maxNumbOfNonImpSGS: return 0 ;
      default: return super.getShapeControlMinimum( _shape, index)}}
  @Override
  public double getShapeControlMaximum( int _shape, int index ) {
    switch( _shape ) {
      case _initNumbOfCity: return 100 ;
      case _lengthOfEnvironment: return 1000 ;
      case _widthOfEnvironment: return 1000 ;
      case _numbOfReg: return 100 ;
      case _maxNumbOfIter: return 1000 ;
      case _maxNumbOfNonImp: return 1000 ;
      case _numbOfBestCan: return 50 ;
      case _avgDistInput: return 10000 ;
      case _numbOfNeg: return 100 ;
      case _numbOfRow: return 100 ;
      case _numbOfCol: return 100 ;
      case _maxNumbOfIterSGS: return 10000 ;
      case _maxNumbOfNonImpSGS: return 10000 ;
      default: return super.getShapeControlMaximum(_shape,index );}}
  @Override
  public boolean getShapeControlDefaultValueBoolean( int _shape, int index )
{
    switch(_shape) {
      case _setInitialParameters: return false ;
      case _changeReportFileLocation: return false ;
      default: return super.getShapeControlDefaultValueBoolean( _shape,
index );}}
  @Override
  public double getShapeControlDefaultValueDouble( int _shape, int index ) {
```

```
    switch(_shape) {
      case _initNumbOfCity: return numbOfCity ;
      case _lengthOfEnvironment: return 100 ;
      case _widthOfEnvironment: return 100 ;
      case _numbOfReg: return totNumbOfReg ;
      case _maxNumbOfIter: return 250 ;
      case _maxNumbOfNonImp: return 100 ;
      case _numbOfBestCan: return 0 ;
      case _avgDistInput: return 0 ;
      case _numbOfNeg: return 10 ;
      case _numbOfRow: return 0 ;
      case _numbOfCol: return 0 ;
      case _maxNumbOfIterSGS: return 0 ;
      case _maxNumbOfNonImpSGS: return 0 ;
      default: return super.getShapeControlDefaultValueDouble( _shape, index
);}
  @Override
  public String getNameOfShape( int _shape ) {
    switch( _shape ) {
      case regionManagerAgents_Presentation: return
"regionManagerAgents_Presentation";
      case vehicle_presentation: return "vehicle_presentation";
      case localVehicle_presentation: return "localVehicle_presentation";
      default: return super.getNameOfShape( _shape );
    }
  }

  @Override
  public int getShapeType( int _shape ) {
    switch( _shape ) {
      case regionManagerAgents_Presentation: return SHAPE_EMBEDDED_OBJECT;
      case vehicle_presentation: return SHAPE_EMBEDDED_OBJECT;
      case localVehicle_presentation: return SHAPE_EMBEDDED_OBJECT;
      default: return super.getShapeType( _shape );
    }
  }
  @Override
  public int getShapeReplication( int _shape ) {
    switch( _shape ) {
      case regionManagerAgents_Presentation: return
regionManagerAgents.size() ;
      case localVehicle_presentation: return
localVehicle.size() ;
      default: return super.getShapeReplication( _shape );
    }
  }

  @Override
  public double getShapeX( int _shape, int index ) {
    switch( _shape ) {
      case regionManagerAgents_Presentation: return 0;
      case vehicle_presentation: return 0;
      case localVehicle_presentation: return 0;
      default: return super.getShapeX( _shape, index );
    }
  }

  @Override
  public double getShapeY( int _shape, int index ) {
    switch( _shape ) {
      case regionManagerAgents_Presentation: return 0;
      case vehicle_presentation: return 0;
      case localVehicle_presentation: return 0;
      default: return super.getShapeY( _shape, index );
    }
  }
  @Override
  public Object getShapeEmbeddedObject( int _shape ) {
```

```
    switch( _shape ) {
      case regionManagerAgents_Presentation: return regionManagerAgents;
      case vehicle_presentation: return vehicle;
      case localVehicle_presentation: return localVehicle;
      default: return super.getShapeEmbeddedObject( _shape );
    }}
  ShapeButton determineRegions;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _initNumbOfCity_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
    shape.setEnabled(
true );
    shape.setRange( getShapeControlMinimum( _initNumbOfCity ),
getShapeControlMaximum( _initNumbOfCity ) );}
  ShapeSlider initNumbOfCity;

  /**
   * <i>This method should not be called by user</i>
   */
  private void _lengthOfEnvironment_SetDynamicParams_xjal( ShapeSlider shape
) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _lengthOfEnvironment ),
getShapeControlMaximum( _lengthOfEnvironment ) );}
  ShapeSlider lengthOfEnvironment;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _widthOfEnvironment_SetDynamicParams_xjal( ShapeSlider shape
) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _widthOfEnvironment ),
getShapeControlMaximum( _widthOfEnvironment ) );}
  ShapeSlider widthOfEnvironment;
    /**
   * <i>This method should not be called by user</i>
   */
  private void _numbOfReg_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _numbOfReg ),
getShapeControlMaximum( _numbOfReg ) );}
  ShapeSlider numbOfReg;
  ShapeButton convertCord;
  ShapeButton createAgentsButton;
  ShapeRadioButtonGroup numbOfRegOpt;
  ShapeComboBox avgDistOpt;
  ShapeComboBox regDistOpt;
  ShapeComboBox regionDetAlg;
  ShapeRadioButtonGroup homeCityOpt;
  ShapeTextField homeCityVal;
  ShapeCheckBox setInitialParameters;
  ShapeComboBox searchAlgorithms;
  ShapeTextField searchAlgP;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _maxNumbOfIter_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _maxNumbOfIter ),
getShapeControlMaximum( _maxNumbOfIter ) );}
  ShapeSlider maxNumbOfIter;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _maxNumbOfNonImp_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
```

```
    shape.setRange( getShapeControlMinimum( _maxNumbOfNonImp ),
getShapeControlMaximum( _maxNumbOfNonImp ) );}
  ShapeSlider maxNumbOfNonImp;

  /**
   * <i>This method should not be called by user</i>
   */
  private void _numbOfBestCan_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _numbOfBestCan ),
getShapeControlMaximum( _numbOfBestCan ) );}
  ShapeSlider numbOfBestCan;
  ShapeComboBox neighOpt;
  ShapeComboBox regCenterOpt;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _avgDistInput_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _avgDistInput ),
getShapeControlMaximum( _avgDistInput ) );}
  ShapeSlider avgDistInput;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _numbOfNeg_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _numbOfNeg ),
getShapeControlMaximum( _numbOfNeg ) )}
  ShapeSlider numbOfNeg;
  ShapeRadioButtonGroup initialSolStrategyGMA;
  ShapeComboBox problemType;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _numbOfRow_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _numbOfRow ),
getShapeControlMaximum( _numbOfRow ) );}
  ShapeSlider numbOfRow;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _numbOfCol_SetDynamicParams_xjal( ShapeSlider shape ) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _numbOfCol ),
getShapeControlMaximum( _numbOfCol ) );
  }
  ShapeSlider numbOfCol;
  ShapeRadioButtonGroup optStrategies;
  ShapeCheckBox changeReportFileLocation;
  ShapeTextField fileLocation;
  ShapeCheckBox avoidSameMembers;

  /**
   * <i>This method should not be called by user</i>
   */
  private void _maxNumbOfIterSGS_SetDynamicParams_xjal( ShapeSlider shape )
{
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _maxNumbOfIterSGS ),
getShapeControlMaximum( _maxNumbOfIterSGS ) );}
  ShapeSlider maxNumbOfIterSGS;

  /**
   * <i>This method should not be called by user</i>
   */
```

```
  private void _maxNumbOfNonImpSGS_SetDynamicParams_xjal( ShapeSlider shape
) {
    boolean _visible;
    shape.setRange( getShapeControlMinimum( _maxNumbOfNonImpSGS ),
getShapeControlMaximum( _maxNumbOfNonImpSGS ) );}
  ShapeSlider maxNumbOfNonImpSGS
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispLOfEnv_SetDynamicParams_xjal( ShapeText shape ) {
    boolean _visible;
    shape.setText(
"Length of environment: " +(int) lengthOfEnvironment.getValue()
);}
  ShapeText dispLOfEnv;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispWOfEnv_SetDynamicParams_xjal( ShapeText shape ) {
    boolean _visible;
    shape.setText(
"Width of environment: " + (int)widthOfEnvironment.getValue() );}
  ShapeText dispWOfEnv;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispNumbOfCityR_SetDynamicParams_xjal( ShapeText shape ) {
    boolean _visible;
    shape.setText(
infCity + (int) initNumbOfCity.getValue()   );}
  ShapeText dispNumbOfCityR;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispNumbOfReg_SetDynamicParams_xjal( ShapeText shape) {
    boolean _visible;
    shape.setText(
(int)numbOfReg.getValue() );}
  ShapeText dispNumbOfReg;
  /**
   * <i>This method should not be called by user</i>
   */
  public ShapeOval _cityP_createShapeWithStaticProperties( final int _index
) {
    ShapeOval shape = new ShapeOval(
       true,54, -211, 0.0,
       black, white,
          6, 5,
          1, LINE_STYLE_SOLID) {
      @Override
      public boolean onClick( double clickx, double clicky ) {
        return onShapeClick( _cityP, _index, clickx, clicky );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1323105401836L;};
    return shape;}
  /**
   * <i>This method should not be called by user</i>
   */
  private int _cityP_Replication() {
    return 1000 ;}
   ReplicatedShape<ShapeOval> cityP;
  /**
   * <i>This method should not be called by user</i>
   */
  public ShapeText _xCordCity_createShapeWithStaticProperties( final int
_index ) {
```

