DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

REALIZING METAHEURISTIC ALGORITHMS IN MULTI-AGENT BASED MODELLING ENVIRONMENTS

by

Mümin Emre ŞENOL

January, 2017

İZMİR

REALIZING METAHEURISTIC ALGORITHMS IN MULTI-AGENT BASED MODELLING ENVIRONMENTS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Industrial Engineering, Industrial Engineering Program

by

Mümin Emre ŞENOL

January, 2017
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"REALIZING METAHEURISTIC ALGORITHMS IN MULTI-AGENT BASED MODELLING ENVIRONMENTS"** completed by **MÜMİN EMRE ŞENOL** under supervision of **PROF. DR. ADİL BAYKASOĞLU** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Adil BAYKASOĞLU

Supervisor

[illegible]	[illegible]
Prof. Dr. Rızvan EROL	Doç. Dr. Serdar TAŞAN
Jury Member	Jury Member

[illegible]

Prof. Dr. Emine İlknur CÖCEN

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENT

Firstly, I would like to thank my supervisor, Prof. Dr. Adil BAYKASOĞLU for his support; advice and encouragement that helped me overcome challenges throughout the my M.Sc. thesis.

I would like to present thanks to Assoc. Prof. Dr. Vahit KAPLANOĞLU for his guidance, suggestions and contribution to this study.

I would like to thank, everyone who have helped me throughout the progress of my M.Sc. thesis, my colleagues and all staff member in department of Industrial Engineering. Especially I am grateful to my friends, Burcu KUBUR, B.Meltem KAYHAN, Nurhan DUDAKLI and F. Selen KARASLAN for their helpfulness, and encouragement. In addition, a special thanks to my friend H. Yağız MERSİN for making this period easier and funny.

Finally, I would like to express my deep thankfulness to my family, who have loved and supported me whole my life, Bahaattin ŞENOL, Fatma ŞENOL and twin sister Merve ŞENOL.

Mümin Emre ŞENOL

REALIZING METAHEURISTIC ALGORITHMS IN MULTI-AGENT BASED MODELLING ENVIRONMENTS

ABSTRACT

Derivative-based numerical methods are generally insufficient for solving difficult computational optimization problems. Therefore, most of the researchers devoted their research efforts towards developing metaheuristic algorithms for solving complex/difficult computational optimization problems. Researchers are usually tried to imitate some natural phenomenon while developing metaheuristic algorithms. Although many effective metaheuristic algorithms were evolved for problem solving, very few of them truthfully realize dynamic characteristics of the phenomenon. We believe that an agent based modeling/design environment will be more useful and natural way for a better realization of the inspired phenomenon. It is possible to model various behaviors such as entering and leaving of agents, group forming etc. Additionally, purposeful inter-agent communications for goal seeking can be achieved easily by making use of the power of agent based system modeling and algorithm platforms.

In this research we implemented a metaheuristic algorithm which is known as Stochastic Diffusion Search (SDS) as a truly dynamic optimization algorithm by using the multi-agent modelling and programming approach. In the proposed SDS algorithm, solution vectors, ie, agents can communicate with each other for determining search direction. They can decide to disappear and new agents can appear. Thus the population size is not fixed. In other words, the proposed algorithm has an inherent dynamic structure so it can be more easily adapted to dynamic optimization problems. The proposed algorithm is developed in JACK multi-agent development environment. Single machine scheduling problem is modelled and solved as a test case.

Keywords: Stochastic diffusion search algorithm, multi-agent based modeling, single machine scheduling problem.

METASEZGİSEL ALGORİTMALARIN ÇOKLU ETMEN BENZETİM ORTAMINDA GERÇEKLEŞTİRİLMESİ

ÖZ

Türeve dayalı nümerik yöntemler zor optimizasyon problemlerinin çözülmesinde genellikle yetersiz kalmaktadır. Bu nedenle, bir çok araştırmacı çalışmalarını kompleks ve zor optimizasyon problemlerin çözümü için metasezgisel algoritmalar geliştirmeye adamıştır. Araştırmacılar genellikle metasezgisel algoritmalarında doğa olaylarını taklit etmeye çalışmaktadırlar. Birçok etkili metasezgisel algoritma geliştirilse de çok azı taklit ettikleri doğal olayın dinamiklerini gerçekten realize etmektedir. İnanıyoruz ki, etmen bazlı modelleme ortamı ilham alınan doğa olayının gerçeklenmesinde daha kullanışlı ve doğal bir yoldur. Etmenlerin sisteme giriş çıkışı, gruplaşma gibi davranışları modelleme, maksatlı etmen iletişimleri gibi aktiviteler etmen tabanlı modelleme ve algoritma geliştirmenin kullanımıyla mümkün hale gelmektedir.