```
    ShapeText shape = new ShapeText(true,108,-216,0.0, black,"text",
        _xCordCity_Font, ALIGNMENT_LEFT);
    return shape;}
  /**
   * <i>This method should not be called by user</i>
   */
  private int _xCordCity_Replication() {
    return 1000 ;}
  ReplicatedShape<ShapeText> xCordCity;
  /**
   * <i>This method should not be called by user</i>
   */
  public ShapeText _xCordLabel_createShapeWithStaticProperties( final int
_index ) {
    ShapeText shape = new ShapeText(
        true,138, -216, 0.0,
        black,"text",
        _xCordLabel_Font, ALIGNMENT_LEFT);
    return shape;}
  /**
   * <i>This method should not be called by user</i>
   */
  private int _xCordLabel_Replication() {
    return 1000 ;}
   ReplicatedShape<ShapeText> xCordLabel;
  /**
   * <i>This method should not be called by user</i>
   */
  public ShapeText _yCordLabel_createShapeWithStaticProperties( final int
_index ) {
    ShapeText shape = new ShapeText(
        true,168, -216, 0.0,
        black,"text",
        _yCordLabel_Font, ALIGNMENT_LEFT
    );return shape;}
  /**
   * <i>This method should not be called by user</i>
   */
  private int _yCordLabel_Replication() {return 1000 ;}
   ReplicatedShape<ShapeText> yCordLabel;
  /**
   * <i>This method should not be called by user</i>
   */
  public ShapeText _yCordCity_createShapeWithStaticProperties( final int
_index ) {
    ShapeText shape = new ShapeText(
        true,198, -216, 0.0,
        black,"text",
        _yCordCity_Font, ALIGNMENT_LEFT);
    return shape;}
  /**
   * <i>This method should not be called by user</i>
   */
  private int _yCordCity_Replication() {
    return 1000 ;}
  ReplicatedShape<ShapeText> yCordCity;
  ShapeText dispSearchAlgP;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispMaxNOfIter_SetDynamicParams_xjal( ShapeText shape ) {
    boolean _visible;
    shape.setText(
"Maximum number of iterations: " +(int) maxNumbOfIter.getValue()
);}
  ShapeText dispMaxNOfIter;
  /**
   * <i>This method should not be called by user</i>
```

```
   */
  private void _dispMaxNOfNonImp_SetDynamicParams_xjal( ShapeText shape ) {
    boolean _visible;
    shape.setText(
"Maximum number of non improved solutions: " +(int)
maxNumbOfNonImp.getValue() );}
  ShapeText dispMaxNOfNonImp;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispMaxNOfBestCanSol_SetDynamicParams_xjal( ShapeText shape
) {
    boolean _visible;
    shape.setText(
"Number of best candidate solutions stored: " +(int)numbOfBestCan.getValue()
);}
  ShapeText dispMaxNOfBestCanSol;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispMultConst_SetDynamicParams_xjal( ShapeText shape) {
    boolean _visible;
    shape.setText(
"Multiply by "+numbOfReg.getValue() );}
  ShapeText dispMultConst;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispAvgDist_SetDynamicParams_xjal( ShapeText shape ) {
    boolean _visible;
    shape.setText(
"Average distance between cities "+avgDistInput.getValue() );}
  ShapeText dispAvgDist;
    /**
   * <i>This method should not be called by user</i>
   */
  private void _dispNumbOfNeg_SetDynamicParams_xjal( ShapeText shape) {
    boolean _visible;
    shape.setText(
"Number of neighbours (population size)"+(int)numbOfNeg.getValue() );}
  ShapeText dispNumbOfNeg;
    /**
   * <i>This method should not be called by user</i>
   */
  private void _dispNumbOfRow_SetDynamicParams_xjal( ShapeText shape) {
    boolean _visible;
    shape.setText(
(int)numbOfRow.getValue() );}
  ShapeText dispNumbOfRow;  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispNumbOfCol_SetDynamicParams_xjal( ShapeText shape) {
    boolean _visible;
    shape.setText(
(int)numbOfCol.getValue() );}
  ShapeText dispNumbOfCol;
  /**
   * <i>This method should not be called by user</i>
   */
  public ShapeText _dispAgentNumber_createShapeWithStaticProperties( final
int _index ) {
    ShapeText shape = new ShapeText(
        true,50, 530, 0.0,
        black,"",_dispAgentNumber_Font, ALIGNMENT_LEFT);
    return shape;}
  /**
   * <i>This method should not be called by user</i>
   */
```

```
  private int _dispAgentNumber_Replication() {
    return 1000 ;}
   ReplicatedShape<ShapeText> dispAgentNumber;
  /**
   * <i>This method should not be called by user</i>
   */
  public ShapeText _dispRegName_createShapeWithStaticProperties( final int
_index ) {
    ShapeText shape = new ShapeText(
        true,70, 560, 0.0,
        black," ",
        _dispRegName_Font, ALIGNMENT_LEFT);
    return shape; }
  /**
   * <i>This method should not be called by user</i>
   */
  private int _dispRegName_Replication() {
    return 1000 ;}
   ReplicatedShape<ShapeText> dispRegName;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispMaxNumbOfIterSGS_SetDynamicParams_xjal( ShapeText shape
) {
    boolean _visible;
    shape.setText(
"Maximum number of iterations for the same group sequence (GTSP): " +(int)
maxNumbOfIterSGS.getValue() );}
  ShapeText dispMaxNumbOfIterSGS;
  /**
   * <i>This method should not be called by user</i>
   */
  private void _dispMaxNumbOfNonImpSGS_SetDynamicParams_xjal( ShapeText
shape ) {
    boolean _visible;
    shape.setText(
"Maximum number of non improved solutions for the same group sequence
(GTSP): " +(int) maxNumbOfNonImpSGS.getValue() );}
  ShapeText dispMaxNumbOfNonImpSGS;
  /**
   * <i>This method should not be called by user</i>
   */
  public ShapeArc _regCenterP_createShapeWithStaticProperties( final int
_index ) {
    ShapeArc shape = new ShapeArc(
     true,300, -197, 2.0943951023931953, magenta, _regCenterP_FillColor,8,
7, 0.5235987755982988, 4.1887902047863905,
                    1, LINE_STYLE_SOLID );
    return shape}
  /**
   * <i>This method should not be called by user</i>
   */
  private int _regCenterP_Replication() {
    return 10000 ;}
  ReplicatedShape<ShapeArc> regCenterP;
  // Static initialization of persistent elements
  {determineRegions = new ShapeButton(
                    Main.this, true, 940, 340,
                    120, 30,
            controlDefault, controlDefault,
            _determineRegions_Font,
                    "START") {
      @Override
      public void action(){
        executeShapeControlAction( _determineRegions, 0 );
      }
      /**
       * This number is here for model snapshot storing purpose
```

```
       */
     private static final long serialVersionUID = 1318380781150L;};
    initNumbOfCity = new ShapeSlider(Main.this, true,170, 60,100, 40,silver,
false, getShapeControlMinimum( _initNumbOfCity ), getShapeControlMaximum(
_initNumbOfCity ), ShapeControl.TYPE_INT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _initNumbOfCity_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly ); }
      @Override
      public void action(){
        executeShapeControlAction( _initNumbOfCity, 0, value )}
      @Override
      public void setValueToDefault() {
       setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_initNumbOfCity, 0 ), getMax() ));}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1318380781156L;};
    lengthOfEnvironment = new ShapeSlider(Main.this, true,30,160,
                    100, 40,transparent, false, getShapeControlMinimum(
_lengthOfEnvironment ), getShapeControlMaximum( _lengthOfEnvironment ),
ShapeControl.TYPE_DOUBLe) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _lengthOfEnvironment_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_lengthOfEnvironment, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1318380781162L;};
    widthOfEnvironment = new ShapeSlider(Main.this, true,30, 70,
             100, 40,transparent,
            false, getShapeControlMinimum( _widthOfEnvironment ),
getShapeControlMaximum( _widthOfEnvironment ), ShapeControl.TYPE_DOUBLE) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _widthOfEnvironment_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly )}
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_widthOfEnvironment, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1318380781166L;};
    numbOfReg = new ShapeSlider(
                Main.this, true,340, 220, 100, 30,
           transparent,
           false, getShapeControlMinimum( _numbOfReg ),
getShapeControlMaximum( _numbOfReg ), ShapeControl.TYPE_INT)
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _numbOfReg_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly )}
      @Override
      public void action(){
        executeShapeControlAction( _numbOfReg, 0, value )}
```

```
      @Override
      public void setValueToDefault() {setValue( limit( getMin(),
getShapeControlDefaultValueDouble( _numbOfReg, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1318380781170L;};
    convertCord = new ShapeButton(
                    Main.this, true, -1798, 118,
                    198, 82,
            controlDefault, controlDefault,
            _convertCord_Font,
                    "Convert to Original") {
      @Override
      public void action(){
        executeShapeControlAction( _convertCord, 0 );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1318876894058L;};
    createAgentsButton = new ShapeButton(Main.this, true, -1800, 6,
                    340, 74,
            controlDefault, controlDefault,
            _createAgentsButton_Font,
                    "Create Agents") {
      @Override
      public void action(){
        executeShapeControlAction( _createAgentsButton, 0 );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1322324803457L;};
    numbOfRegOpt = new ShapeRadioButtonGroup(Main.this, true,160, 150, 360,
41,transparent, controlDefault,_numbOfRegOpt_Font, false,
            new String[]{"Number of region is not known", "Number of region
is known", } ) {
      @Override
      public void action(){executeShapeControlAction( _numbOfRegOpt, 0,
value );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324128531373L;};
    avgDistOpt = new ShapeComboBox(Main.this, true, 160, 220,
                    130, 20,controlDefault, controlDefault,
            _avgDistOpt_Font,
            new String[]{"Euclidean distance", "Circle diameter", "User
defined", }, false, ShapeControl.TYPE_STRING
    ) {
      @Override
      public void action(){
        executeShapeControlAction( _avgDistOpt, 0, value );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324129533183L;};
    regDistOpt = new ShapeComboBox(
                  Main.this, true, 160, 190,
                    130, 20,
            controlDefault, controlDefault,
            _regDistOpt_Font,
            new String[]{"Equal distance", "Modified equal distance (Next
minimum)", "Modified equal distance (Multiply constant)", }, false,
ShapeControl.TYPE_STRING
    ) {
      @Override
      public void action(){
        executeShapeControlAction( _regDistOpt, 0, value );
```

```
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324132415422L;};
    regionDetAlg = new ShapeComboBox(Main.this, true, 570, 190,
                    100, 20,
            controlDefault, controlDefault,
            _regionDetAlg_Font,
            new String[]{"K-Means", }, false, ShapeControl.TYPE_STRING
    );
    homeCityOpt = new ShapeRadioButtonGroup(
                 Main.this, true,300, 40,
                    200, 100,
            transparent, controlDefault,
            _homeCityOpt_Font, true,
            new String[]{"Home city (or region) is the first city(or
region)", "Home city (or region) is the given city(or region)", "Home city
(or region) can be any city (or region)", }
    ) {

      @Override
      public void action(){
        executeShapeControlAction( _homeCityOpt, 0, value );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324135076149L;
    };
    homeCityVal = new ShapeTextField(
                    Main.this, true,480, 70,
                    30, 20,
            controlDefault, controlDefault, _homeCityVal_Font
    ) {

      @Override
      public void action(){
        executeShapeControlAction( _homeCityVal, 0, value );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324135152641L;
    };
    setInitialParameters = new ShapeCheckBox(
             Main.this,true,950, 310,
             100, 30,
            transparent, controlDefault,
            _setInitialParameters_Font,
                    "Initialization is completed"
    ) {
      @Override
      public void action(){
        executeShapeControlAction( _setInitialParameters, 0, value)}
      @Override
      public void setValueToDefault() {
             setSelected( getShapeControlDefaultValueBoolean(
_setInitialParameters, 0 ) );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324137324661L;};
    searchAlgorithms = new ShapeComboBox(
                 Main.this, true, 800, 40,
                    100, 20,
            controlDefault, controlDefault,
```

```
            _searchAlgorithms_Font,
            new String[]{"GDA", }, false, ShapeControl.TYPE_STRING
    ) {
      @Override
      public void action(){
        executeShapeControlAction( _searchAlgorithms, 0, value );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324138032347L;};
    searchAlgP = new ShapeTextField(Main.this, true,1040, 90,
                  30, 20,
            controlDefault, controlDefault, _searchAlgP_Font) {
      @Override
      public void action(){
        executeShapeControlAction( _searchAlgP, 0, value );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324140230696L;};
    maxNumbOfIter = new ShapeSlider(Main.this, true,700, 270,
                  100, 30,transparent,
            false, getShapeControlMinimum( _maxNumbOfIter ),
getShapeControlMaximum( _maxNumbOfIter ), ShapeControl.TYPE_DOUBLE) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _maxNumbOfIter_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );
      }
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_maxNumbOfIter, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324141052419L;
    };
    maxNumbOfNonImp = new ShapeSlider(
                Main.this, true,700, 320,
                  100, 30,
            transparent,
            false, getShapeControlMinimum( _maxNumbOfNonImp ),
getShapeControlMaximum( _maxNumbOfNonImp ), ShapeControl.TYPE_DOUBLE
    ) {

      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _maxNumbOfNonImp_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );
      }

      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_maxNumbOfNonImp, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324141250357L;};
    numbOfBestCan = new ShapeSlider(
                Main.this, true,700, 220,
                  110, 30,
            transparent,
```

```
            false, getShapeControlMinimum( _numbOfBestCan ),
getShapeControlMaximum( _numbOfBestCan ), ShapeControl.TYPE_DOUBLE) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _numbOfBestCan_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_numbOfBestCan, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324141319810L;};
    neighOpt = new ShapeComboBox(Main.this, true, 800, 150,
                  100, 20,controlDefault, controlDefault,
            _neighOpt_Font,
            new String[]{"Change two cities", "CrossOver(Point)", "2-Opt",
}, false, ShapeControl.TYPE_STRING
    ) {
      @Override
      public void action(){
        executeShapeControlAction( _neighOpt, 0, value );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324142068495L;};
    regCenterOpt = new ShapeComboBox(
                Main.this, true, 160, 250,
                  130, 20,
            controlDefault, controlDefault,
            _regCenterOpt_Font,
            new String[]{"First selected city", "Center of gravity",},
false, ShapeControl.TYPE_STRING
    );
    avgDistInput = new ShapeSlider(
                Main.this, true,340, 260,
                  100, 30,
            transparent,
            false, getShapeControlMinimum( _avgDistInput ),
getShapeControlMaximum( _avgDistInput ), ShapeControl.TYPE_DOUBLE
    ) {

      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _avgDistInput_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );
      }

      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_avgDistInput, 0 ), getMax() ) );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324455560451L;
    };
    numbOfNeg = new ShapeSlider(
                Main.this, true,690, 100,
                  100, 30,
            transparent,
            false, getShapeControlMinimum( _numbOfNeg ),
getShapeControlMaximum( _numbOfNeg ), ShapeControl.TYPE_DOUBLE
    )
```

```
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _numbOfNeg_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_numbOfNeg, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324999383894L;};
    initialSolStrategyGMA = new ShapeRadioButtonGroup(
                Main.this, true,510, 50,
                   150, 50,
            transparent, controlDefault,
            _initialSolStrategyGMA_Font, true,
            new String[]{"Best tour of each agent", "GMA determines", }) {
      @Override
      public void action(){
        executeShapeControlAction( _initialSolStrategyGMA, 0, value );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1325150525975L;
    };
    problemType = new ShapeComboBox(
                Main.this, true, 30, 250, 120, 20,
            controlDefault, controlDefault,
            _problemType_Font,
            new String[]{"Dynamic TSP", "Dynamic GTSP (Regions are known)",
"Dynamic GTSP (Regions are not known)", "Dynamic GTSP with TSP (Regions are
known)", "Dynamic GTSP with TSP (Regions are not known)", }, false,
ShapeControl.TYPE_STRING
    ) {
      @Override
      public void action(){
        executeShapeControlAction( _problemType,0,value);}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1326893828651L;};
    numbOfRow = new ShapeSlider(
                Main.this, true,150, 310,
                   100, 30,
            transparent,
            false, getShapeControlMinimum( _numbOfRow ),
getShapeControlMaximum( _numbOfRow ), ShapeControl.TYPE_DOUBLE
    ) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _numbOfRow_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_numbOfRow, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1326897296522L;};
    numbOfCol = new ShapeSlider(Main.this, true,150, 340,
                   100, 30,transparent, false, getShapeControlMinimum(
_numbOfCol ), getShapeControlMaximum( _numbOfCol ),
ShapeControl.TYPE_DOUBLE) {
      @Override
```

```
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _numbOfCol_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_numbOfCol, 0 ), getMax() ) );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1326897307841L;
    };optStrategies = new ShapeRadioButtonGroup(
                Main.this, true,490, 230,
                   150, 50,
            transparent, controlDefault,
            _optStrategies_Font, true,
            new String[]{"Strategy1(NextTour)", "Strategy2(SameTour)", });
    changeReportFileLocation = new ShapeCheckBox(
             Main.this,true,900, 200,
             160, 50,
            transparent, controlDefault,
            _changeReportFileLocation_Font,
                    "Change Report File Location) {
      @Override
      public void action(){
        executeShapeControlAction( _changeReportFileLocation, 0, value );
      }

      @Override
      public void setValueToDefault() {
             setSelected( getShapeControlDefaultValueBoolean(
_changeReportFileLocation, 0 ) );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1331837354227L};
    fileLocation = new ShapeTextField(
                  Main.this, true,910, 240,
                  100, 30,
           controlDefault, controlDefault, _fileLocation_Font);
    avoidSameMembers = new ShapeCheckBox(
             Main.this,true,690, 120,
             100, 30,
            transparent, controlDefault,
            _avoidSameMembers_Font,
                    "Avoid same members");
    maxNumbOfIterSGS = new ShapeSlider(
                Main.this, true,500, 310,
                   100, 20,
            transparent,
            false, getShapeControlMinimum( _maxNumbOfIterSGS ),
getShapeControlMaximum( _maxNumbOfIterSGS ), ShapeControl.TYPE_DOUBLE
    ) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _maxNumbOfIterSGS_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_maxNumbOfIterSGS,0 ), getMax()));}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1333550299348L;
```