Bu çalışmada, dinamik bir metasezgisel optimizasyon algoritması olan Stokastik Yayılım Arama Agoritmasının(SYA) çoklu etmen modelleme ortamında gerçeklenmesi realize edilmeye çalışılmıştır. Önerilen SYA algoritmasında çözüm vektörleri(etmenler) kendi aralarında arama yönünün belirlenmesi konusunda iletişime geçebilmekte, etmenler sistemden çıkmaya karar verebilmekte, yeni etmenler sisteme girebilmektedir. Bir diğer deyişle, önerilen algoritma doğal bir dinamik yapıya sahip olduğundan, dinamik optimizasyon problemlerine kolayca adapte edilebilmektedir. Önerilen algoritma JACK çoklu-etmen modelleme ortamında geliştirilmiştir. Tek makine çizelgeleme problemi modellenmiş ve çözülmüştür.

Anahtar kelimeler: Stokastik yayılım arama algoritması, çoklu-etmen tabanlı benzetim, tek makine çizelgeleme problemi

CONTENTS

LIST OF FIGURES

Pages

LIST OF TABLES

Pages

CHAPTER ONE
INTRODUCTION

Nowadays, combinatorial problems appear in many applications of human’s life such as routing, packing, assignment, scheduling, cutting, transportation network design, protein alignment and other research areas of extreme economic, industrial and scientific importance (Sorensen and Glover,2013). Scientists use different methods to solve these combinatorial problems. We can categorize these methods into two groups namely, heuristic and exact methods. Exact algorithms are adequate for solving small size combinatorial problems and they guarantee optimal solution in a timely manner. On the other hand, if the problem size increases, run time of the algorithm also increases dramatically. In other words, exact algorithms can only solve small size instances practically. For this reason, researchers generally use heuristic algorithms for larger size instances. At that point, there is a trade of between reasonable run-time and optimality.

Dynamic programming, branch-and-bound (B&B), Langrangian relaxation based methods, and linear, integer programming based methods (branch-and-cut, branch-and-price, and branch-and-cut-and price) are some exact methods for combinatorial optimization problems (Laporte and Nobert, 1987).

Local search mechanisms and independently developed solutions are called as metaheuristic algorithms (Sorensen and Glover,2013). Metaheuristic algorithms can be classified into two groups as single solution metaheuristics and population based metaheuristics. Greedy randomized adaptive search procedure (GRASP), simulated annealing (SA), variable neighborhood search (VNS) are some examples of single solution metaheuristics. On the other hand, ant colony optimization (ACO), evolutionary algorithms (EAs), scatter search (SS) are some of the popular instances of population based metaheuristics (Gendreau and Potvin, 2005).

Many of the researchers try to mimic some natural phenomena in their mataheuristic algorithms which are independent from the group of metaheuristic algorithm. For example, researchers imitate food-searching mechanism in ant colony optimization. Also gene cloning, crossover operations between genes are imitated through evolutionary algorithms. Moreover, events in the nature are dynamic and their states changes in time. In addition, living organisms interact with each other in many of the natural events. They communicate and share information about their thoughts and beliefs. Additionally, they try to solve their problems by sharing information about their states.

It is impossible to imitate natural phenomena and communication among the organisms via classical algorithm procedures. In order to represent phenomena and interaction between organisms, dynamic methods such as multi-agent based modeling environments should be used instead of procedural programming approaches.

There exist many different definitions of an agent because of agent based systems is a very new area of research (Padgham and Winikoff, 2004). According to definition of Wooldridge and Jennings (1995), an agent is a computer system which is able act autonomously in some environment for reaching goals. The agents are used for meeting their aims. Agents have many capabilities such as being mobile, truthful and rational. They have also ability of learning (Woolridge, 2001). Moreover, agents are social; they can interact with agents and human-beings for meeting their goals. They are also reactive; they are aware of changes in their environment and respond these changes in a timely fashion. In addition, they are proactive; they are able to display initiative behavior to meet their purposes (Bellifemine et al., 2007).