```
    };
    maxNumbOfNonImpSGS = new ShapeSlider(
                Main.this, true,500, 360, 100, 20,
            transparent,
            false, getShapeControlMinimum( _maxNumbOfNonImpSGS ),
getShapeControlMaximum( _maxNumbOfNonImpSGS ), ShapeControl.TYPE_DOUBLE) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _maxNumbOfNonImpSGS_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );
      @Override
      public void setValueToDefault() {
             setValue( limit( getMin(), getShapeControlDefaultValueDouble(
_maxNumbOfNonImpSGS, 0 ), getMax() ) );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1333550408108L;};
    dispLOfEnv = new ShapeText(
        true,40, 130, 0.0,
        black," ",
        _dispLOfEnv_Font, ALIGNMENT_LEFT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispLOfEnv_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly )}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1317258193986L;
    };
    dispWOfEnv = new ShapeText(
        true,40, 50, 0.0,
        black," ",
        _dispWOfEnv_Font, ALIGNMENT_LEFT
    ) {

      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispWOfEnv_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1317258488278L;
    };
    dispNumbOfCityR = new ShapeText(
        true,170, 110, 0.0,
        black,"",
        _dispNumbOfCityR_Font, ALIGNMENT_LEFT
    ) {

      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispNumbOfCityR_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1318380781140L;
    };
```

```
    dispNumbOfReg = new ShapeText(
        true,340, 190, 0.0, black,"",
        _dispNumbOfReg_Font, ALIGNMENT_LEFT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispNumbOfReg_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly )}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1318380781145L;};
    cityP = new ReplicatedShape<ShapeOval>() {
      @Override
      public int getReplication() {
        return _cityP_Replication()}
      @Override
      public ShapeOval createShapeWithStaticProperties( int index ) {return
_cityP_createShapeWithStaticProperties( index )}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1323105401836L;};
    xCordCity = new ReplicatedShape<ShapeText>() {
      @Override
      public int getReplication() {
      return _xCordCity_Replication();
      @Override
      public ShapeText createShapeWithStaticProperties( int index ) {return
_xCordCity_createShapeWithStaticProperties( index );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1323105401841};
    xCordLabel = new ReplicatedShape<ShapeText>() {
      @Override
      public int getReplication() {
        return _xCordLabel_Replication();}
          @Override
      public ShapeText createShapeWithStaticProperties( int index ) {
return _xCordLabel_createShapeWithStaticProperties( index );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1323105401845L;};
    yCordLabel = new ReplicatedShape<ShapeText>() {
      @Override
      public int getReplication() {
      return _yCordLabel_Replication();}
      @Override
      public ShapeText createShapeWithStaticProperties( int index ){
        return _yCordLabel_createShapeWithStaticProperties( index )}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1323105401850L;};
    yCordCity = new ReplicatedShape<ShapeText>() {
      @Override
      public int getReplication() {
        return _yCordCity_Replication();}
      @Override
      public ShapeText createShapeWithStaticProperties( int index ) {return
_yCordCity_createShapeWithStaticProperties( index );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1323105401854L;};
```

```
    dispSearchAlgP = new ShapeText(true,790, 100, 0.0, black,"",
        _dispSearchAlgP_Font, ALIGNMENT_LEFT);
    dispMaxNOfIter = new ShapeText(true,710, 260, 0.0, black," ",
        _dispMaxNOfIter_Font, ALIGNMENT_LEFT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispMaxNOfIter_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324140991129L;};
    dispMaxNOfNonImp = new ShapeText(
        true,710, 300, 0.0,
        black," ",
        _dispMaxNOfNonImp_Font, ALIGNMENT_LEFT
    ) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispMaxNOfNonImp_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly ); }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324141226890L};
    dispMaxNOfBestCanSol = new ShapeText(
        true,710, 210, 0.0,
        black," ",
        _dispMaxNOfBestCanSol_Font, ALIGNMENT_LEF) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispMaxNOfBestCanSol_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324141303954L;};
    dispMultConst = new ShapeText(
        true,340, 200, 0.0,
        black,"",
        _dispMultConst_Font, ALIGNMENT_LEFT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispMultConst_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324219125336L};
    dispAvgDist = new ShapeText(true,350, 240, 0.0,
        black," ",
        _dispAvgDist_Font, ALIGNMENT_LEFT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispAvgDist_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1324455490027L;};
    dispNumbOfNeg = new ShapeText(
        true,690, 70, 0.0,
        black,"",
```

```
        _dispNumbOfNeg_Font, ALIGNMENT_LEFT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispNumbOfNeg_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1325003240274L;};
    dispNumbOfRow = new ShapeText(
        true,50, 320, 0.0,
        black,"dispNumOfRow",
        _dispNumbOfRow_Font, ALIGNMENT_LEFT
    ) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispNumbOfRow_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1326897207291L;
    };
    dispNumbOfCol = new ShapeText(
        true,50, 350, 0.0,
        black,"dispNumbOfCol",
        _dispNumbOfCol_Font, ALIGNMENT_LEFT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ){ _dispNumbOfCol_SetDynamicParams_xjal( this
);super.draw( panel, graphics, xform, publicOnly );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1326897254648L;};
    dispAgentNumber = new ReplicatedShape<ShapeText>() {
      @Override
      public int getReplication() {
        return _dispAgentNumber_Replication();
      }

      @Override
      public ShapeText createShapeWithStaticProperties( int index ) {
        return _dispAgentNumber_createShapeWithStaticProperties( index );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1329148420294L;};
    dispRegName = new ReplicatedShape<ShapeText>() {
      @Override
      public int getReplication() {
        return _dispRegName_Replication();}
      @Override
      public ShapeText createShapeWithStaticProperties( int index ){
        return _dispRegName_createShapeWithStaticProperties( index );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1330793590639L;};
    dispMaxNumbOfIterSGS = new ShapeText(true,500, 290, 0.0,
        black," ",
        _dispMaxNumbOfIterSGS_Font, ALIGNMENT_LEFT
    ) {
      @Override
```

```
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispMaxNumbOfIterSGS_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );
      }
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1333550892905L;
    };
    dispMaxNumbOfNonImpSGS = new ShapeText(true,500, 350, 0.0,
        black," ",
        _dispMaxNumbOfNonImpSGS_Font, ALIGNMENT_LEFT) {
      @Override
      public void draw( Panel panel, Graphics2D graphics, AffineTransform
xform, boolean publicOnly ) {
        _dispMaxNumbOfNonImpSGS_SetDynamicParams_xjal( this );
        super.draw( panel, graphics, xform, publicOnly );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1333551022584L;};
    regCenterP = new ReplicatedShape<ShapeArc>() {
      @Override
      public int getReplication() {
        return _regCenterP_Replication();}
      @Override
      public ShapeArc createShapeWithStaticProperties( int index ) {
      return _regCenterP_createShapeWithStaticProperties( index );}
      /**
       * This number is here for model snapshot storing purpose
       */
      private static final long serialVersionUID = 1336304804382L;
    };}
  ShapeGroup presentation;
  ShapeGroup icon;
  @Override
  public Object getPersistentShape( int _shape ) {
    switch(_shape){
      case _presentation: return presentation;
      case _icon: return icon;
      case _determineRegions: return determineRegions;
      case _initNumbOfCity: return initNumbOfCity;
      case _lengthOfEnvironment: return lengthOfEnvironment;
      case _widthOfEnvironment: return widthOfEnvironment;
      case _numbOfReg: return numbOfReg;
      case _convertCord: return convertCord;
      case _createAgentsButton: return createAgentsButton;
      case _numbOfRegOpt: return numbOfRegOpt;
      case _avgDistOpt: return avgDistOpt;
      case _regDistOpt: return regDistOpt;
      case _regionDetAlg: return regionDetAlg;
      case _homeCityOpt: return homeCityOpt;
      case _homeCityVal: return homeCityVal;
      case _setInitialParameters: return setInitialParameters;
      case _searchAlgorithms: return searchAlgorithms;
      case _searchAlgP: return searchAlgP;
      case _maxNumbOfIter: return maxNumbOfIter;
      case _maxNumbOfNonImp: return maxNumbOfNonImp;
      case _numbOfBestCan: return numbOfBestCan;
      case _neighOpt: return neighOpt;
      case _regCenterOpt: return regCenterOpt;
      case _avgDistInput: return avgDistInput;
      case _numbOfNeg: return numbOfNeg;
      case _initialSolStrategyGMA: return initialSolStrategyGMA;
      case _problemType: return problemType;
      case _numbOfRow: return numbOfRow;
      case _numbOfCol: return numbOfCol;
```

```
        case _optStrategies: return optStrategies;
        case _changeReportFileLocation: return changeReportFileLocation;
        case _fileLocation: return fileLocation;
        case _avoidSameMembers: return avoidSameMembers;
        case _maxNumbOfIterSGS: return maxNumbOfIterSGS;
        case _maxNumbOfNonImpSGS: return maxNumbOfNonImpSGS;
        case _dispLOfEnv: return dispLOfEnv;
        case _dispWOfEnv: return dispWOfEnv;
        case _dispNumbOfCityR: return dispNumbOfCityR;
        case _dispNumbOfReg: return dispNumbOfReg;
        case _cityP: return cityP;
        case _xCordCity: return xCordCity;
        case _xCordLabel: return xCordLabel;
        case _yCordLabel: return yCordLabel;
        case _yCordCity: return yCordCity;
        case _dispSearchAlgP: return dispSearchAlgP;
        case _dispMaxNOfIter: return dispMaxNOfIter;
        case _dispMaxNOfNonImp: return dispMaxNOfNonImp;
        case _dispMaxNOfBestCanSol: return dispMaxNOfBestCanSol;
        case _dispMultConst: return dispMultConst;
        case _dispAvgDist: return dispAvgDist;
        case _dispNumbOfNeg: return dispNumbOfNeg;
        case _dispNumbOfRow: return dispNumbOfRow;
        case _dispNumbOfCol: return dispNumbOfCol;
        case _dispAgentNumber: return dispAgentNumber;
        case _dispRegName: return dispRegName;
        case _dispMaxNumbOfIterSGS: return dispMaxNumbOfIterSGS;
        case _dispMaxNumbOfNonImpSGS: return dispMaxNumbOfNonImpSGS;
        case _regCenterP: return regCenterP;
        default: return null; }}
  @Override
  public void drawModelElements(Panel _panel, Graphics2D _g, boolean
_publicOnly ) {
    if (!_publicOnly) {drawEvent( _panel, _g, 2220, 270, -20, 10,
"newEvent", newEvent );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2070, 30, 10, 0,
"maxX", maxX, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2070, 60, 10, 0,
"maxY", maxY, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2190, 80, 10, 0,
"rangeX", rangeX, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2190, 110, 10, 0,
"rangeY", rangeY, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2070, 90, 10, 0,
"maxXCity", maxXCity, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2070, 120, 10, 0,
"maxYCity", maxYCity, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1970, 90, 10, 0,
"minXCity", minXCity, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1970, 120, 10, 0,
"minYCity", minYCity, false );}
    if (!_publicOnly){drawPlainVariable( _panel, _g, 1970, 150, 10, 0,
"avgDistBtwCities", avgDistBtwCities, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1610, 160, 10, 0,
"rndXCord", rndXCord, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1610, 130, 10, 0,
"rndYCord", rndYCord, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2190, 30, 10, 0,
"sumX", sumX, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2190, 50, 10, 0,
"sumY", sumY, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1970, 200, 10, 0,
"xCordOfHomeCity", xCordOfHomeCity, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1970, 230, 10, 0,
"yCordOfHomeCity", yCordOfHomeCity, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 380, 700, 10, 0,
"deltaB", deltaB, false );}
```

```
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2460, 210, 10, 0,
"simStartTime", simStartTime, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2570, 210, 10, 0,
"totNumbOfCityInTour", totNumbOfCityInTour, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2330, 240, 10, 0,
"newCityXCord", newCityXCord, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2430, 240, 10, 0,
"newCityYCord", newCityYCord, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2530, 240, 10, 0,
"newCityName", newCityName, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 380, 780, 10, 0,
"tempX", tempX, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 380, 800, 10, 0,
"tempY", tempY, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 380, 830, 10, 0,
"dispIndex", dispIndex, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2350, 130, 10, 0,
"neighType", neighType, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 100, 890, 10, 0,
"regCenterPInitialX", regCenterPInitialX, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 100, 920, 10, 0,
"regCenterPInitialY", regCenterPInitialY, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 260, 900, 10, 0,
"allCitiesIndex", allCitiesIndex, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2190, 160, 10, 0,
"cellXCord", cellXCord, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2190, 190, 10, 0,
"cellYCord", cellYCord, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1980, 260, 10, 0,
"cityOrRegion", cityOrRegion, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2660, 130, 10, 0,
"allRegionsIndex", allRegionsIndex, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1970, 30, 10, 0,
"minX", minX, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 1970, 60, 10, 0,
"minY", minY, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2350, 110, 10, 0,
"tourCounter", tourCounter, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2350, 80, 10, 0,
"numbOfCity", numbOfCity, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 90, 750, 10, 0,
"unVisitedString", unVisitedString, true );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 90, 770, 10, 0,
"visitedString", visitedString, true );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 90, 790, 10, 0,
"deletedString", deletedString, true );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 310, 30, 10, 0,
"homeCity", homeCity, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2330, 210, 10, 0,
"startToCreateEvent", startToCreateEvent, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 90, 820, 10, 0,
"neighborType", neighborType, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 380, 720, 10, 0,
"numbOfAttributes", numbOfAttributes, true );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 260, 870, 10, 0,
"constant1Or0", constant1Or0, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 2350, 150, 10, 0,
"partial", partial, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 100, 950, 10, 0,
"count", count, false );}
    if (!_publicOnly) {drawPlainVariable( _panel, _g, 280, 930, 10, 0,
"numbOfRegOptString", numbOfRegOptString, false );}
    if (!_publicOnly){drawPlainVariable( _panel,_g, 280, 950, 10,
0, "initialSolStrategyGMAString",initialSolStrategyGMAString,false );}
   if (!_publicOnly) {drawPlainVariable( _panel, _g, 2350, 40, 10, 0,
"costString", costString, true );}
   if (!_publicOnly) {drawCollection( _panel, _g, 2480, 80, 10, 0,
"RegionsAndCities", RegionsAndCities );}
```

```
    if (!_publicOnly) {drawCollection( _panel, _g, 2480, 30, 10, 0,
"CitiesDataBase", CitiesDataBase );}
    if (!_publicOnly) {drawCollection( _panel, _g, 2480, 60, 10, 0,
"RegionsSequence", RegionsSequence );}
    if (!_publicOnly) {drawCollection( _panel, _g, 2490, 110, 10, 0,
"InstCityNames", InstCityNames );}
    if (!_publicOnly) {drawActionChart( _panel, _g, 1790, 20, 20, 20, -30, -
10, null, null );
      drawActionChartLink( _panel, _g, 1790, 40, 70 );
      drawLoopArea( _panel, _g, 1680, 85,250, 235 );
      drawActionChartLink( _panel, _g, 1790, 300, 320 );
      drawLoopBlock( _panel, _g, 1790, 70, 160, 30, "for ( int i = 1; i <=
numbOfCity; i++ )", false, null );
      drawActionChartLink( _panel, _g, 1790, 100, 120 );
      drawCodeBlock( _panel, _g, 1790, 120, 180, 60,
"InitializationOfEnvironment:\nGenerate location for\npredetermined
numbOfCity", true, null );
      drawActionChartLink( _panel, _g, 1790, 180, 200 );
      drawDecisionBlock( _panel, _g, 1790, 200, 100, 40,
"Regions\nare\nknown\nor not", true, null, true, 70, 80, true, -70, 20, 100,
0 );
      drawActionChartLink( _panel, _g, 1860, 220, 250 );
      drawCodeBlock( _panel, _g, 1860, 250, 100, 30, "Regions are\nknown",
true, null );
      drawActionChartLink( _panel, _g, 1790, 320, 340 );
      drawCodeBlock( _panel, _g, 1790, 340, 100, 30,
"InstCityNames.removeFirst();\r\n", false, null );
      drawActionChartLink( _panel, _g, 1790, 370, 390 );
      drawDecisionBlock( _panel, _g, 1790, 390, 100, 40,
"Skip\nunnecessary\ncalculations", true, null, true, 70, 80, true, -130,
970, 990, 0 );
      drawActionChartLink( _panel, _g, 1860, 410, 440 );
      drawCodeBlock( _panel, _g, 1860, 440, 100, 30, "Print Regions\nand
Cities", true, null );
      drawActionChartLink( _panel, _g, 1660, 410, 440 );
      drawCodeBlock( _panel, _g, 1660, 440, 100, 30, "Make
Initial\nCalculations", true, null );
      drawActionChartLink( _panel, _g, 1660, 470, 490 );
      drawDecisionBlock( _panel, _g, 1660, 490, 100, 40, "Problem\nType",
true, null, true, 70, 80, true, -180, 850, 870, 0);
      drawActionChartLink( _panel, _g, 1730, 510, 540 );
      drawCodeBlock( _panel, _g, 1730, 540, 100, 30, "Dynamic TSP", true,
null );
      drawActionChartLink( _panel, _g, 1480, 510, 540 );
      drawCodeBlock( _panel, _g, 1480, 540, 100, 30, "Calculations
for\nGTSP", true, null );
      drawActionChartLink( _panel, _g, 1480, 570, 590 );
      drawDecisionBlock( _panel, _g, 1480, 590, 100, 40, "GTSP\nType", true,
null, true, 70, 80, true, -130, 610, 630, 0 );
      drawActionChartLink( _panel, _g, 1550, 610, 640 );
      drawCodeBlock( _panel, _g, 1550, 640, 100, 30, "GTSP Number of\nRegion
is Known", true, null );
      drawActionChartLink(_panel, _g, 1350, 610, 640 );
      drawCodeBlock(_panel,_g,1350, 640, 100, 30, "", false, null);
      drawActionChartLink( _panel, _g, 1350, 670, 690 );
      drawDecisionBlock( _panel, _g, 1350, 690, 100, 40,
"Average\nDistance\nuser\ndefined", true, null, true, 70, 80, true, -260,
490, 510, 0 );
      drawActionChartLink( _panel, _g, 1420, 710, 740 );
      drawCodeBlock( _panel, _g, 1420, 740, 100, 30, "User
defined\naverage\ndistance", true, null );
      drawActionChartLink( _panel, _g, 1090, 710, 740 );
      drawCodeBlock( _panel, _g, 1090, 740, 100, 30, "", false, null);
      drawActionChartLink( _panel, _g, 1090, 770, 790 );
      drawDecisionBlock( _panel, _g, 1090, 790, 100, 40,
"Average\ndistance\noption", true, null, true, 70, 80, true, -70, 80, 100, 0
);
      drawActionChartLink( _panel, _g, 1160, 810, 840 );
```