There are some characteristics of agents that make them different from classical software approaches;

- The first characteristic of an agent is being autonomous. By this characteristic, agents have the ability of making self-decisions. They are capable of taking actions in order to satisfy their desires. This characteristic differs agents from objects of object-oriented programming.

When we consider a system including many agents, system tends to be decentralized because of the agents' autonomous characteristic (Padgham and Winikoff, 2004).

- The second characteristic of an agent is situatedness. According to this characteristic, agents adapted to challenging environments more easily. Through this characteristic, agents are suitable for dynamic, unpredictable and unreliable environments.

- The third characteristic of an agent is being reactive. Agents are usually took place in dynamic environments where changes are occurred very quickly. Agents must give reactions to these rapid changes in its environment simultaneously. In other words, agents have to be reactive for responding these changes in a timely manner (Padgham and Winikoff, 2004).

- Fourth characteristic of an agent is being proactive. An agent should continue to seek its goal even if its plan fails. It has to be robust for tolerating failures easily. Therefore, they must be flexible for being robust. Moreover, an agent should not only give reactions to its environmental changes; but also show behavior by taking initiative. It should make alternative plans for different situations.

- Lastly, an agent has to be social for communicating with other agents in its environment. In addition to send messages to other agents within the environment, an agent should engage all other social activities of human beings such as cooperation, coordination, negotiation etc. (Woolridge, 2001).

The selection of the programming environment where the imitation will be programmed is as important as choosing natural phenomena or human behavior that will be imitated. Programmers will obtain better results, when the structure of the environment and natural phenomena fits. For instance, Stochastic Diffusion Search (SDS) algorithm fits well with multi-agent based environment with its distributed structure (Bishop, 1989).In SDS; programmers are able to mimic gold searching process of a group of miners (Bishop, 1989). They divide mountain in different hills and distribute each miners to these hills. Moreover, all of the miners show some human behaviors such as communicating, negotiating, and cooperating. In addition to these, different strategies such as passive-recruitment mode, active-recruitment mode, dual-recruitment mode, context-sensitive mechanism and context-free mechanism can easily be modeled through SDS (Myatt et al., 2006). Through all of these strategies, it is possible to see different types of human behavior like getting other ideas, self-reasoning, manipulating people etc. By all of these features, SDS and multi-agent based environments fit well for realizing metaheuristic algorithms in multi-agent based modeling environments.

1.1 Aim of the Study

This thesis is conducted to realize a metaheuristic algorithm in a multi-agent based modeling environment. SDS is selected for implementation. To the best of our knowledge, this will be the first study in the literature which SDS algorithm is realized in a multi-agent based environment. Moreover, we can say that realization of other metaheuristics in multi-agent environments is very rare. An agent based modeling/design environment will be more beneficial and natural way for a better realization of the miners' movements, communication, and gold seeking process in SDS algorithm. It is possible to model various behaviors easily such as, going to hill, turning back to saloon, leaving the system, group forming of agents, manipulating other agents' ideas etc. by making use of the power of agent based system modeling and algorithm development. Through realization of SDS algorithm, dynamic nature of a human behavior can be mimicked truthfully. In addition, with the help of agents' being proactive and reactive, searching for the best solution never ends. Moreover, the proposed algorithm can be easily adapted to the dynamic optimization problems due to its inherent dynamic structure. Hence, both of the dynamic and classic combinatorial optimization problems can be designated easily by realizing the SDS algorithm in a multi-agent based environment. In addition, eclipse JAVA platform and JACK multi agent-based modeling environment can be connected and cooperated with each other for a possible implementation. Dynamic structure of JACK multi agent-based environment and the programming flexibility of eclipse JAVA platform can be combined and a hybrid way of implementation can be utilized. Finally, the proposed algorithm in this thesis is tested against single machine scheduling problems. According to computational experiments, results are very promising. Furthermore, our proposed system can be adapted for solving many other problems in the literature by the help of modular structure of the agent systems.

1.2 Framework of the Study

We can summarize the framework of the thesis as follow:

- A specific multi-agent model is generated through JACK multi agent-based environment.
- Procedural applications such as neighborhood mechanisms, crossover mechanisms, and system management are performed in eclipse JAVA platform.