```
      drawCodeBlock( _panel, _g, 1160, 840, 100, 30, "Circle
diameter\ndistance", true, null );
      drawActionChartLink( _panel, _g, 1020, 810, 840 );
      drawCodeBlock( _panel, _g, 1020, 840, 100, 30, "Euclidean\ndistance",
true, null );
      drawActionChartLink( _panel, _g, 1090, 890, 910 );
      drawCodeBlock( _panel, _g, 1090, 910, 100, 30,
"print.writeToTxtFile(\"Average distance: \"+avgDistBtwCities);", false,
null );
      drawActionChartLink( _panel, _g, 1090, 940, 960 );
      drawDecisionBlock( _panel, _g, 1090, 960, 100, 40, "Modified\n
or\nequal\ndistance", true, null, true, 130, 200, true, -70, 80, 220, 0 );
      drawActionChartLink( _panel, _g, 1220, 980, 1010 );
      drawCodeBlock( _panel, _g, 1220, 1010, 100, 30, "", false, null );
      drawActionChartLink( _panel, _g, 1220, 1040, 1060 );
      drawDecisionBlock( _panel, _g, 1220, 1060, 100, 40,
"Next\nminimum\ndistance\nor\nmultiply\nconstant", true, null, true, 70, 80,
true, -70, 80, 100, 0 );
      drawActionChartLink( _panel, _g, 1290, 1080, 1110 );
      drawCodeBlock( _panel, _g, 1290, 1110, 100, 30, "Modified
equal\ndistance (Next\nminimum)", true, null );
      drawActionChartLink( _panel, _g, 1150, 1080, 1110 );
      drawCodeBlock( _panel, _g, 1150, 1110, 100, 30, "Modified
equal\ndistance\n(Multiply\nconstant)", true, null );
      drawActionChartLink( _panel, _g, 1020, 980, 1010 );
      drawCodeBlock( _panel, _g, 1020, 1010, 100, 30, "Equal distance",
true, null );
      drawActionChartLink( _panel, _g, 1480, 1220, 1240 );
      drawDecisionBlock( _panel, _g, 1480, 1240, 100, 40,
"Region\nCenter\nDecision",true,null,true,90,80,true,-70,20,100,0 );
      drawActionChartLink( _panel, _g, 1570, 1260, 1290 );
      drawCodeBlock( _panel, _g, 1570, 1290, 160, 30, "Region Center (Center
of\nGravity)", true, null );
      drawActionChartLink( _panel, _g, 1790, 1380, 1400 );
      drawReturnBlock(_panel,_g,1790,1400,80,30,null,false,null )}
    // Embedded object "generalManager"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault( _panel, _g, 60 ,-320 ,-52,-21,
"generalManager", this.generalManager );}
    // Embedded object "regionManagerAgents"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault(_panel,_g,190,-320,-71,-21,
"regionManagerAgents", this.regionManagerAgents )}
    // Embedded object "vehicle"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault(_panel, _g,60 ,-260 , -24, -21,
"vehicle", this.vehicle );}
    // Embedded object "localVehicle"
    if (!_publicOnly) {
      drawEmbeddedObjectModelDefault( _panel, _g, 190 , -270 , -40, -21,
"localVehicle", this.localVehicle );}
    if (!_publicOnly) {
      drawEnvironment( _panel, _g, 50, -130, 10, 0, "negotiationEnvCont",
negotiationEnvCont );}
  }
  @Override
  public boolean onClickModelAt( Panel panel, double x, double y, int
clickCount, boolean publicOnly ) {
    if( !publicOnly && modelElementContains(x, y, 2070, 30) ) {
      panel.addInspect( 2070, 30, this, "maxX" );
      return true;}
    if( !publicOnly && modelElementContains(x, y, 2070, 60) ) {
      panel.addInspect( 2070, 60, this, "maxY" );
      return true;}
    if( !publicOnly && modelElementContains(x, y, 2190, 80) ) {
      panel.addInspect( 2190, 80, this, "rangeX" );
      return true;}
    if( !publicOnly && modelElementContains(x, y, 2190, 110) ) {
```

```
    panel.addInspect( 2190, 110, this, "rangeY" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2070, 90) ) {
    panel.addInspect( 2070, 90, this, "maxXCity" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2070, 120) ) {
    panel.addInspect( 2070, 120, this, "maxYCity" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 1970, 90) ) {
    panel.addInspect( 1970, 90, this, "minXCity" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 1970, 120) ) {
    panel.addInspect( 1970, 120, this, "minYCity" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 1970, 150) ) {
    panel.addInspect( 1970, 150, this, "avgDistBtwCities" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 1610, 160) ) {
    panel.addInspect( 1610, 160, this, "rndXCord" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 1610, 130) ) {
    panel.addInspect( 1610, 130, this, "rndYCord" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2190, 30) ) {
    panel.addInspect( 2190, 30, this, "sumX" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2190, 50) ) {
    panel.addInspect( 2190, 50, this, "sumY" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 1970, 200) ) {
    panel.addInspect( 1970, 200, this, "xCordOfHomeCity" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 1970, 230) ) {
    panel.addInspect( 1970, 230, this, "yCordOfHomeCity" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 380, 700) ) {
    panel.addInspect( 380, 700, this, "deltaB" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2460, 210) ) {
    panel.addInspect( 2460, 210, this, "simStartTime" );
    return true; }
  if( !publicOnly && modelElementContains(x, y, 2570, 210) ) {
    panel.addInspect( 2570, 210, this, "totNumbOfCityInTour" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2330, 240) ) {
    panel.addInspect(2330, 240, this, "newCityXCord" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2430, 240) ) {
    panel.addInspect(2430, 240, this, "newCityYCord" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2530, 240) ) {
    panel.addInspect(2530, 240, this, "newCityName" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 380, 780) ) {
    panel.addInspect( 380, 780, this, "tempX" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 380, 800) ) {
    panel.addInspect( 380, 800, this, "tempY" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 380, 830) ) {
    panel.addInspect( 380, 830, this, "dispIndex" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 2350, 130) ) {
    panel.addInspect( 2350, 130, this, "neighType" );
    return true;}
  if( !publicOnly && modelElementContains(x, y, 100, 890) ) {
    panel.addInspect( 100, 890, this, "regCenterPInitialX" );
    return true;}
```

```
if( !publicOnly && modelElementContains(x, y, 100, 920) ) {
  panel.addInspect( 100, 920, this, "regCenterPInitialY" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 260, 900) ) {
  panel.addInspect( 260, 900, this, "allCitiesIndex" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 2190, 160) ) {
  panel.addInspect( 2190, 160, this, "cellXCord" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 2190, 190) ) {
  panel.addInspect( 2190, 190, this, "cellYCord" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 1980, 260) ) {
  panel.addInspect( 1980, 260, this, "cityOrRegion" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 2660, 130) ) {
  panel.addInspect( 2660, 130, this, "allRegionsIndex" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 1970, 30) ) {
  panel.addInspect( 1970, 30, this, "minX" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 1970, 60) ) {
  panel.addInspect( 1970, 60, this, "minY" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 2350, 110) ) {
  panel.addInspect( 2350, 110, this, "tourCounter" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 2350, 80) ) {
  panel.addInspect( 2350, 80, this, "numbOfCity" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 90, 750) ) {
  panel.addInspect( 90, 750, this, "unVisitedString" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 90, 770) ) {
  panel.addInspect( 90, 770, this, "visitedString" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 90, 790) ) {
  panel.addInspect( 90, 790, this, "deletedString" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 310, 30) ) {
  panel.addInspect( 310, 30, this, "homeCity" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 2330, 210) ) {
  panel.addInspect( 2330, 210, this, "startToCreateEvent" );
  return true; }
if( !publicOnly && modelElementContains(x, y, 90, 820) ) {
  panel.addInspect( 90, 820, this, "neighborType" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 380, 720) ) {
  panel.addInspect( 380, 720, this, "numbOfAttributes" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 260, 870) ) {
  panel.addInspect( 260, 870, this, "constant1Or0" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 2350, 150) ) {
  panel.addInspect( 2350, 150, this, "partial" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 100, 950) ) {
  panel.addInspect( 100, 950, this, "count" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 280, 930) ) {
  panel.addInspect( 280, 930, this, "numbOfRegOptString" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 280, 950) ) {
  panel.addInspect( 280, 950, this, "initialSolStrategyGMAString" );
  return true;}
if( !publicOnly && modelElementContains(x, y, 2350, 40) ) {
  panel.addInspect( 2350, 40, this, "costString" );
```

```
      return true;}
    if( !publicOnly && modelElementContains(x, y, 2480, 80) ) {
      panel.addInspect( 2480, 80, this, "RegionsAndCities" );
      return true;}
    if( !publicOnly && modelElementContains(x, y, 2480, 30) ) {
      panel.addInspect( 2480, 30, this, "CitiesDataBase" );
      return true;}
    if( !publicOnly && modelElementContains(x, y, 2480, 60) ) {
      panel.addInspect( 2480, 60, this, "RegionsSequence" );
      return true;}
    if( !publicOnly && modelElementContains(x, y, 2490, 110) ) {
      panel.addInspect( 2490, 110, this, "InstCityNames" );
      return true;}
    if( !publicOnly && modelElementContains(x, y, 2220, 270) ) {
      panel.addInspect( 2220, 270, this, "newEvent" );
      return true;}
    if( !publicOnly && modelElementContains(x, y, 50, -130) ) {
       panel.addInspect( 50, -130, this, "negotiationEnvCont" );
      return true; }
    if ( modelElementContains(x, y, 60, -320) ) {
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject(60,-320, this, "generalManager");
      } else {
        panel.addInspect( 60, -320, this, "generalManager" );
      }
      return true;}
    if ( !regionManagerAgents.isEmpty() && modelElementContains(x, y, 190, -
320) ) {
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 190, -320, this, "regionManagerAgents"
);
      } else {
        panel.addInspect( 190, -320, this, "regionManagerAgents" );
      }return true;}
    if ( modelElementContains(x, y, 60, -260) ) {
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 60, -260, this, "vehicle" );
      } else {
        panel.addInspect( 60, -260, this, "vehicle" );
      }
      return true;
    }
    if ( !localVehicle.isEmpty() && modelElementContains(x, y, 190, -270) )
{
      if ( clickCount == 2 ) {
        panel.browseEmbeddedObject( 190, -270, this, "localVehicle");
      } else {
        panel.addInspect( 190, -270, this, "localVehicle" );
      }
      return true;
    }return false;}
  // Environments
public final Environment negotiationEnvCont = new Environment( this );
  /**
   * Constructor
   */
  public Main( Engine engine, ActiveObject owner, ActiveObjectCollection<?
extends Main> collection ) {
    super( engine, owner, collection );}
  @Override
  public void create() {
    // Creating embedded object instances
    generalManager = instantiate_generalManager_xjal();
    for ( int i = 0; i < 0 ; i++ ) {
      instantiate_regionManagerAgents_xjal( i );}
    vehicle = instantiate_vehicle_xjal();
    for ( int i = 0; i < 0 ; i++ ) {
      instantiate_localVehicle_xjal( i );}
```

```
    // Assigning initial values for plain variables
    infCity = "Initial number of city: " ;
    minX = 10000000 ;
    minY = 10000000 ;
    totNumbOfReg = 0 ;
    numbOfCity = 0 ;
    homeCity = 1 ;
    startToCreateEvent = false ;
    numbOfRegOptString = "" ;
    initialSolStrategyGMAString = "" ;
    // Dynamic initialization of persistent elements
    presentation = new ShapeGroup( Main.this, true, 0, 0, 0,
determineRegions, initNumbOfCity, lengthOfEnvironment, widthOfEnvironment,
numbOfReg, convertCord, createAgentsButton, numbOfRegOpt, avgDistOpt,
regDistOpt, regionDetAlg, homeCityOpt, homeCityVal, setInitialParameters,
searchAlgorithms, searchAlgP, maxNumbOfIter, maxNumbOfNonImp, numbOfBestCan,
neighOpt, regCenterOpt, avgDistInput, numbOfNeg, initialSolStrategyGMA,
problemType, numbOfRow, numbOfCol, optStrategies, changeReportFileLocation,
fileLocation, avoidSameMembers, maxNumbOfIterSGS, maxNumbOfNonImpSGS,
dispLOfEnv, dispWOfEnv, dispNumbOfCityR, dispNumbOfReg,
regionManagerAgents_Presentation, vehicle_presentation, cityP, xCordCity,
xCordLabel, yCordLabel, yCordCity, dispSearchAlgP, dispMaxNOfIter,
dispMaxNOfNonImp, dispMaxNOfBestCanSol, dispMultConst, dispAvgDist,
dispNumbOfNeg, dispNumbOfRow, dispNumbOfCol, dispAgentNumber, dispRegName,
dispMaxNumbOfIterSGS, dispMaxNumbOfNonImpSGS, localVehicle_presentation,
regCenterP );
    icon = new ShapeGroup( Main.this, true, 0, 0, 0 );
    // Creating contents for replicated shapes
    cityP.createShapes();
    xCordCity.createShapes();
    xCordLabel.createShapes();
    yCordLabel.createShapes();
    yCordCity.createShapes();
    dispAgentNumber.createShapes();
    dispRegName.createShapes();
    regCenterP.createShapes();
    // Environments setup
    negotiationEnvCont.disableSteps();
    negotiationEnvCont.setSpaceContinuous(
((int) widthOfEnvironment.getValue()) ,
((int) lengthOfEnvironment.getValue())  );
    negotiationEnvCont.setNetworkUserDefined();
    negotiationEnvCont.setLayoutType( Environment.LAYOUT_USER_DEFINED );
    // Creating non-replicated embedded objects
    setupParameters_generalManager_xjal( generalManager );
    create_generalManager_xjal( generalManager );
    setupParameters_vehicle_xjal( vehicle );
    create_vehicle_xjal( vehicle );
    // Port connectors with non-replicated objects
    // Creating replicated embedded objects
    for ( int i = 0; i < regionManagerAgents.size(); i++ ) {
       setupParameters_regionManagerAgents_xjal( regionManagerAgents.get(i),
i );
create_regionManagerAgents_xjal(regionManagerAgents.get(i),i)}
    for ( int i = 0; i < localVehicle.size(); i++ ) {
       setupParameters_localVehicle_xjal( localVehicle.get(i), i );
       create_localVehicle_xjal( localVehicle.get(i), i );}
    assignInitialConditions();
    initNumbOfCity.setValueToDefault();
    lengthOfEnvironment.setValueToDefault();
    widthOfEnvironment.setValueToDefault();
    numbOfReg.setValueToDefault();
    setInitialParameters.setValueToDefault();
    maxNumbOfIter.setValueToDefault();
    maxNumbOfNonImp.setValueToDefault();
    numbOfBestCan.setValueToDefault();
    avgDistInput.setValueToDefault();
    numbOfNeg.setValueToDefault();
```

```
    numbOfRow.setValueToDefault();
    numbOfCol.setValueToDefault();
    changeReportFileLocation.setValueToDefault();
    maxNumbOfIterSGS.setValueToDefault();
    maxNumbOfNonImpSGS.setValueToDefault();
    onCreate();}
@Override
  public void start() {
    newEvent.start();
    generalManager.start();
    for (ActiveObject embeddedObject : regionManagerAgents){
      embeddedObject.start();
      }vehicle.start();
    for (ActiveObject embeddedObject : localVehicle){
      embeddedObject.start();}
    onStartup();}
  public void onStartup() {
    super.onStartup();
determineRegions.setEnabled(false);
avgDistOpt.setVisible(false);
numbOfReg.setVisible(false);
regDistOpt.setVisible(false);
dispNumbOfReg.setVisible(false);
regionDetAlg.setVisible(false);
homeCityVal.setVisible(false);
searchAlgP.setVisible(false);
dispSearchAlgP.setVisible(false);
dispMaxNOfIter.setVisible(false);
maxNumbOfIter.setVisible(false);
maxNumbOfNonImp.setVisible(false);
dispMaxNOfNonImp.setVisible(false);
maxNumbOfNonImpSGS.setVisible(false);
dispMaxNumbOfNonImpSGS.setVisible(false);
maxNumbOfIterSGS.setVisible(false);
dispMaxNumbOfIterSGS.setVisible(false);
numbOfBestCan.setVisible(false);
dispMaxNOfBestCanSol.setVisible(false);
neighOpt.setVisible(false);
dispMultConst.setVisible(false);
regCenterOpt.setVisible(false);
avgDistInput.setVisible(false);
dispAvgDist.setVisible(false);
dispNumbOfNeg.setVisible(false);
avoidSameMembers.setVisible(false);
numbOfNeg.setVisible(false);
dispNumbOfRow.setVisible(false);
dispNumbOfCol.setVisible(false);
numbOfRow.setVisible(false);
numbOfCol.setVisible(false);
optStrategies.setVisible(false);
fileLocation.setVisible(false);
searchAlgorithms.setVisible(false); }
  public List<Object> getEmbeddedObjects() {
    LinkedList<Object> list = new LinkedList<Object>();
    list.add( generalManager );
    list.add( regionManagerAgents );
    list.add( vehicle );
    list.add( localVehicle );
    return list;}
  // Reaction on changes -------------------------------------
  public void onChange() {
    newEvent.onChange();}
  public void onDestroy() {
    super.onDestroy();
    newEvent.onDestroy();
    negotiationEnvCont.onDestroy();
    generalManager.onDestroy();
    for (ActiveObject embeddedObject : regionManagerAgents) {
```