There are four different kinds of the agents in the proposed models. These agents can be defined as:

1. Manager Agent
2. Position Update Agent
3. Trigger Agent
4. Solution Agent

Many plans and events are developed to connect these agents and model their interactions. Additionally, all agents are modeled for several different duties. Moreover, system is controlled and all of the agents and solution data, and problems instances are kept through eclipse JAVA platform.

The main steps of the SDS algorithm to model scenarios can be summarized as follows:

1. Initializing system
2. Updating miners positions
3. Triggering miners to interact with each other
4. Determining the agents' next move after communication with other agents
5. Deciding the agents' action for entering and leaving the system

Developed model is tested on a combinatorial optimization problem which is known as single machine scheduling problem. The problem is solved for minimizing total tardiness.

1.3 Outline of the Study

The study contains six chapters. The following chapter includes concept of multi agent-based modeling environments. In that chapter, detailed information about the properties of multi-agent based modeling environments and JACK are given.

Chapter three mainly focuses on SDS algorithm. In this chapter, detailed information about the natural phenomena of SDS algorithm is presented. In addition, chapter three includes some other information about the different strategies and specifications of agents' duties for each strategy. Moreover, procedural flow of the SDS algorithm is also given in this chapter.

Chapter four focuses on the implementation of the proposed model and realization of SDS algorithm in a multi-agent based modeling environment. The first part of the chapter four describes agent specifications. The second part is constituted by describing plans and events of each agent. The third part of that chapter focuses on the connection between JACK and eclipse JAVA platform and operations that are held in eclipse JAVA platform. Flow chart of the proposed model is also explained and several model designs are illustrated in the last part of that chapter.

In chapter five, the developed model is tested on the minimizing total tardiness in a single machine scheduling problem. Detailed information about the chosen test problems are also presented in that chapter. Moreover, the reason behind the selection of this combinatorial optimization problem is explained in that chapter. Experimental results and comparative results their ranks in the literature are also depicted by the help of figures.

Finally, in addition to summary of the thesis, the contribution of this thesis and future works are discussed in the conclusion chapter.

CHAPTER TWO
MULTI-AGENT BASED SYSTEMS

In recent years, all systems tend to be intelligent and people try to design intelligent system. Agent technology is one of the main approaches for developing intelligent systems. Multi-agent based systems are becoming more useful than the other approaches in terms of developing intelligent systems for solving complex problems.

A multi-agent system consists of organized intelligent agents that work or compete for the same goal that is very difficult to by the other single software components. The most appropriate way of solving a problem whose domain is particularly complex, large or unpredictable is to develop a multi-agent based system (Sycara, 1998).

There are many applications of multi-agent based systems in broad range of areas and disciplines. In economic sciences, Arthur et al. (1997) modeled agent behavior in stock markets. Axtell (2000) investigated transient states that are encountered along the way to equilibrium. These fields of interests affected other researchers in economic sciences and agent based computational systems spreaded among economists. Tesfatsion (2002) and Tesfatsion and Judd (2006) modeled agents situations in their researches for getting information about how the people make decisions in real studies.

Moreover, many scientists utilized agent-based models to analyze existing and hypothetical markets. Charania et al. (2006) modeled agents to guess possible futures for a market in space tourism. In their research, agents represent each stakeholder in the space industry. Through this research, tourism companies can seek to maximize profits when they compete with other companies for sales. Lopez Sanchez et al. (2005) proposed a multi agent based simulation for investigating market dynamics. Kuhn et al. (2010) develop a model to determine market share of a consumer airline.

Testa et al. (2012) developed a model for predator-prey link among other marine mammals and transient killer whales. In addition, Menges et al. (2008) employed agent-based modeling approach for establishing email-based social networks. Moreover, Aleman et al. (2009) used agent systems for modeling pandemic disease accounting for individual behavior and demographics. Malleson (2010) applied agent-based models for crime analysis. In fact, a virtual urban environment populated with virtual burglar agents was developed.

For better understanding on multi-agent based systems, it is vital to declare agents and their characteristics. Therefore, before the agent based dynamic optimization solution strategies detailed information about agents is given in this chapter. In addition, the proposed multi-agent based system coded and implemented in JACK^{TM} agent software. Moreover, properties of JACK Intelligent Agents^{TM} software are also mentioned in this chapter.

2.1 Agents

Despite of the several definitions for agents in the literature, researchers do not have concurrence of agent description. Wooldridge and Jennings (1995), defined the term agent as a computer system which has the ability of autonomous action in its environment to aim its pre-determined goals. According to the definition of Padgham and Winikoff (2004), an intelligent agent is a part of a computer program that has following properties.

- Situated – reside in an environment
- Autonomous – independent and cannot be controlled from outside
- Reactive –reacts changes in its environment in a timely manner
- Proactive – insistently chases aims
- Flexible – has many ways of acquiring its aims
- Robust – recoup its mistakes
- Social – negotiate with the other agents

All of the mentioned characteristics above can be described by different terms in multi-agent based design environment. Designed and built system should have these characteristics. For instance, for being proactive, an agent should have aims. Therefore, scope of the aim is a very crucial point for designing and constructing proactive agents.

The first property is defined by Padgham and Winikoff (2004) for an agent who is situated in an environment. While designing a multi-agent system, this property is associated with two terms as actions and percepts. According to Howden et al. (2001), interface among its environment and agent is called as percepts. Actions can be described as the events which are performed by the agents, for effecting its environment. In other words, percepts are signals that an agent receives from its environment and actions are agent’s reaction to these percepts. Relationship between percepts and actions are illustrated as in the Figure 2.1

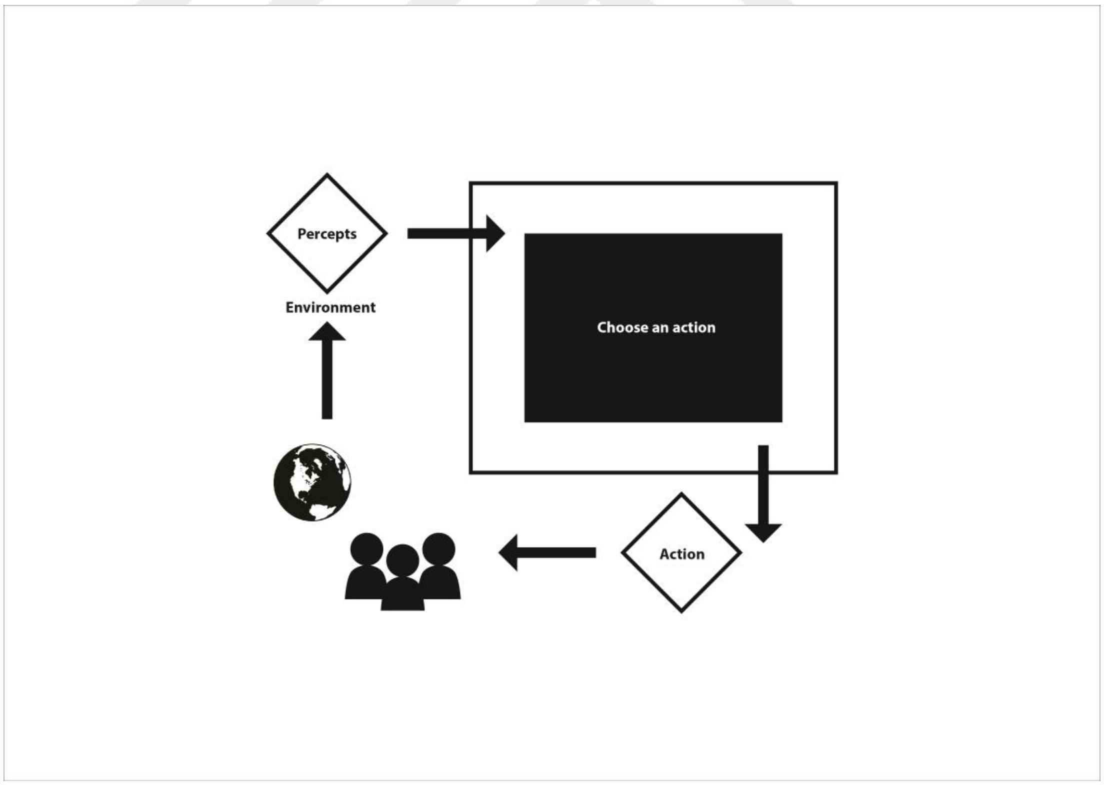


Figure 2.1 Relationships between percept and actions (Padgham and Winikoff, 2004)

The second and the third properties are being proactive and reactive. The term of goal is implied through agent's proactiveness. According to definition of Padgham and Winikoff (2004), a goal is something that the agent is working on or towards to it. It is very important to balance agent's proactive and reactive behavior. Besides, an agent should give reactions to changes in its environment while seeking its goal. At that point, the event is another essential term. Padgham and Winikoff (2004) defined the event as an important case which the agent must react. Relationship between agents' goals and events is depicted in Figure 2.2.