```
      embeddedObject.onDestroy()}
    vehicle.onDestroy();
    for (ActiveObject embeddedObject : localVehicle) {
      embeddedObject.onDestroy();
    }}
  // Additional class code
Random rndm = new Random(2);
Print print=new Print(true);
SetPresentations setPresent= new SetPresentations();
NecessaryCalculations necessaryCalculations = new
NecessaryCalculations(rangeX,rangeY,numbOfCity);
  // End of additional class code
  /**
   * This number is here for model snapshot storing purpose
   */
  private static final long serialVersionUID = 1319105663276L;
  private void writeObject(java.io.ObjectOutputStream _stream) throws
java.io.IOException {
    _stream.defaultWriteObject();
    _stream.writeObject( generalManager.getConnections() );
    _stream.writeObject( vehicle.getConnections() );
      writeCustomData( _stream );}
  @SuppressWarnings("unchecked")
  private void readObject(java.io.ObjectInputStream _stream) throws
java.io.IOException, ClassNotFoundException {
    _stream.defaultReadObject();
    generalManager.restoreOwner( this );
    regionManagerAgents.restoreOwner( this );
    vehicle.restoreOwner( this );
    localVehicle.restoreOwner( this );
    negotiationEnvCont.restoreOwner( this );
    newEvent.restoreOwner( this );
    createInitialCities.restoreOwner( this );
    inputFromUser.restoreOwner( this );
    cityArea.restoreOwner( this );
    agentsObjectsView.restoreOwner( this );
    generalManager.restoreConnections_xjal( (LinkedList<Agent>)
_stream.readObject() );
    vehicle.restoreConnections_xjal( (LinkedList<Agent>)
_stream.readObject() );
      finishReadObject_xjal( _stream, Main.class ); }}
```

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: *Unutmaz Durmuşoğlu, Zeynep Didem*
Nationality: *Turkish (TC)*
Date and Place of Birth: *13 January 1984, Malatya*
Marital Status: *Married*
Phone: +90 342 3172614
Fax: +90 342 3604383
email: unutmaz@gantep.edu.tr

EDUCATION

Degree	Institution
MS	University of Gaziantep
BS	University of Gaziantep
High School	Adıyaman Anatolian High School

WORK EXPERIENCE

Year	Place	Enrollment
2007- Present	University of Gaziantep	Research Assistant

FOREIGN LANGUAGES

ENGLISH (Advanced)
GERMAN (Basic)

PUBLICATIONS

In International Journals:

1. Baykasoglu, A., **Durmuşoğlu, Z.** (2012)**.** A classification scheme for agent based approaches to dynamic optimization. *Artificial Intelligence Review*. 1-26. doi:10.1007/s10462-011-9307-x (Article in Press)

2. Baykasoglu, A., **Durmuşoğlu, Z.D.U**. (2012). Flow time analyses of a simulated flexible job shop by considering jockeying, International Journal of Advanced Manufacturing Technology, 58(5-8), 693-707.

3. Dereli, T., Baykasoğlu, A., Durmuşoğlu, A., **Durmuşoğlu, Z. D. U. (2011)**. Enhancing technology clustering through heuristics by using patent counts, *Expert Systems with Applications.* 38(12), 15383-15391.

4. Baykasoğlu, A., **Durmuşoğlu, Z. D. U.**, Kaplanoğlu, V. (2011). Training fuzzy cognitive maps via extended great deluge algorithm with applications. *Computers in Industry*. 62(2), 187-195.

5. Baykasoğlu, A., Göçken, M., **Unutmaz, Z. D.** (2008). New approaches to due date assignment in job shops, European Journal of Operational Research, 187, 31-45.

6. Baykasoğlu, A., Göçken, M., **Unutmaz, Z. D.** (2008). Due date assignment using ADRES and simulated annealing, International Journal of Industrial and Systems Engineering, 3(3), 277-297.

In International Conferences:

1. Baykasoğlu, A., **Durmuşoğlu, Z.D.U.** (2011). Dynamic optimization in a dynamic and unpredictable world, 2011 Proceedings of PICMET '11: Technology Management In The Energy-Smart World (PICMET), July 31-August 4, Hilton Portland and Executive Tower, Portland, Oregon, USA, pp. 2312-2319.

2. Baykasoğlu, A., **Durmuşoğlu, Z.D.U.,** Görkemli, L. (2011). Solving vehicle deployment planning problem by using agent based simulation modeling, 2nd International Symposium on Computing in Science & Engineering, June, 1-4, Gediz University Publications, editor: M. Güneş, ISBN:978-605-61394-2-0, 338-340, Kuşadası, Aydin, Turkey.

3. Baykasoğlu, A., Kaplanoğlu, V., **Durmuşoğlu, Z.D.U**., Şahin, C. (2011). A fuzzy TOPSIS approach to truck selection, FUZZYSS'11: 2nd International Fuzzy Systems Symposium, (ed., Gokceoglu C., Aladag HC., Akgun A) Hacettepe University, Cultural Center, November 17-18, Ankara, Turkey, pp. 142-145.

4. Baykasoğlu, A., **Durmuşoğlu, Z. D. U**, Kaplanoğlu, V. (2009). Training fuzzy cognitive maps by extended great deluge algorithm, FUZZYSS'09: 1st International Fuzzy Systems Symposium, TOBB Economy and Technology University, October 1-2, Ankara, Turkey, 110-117.

5. Baykasoğlu, A., **Unutmaz, Z. D.**, Kaplanoğlu, V. (2007). A truck load consolidation approach for a logistics company, International Logistics & Supply Chain Congress, 8-9 November, İstanbul, Turkey, 606-613.

In National Conferences:

1. Baykasoğlu, A., **Durmuşoğlu, Z. D. U**., Görkemli, L. (2011). Etmen tabanlı benzetim: ANYLOGIC^{TM} yazılımı ve örnek bir uygulama, Endüstri Mühendisliği Yazılımları ve Uygulamaları Kongresi, İzmir, 30 Eylül-01/02 Ekim, TMMOB Makina Mühendisleri Odası Yayın No: E/2011/559, 197-204.

2. Baykasoğlu, A., **Durmuşoğlu, Z. D. U**. (2010). Özel İlköğretim Okulu seçimi için çok kriterli karar destek modeli, YA/EM 2010: Yöneylem Araştırması/Endüstri Mühendisliği Kongresi 30. Ulusal Kongresi, Sabancı Üniversitesi, 30 Haziran-2 Temmuz 2010, İstanbul, Bildiriler Özeti Kitabında, 101.

3. Baykasoğlu, A., **Durmuşoğlu, Z. D. U** (2009). Kuyruk Atlama Mekanizmalarının Esnek Atölyelerde Akis Suresine Etkisi, YA/EM 2009: Yöneylem Araştırması/Endüstri Mühendisliği Kongresi 29. Ulusal Kongresi, Bilkent Üniversitesi, 22-24 Haziran 2009, Ankara, Bildiriler Kitabı CD'sinde.

4. Baykasoğlu, A., **Unutmaz, Z. D.** (2008). Atölye tipi üretim sistemlerinde esnekliğin akis süresine etkisi, YA/EM'2008: Yöneylem Araştırması/ Endüstri Mühendisliği Kongresi XXVIII. Ulusal Kongresi, Galatasaray Üniversitesi, 30 Haziran-2 Temmuz 2008, İstanbul, Bildiriler Kitabı CD'sinde.

5. Baykasoğlu, A., Kaplanoğlu, V., **Unutmaz, Z.D**. (2008). An agent based framework for truck load consolidation, YA/EM'2008: Yöneylem Araştırması / Endüstri Mühendisliği Kongresi XXVIII. Ulusal Kongresi, Galatasaray Üniversitesi, 30 Haziran-2 Temmuz 2008, İstanbul, Bildiriler Kitabı CD'sinde.

6. Baykasoğlu, A., Göçken, M., **Unutmaz, Z. D**. (2007). Atölye tipi üretim sisteminde girdi kontrolü etkilerinin araştırılması, YAEM'07, Yöneylem Araştırması ve Endüstri Mühendisliği Kongresi, 02-04 Temmuz, İzmir.

7. Baykasoğlu, A., Kaplanoğlu, V., **Unutmaz, Z. D.** (2007). Lojistik firmalarında yük konsolidasyonu için bir model önerisi, YAEM'07, Yöneylem Araştırması ve Endüstri Mühendisliği Kongresi, 02-04 Temmuz, İzmir.

8. Baykasoğlu, A., Göçken, M., **Unutmaz, Z. D.** (2006), Atölye tipi üretimde akış süresi tahmini ve teslim tarihi belirlenmesi için yeni bir yaklaşım, GAP V. Mühendislik Kongresi, 26-28 Nisan, Şanlıurfa.

National Journal Papers:

1. Dereli, T., Baykasoğlu, A., Durmuşoğlu, A., **Unutmaz, Z. D**., Sönmez, A. (2007). Futbol endüstrisi: Seyirci Sayısı ve Takımların Forma Renkleri, Çağ Üniversitesi Sosyal Bilimler Dergisi, ISSN 1304-8392, 4(1), 13-25.

HOBBIES

She is interested in reading mathematics, puzzles, theater plays and she enjoys cooking.