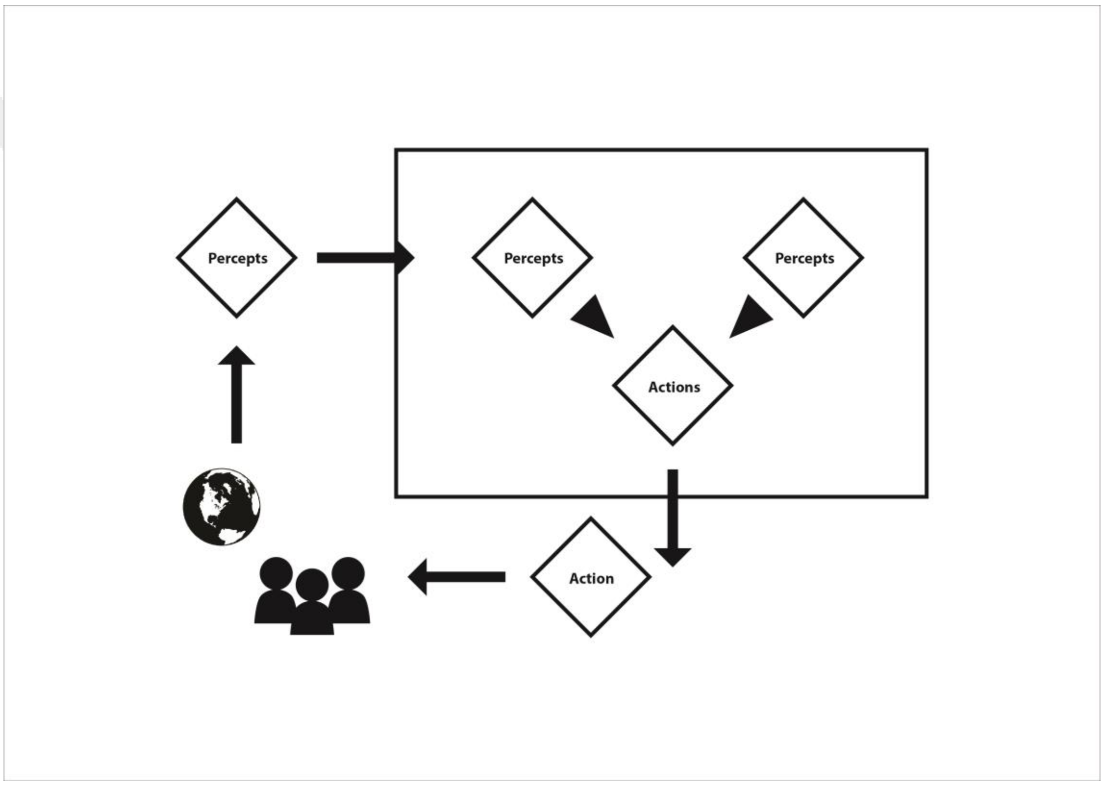


Figure 2.2 Relationships between goals and events (Padgham and Winikoff, 2004)

The fourth and fifth properties are being flexible and robust. These two properties are related to the terms plan and beliefs. Winikoff and Padgham (2004) defined the belief as some view of the agent's knowledge related with its environment, itself and other agents. On the other hand, plan is a method of reaching an aim. Goals can be achieved by acting different plans. An agent selects a plan to achieve a goal. Then, agent tries to apply the selected plan. If the plan fails, the agent can implement another plan. Through this mechanism, agents gain the characteristic of flexibility, since they may have variety of plans to achieve a goal. Moreover, they also gain the characteristic of robustness; goals are still achievable even failures of a plan. Relationship between plans and beliefs is demonstrated as in Figure 2.3.

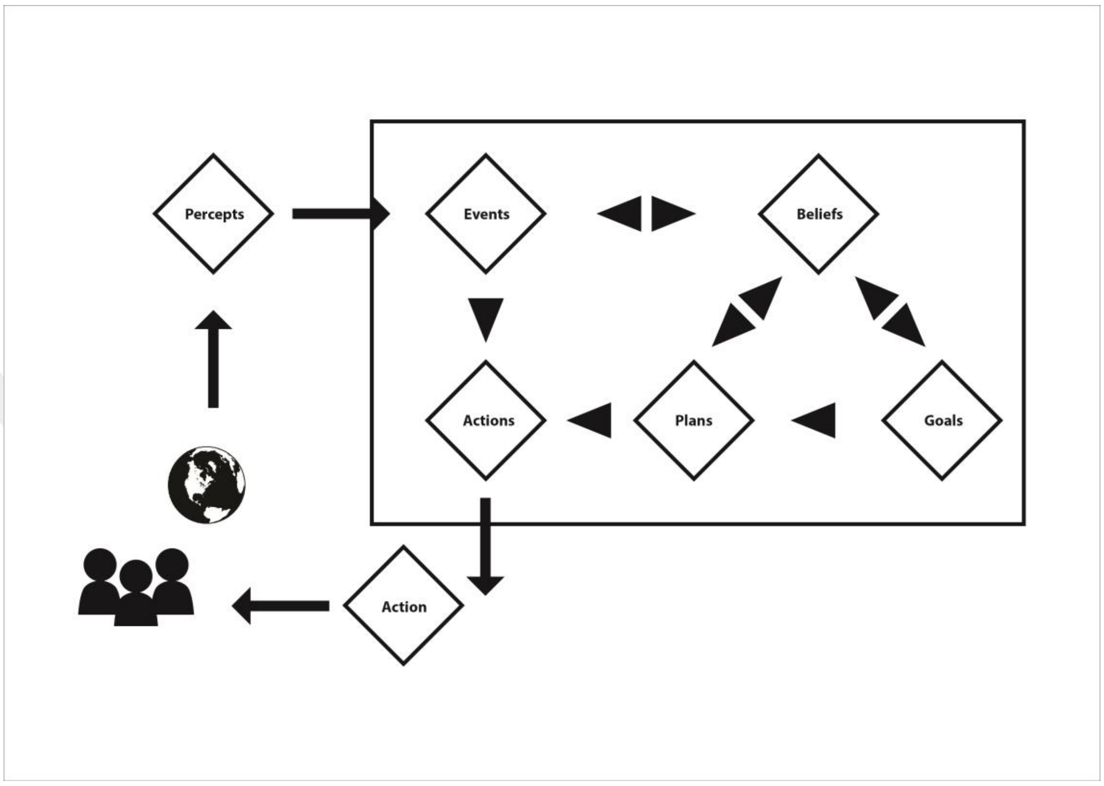


Figure 2.3 Relationships between plans and beliefs (Padgham and Winikoff, 2004)

The last but the most important characteristic of an agent is being social. By this characteristic, agents are able to interact with other agents. Agents can send messages to other agents via pre-determined protocols. Throughout this messaging mechanism, agents can work for the same goal.

Main concepts of an agent design procedure can be summarized as follows:

- Percepts – Related knowledge about the environment
- Events – Related knowledge about a change in the situation
- Goals – Aims to be achieved
- Beliefs – Knowledge related to the environment (unchanged)
- Plans – Expressed as accomplishing certain future world states.
- Actions – Ways of operations on the environment by agents

- Messages – Way of agents to negotiate
- Protocols – Descriptions of interaction rules

2.2 An Overview of JACK Intelligent Agents^{TM}

JACK Intelligent Agents^{TM} is evolved by Agent Oriented Software Pty. Ltd. (AOS) for multi-agent system modeling. Developers integrated JACK Intelligent Agents^{TM} with JAVA. Therefore, JAVA codes can be easily embedded into any place of JACK^{TM} Agent Language (JAL). Throughout the embedding process, the developers can gain the advantage of big flexibility. All of the JACK^{TM} source codes are compiled into regular JAVA code before their execution.

JACK Intelligent Agents^{TM} is one of the major multi agent-based development environments for building and deploying autonomous systems. Developers have the following design aims for JACK Intelligent Agents^{TM} (Howden et al., 2001);

- To ensure a robust, stable and light-weight product to programmers
- To meet many of practical application requirements
- Technology transfer between the theory and industrial applications
- To provide more applied studies

Moreover, JACK Intelligent Agents^{TM} is stand on the BDI paradigm (Beliefs/Desires/Intensions). It provides a mature implementation environment of the BDI paradigm to the developers. Throughout this implementation environment, developers have the chance of designing the model graphically. A general view of JACK Intelligent Agents^{TM} is demonstrated by the following Figure 2.4.

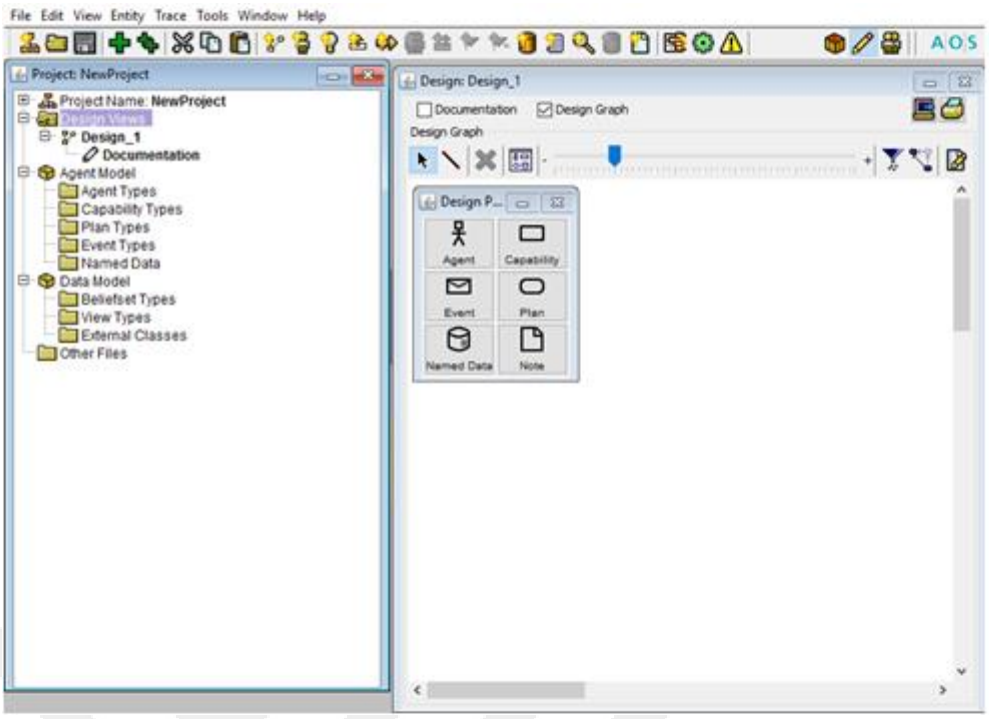


Figure 2.4 A general view of JACK Intelligent AgentsTM

In addition, JACK Intelligent Agents^{TM} incorporates three main extensions to JAVA. These extensions can be expressed as follows (Howden et al., 2001):

- To define agent's basic elements, some keywords (such as agent, plan and event);
- A group of statements to define attributes and other characteristics of the system elements
- A group of statements for the declaration of non-dynamic links
- A group of statements for changing the state of agents.

As mentioned previously the syntactic additions are converted into only JAVA statements and classes by making use of second extension to JAVA compiler. Then, it can be loaded with and named by another JAVA code. Moreover, the code of plans are transformed by the compiler for getting proper semantics of the BDI architecture.

Lastly, a group of classes which are named as kernel satisfies the needed run-time supporting the produced code. This also contains:

- Automatic management of concurrency between the tasks which are pursued in parallel;
- Set agent behavior related to events, failure of actions and tasks and so on;
- Native lightweight and highly performed negotiations infrastructure for multi-agent applications.

BDI paradigm is extended for dealing with inter-agent coordination by JACK Intelligent Agents^{TM}. In addition, new tools and developments that guide recent software engineering applications are also included. JACK Intelligent Agents^{TM} ‘s key constructs are tabulated in Table 2.1.

Table 2.1 JACK Intelligent Agents^{TM} ‘s key constructs (Howden et al., 2001)

Programming Construct	**Description**
Event	Events are the main factor of motivation for agents. Without events, the agents lose the ability of acting. Events are demonstrated after an internal stimulus. Events trigger operations for computational operations.
Plan	Plans are small computational operations that are the ways of agents' reactions to the different events. JACK calculates all of the possible alternative and applicable plan results and chooses one of them. Plans have a body that declared the stages to be computed in related to the event. Agent have different plans for reaching its aims.
Beliefset	Beliefsets are used to show the agent's indicative knowledge about its environment and itself.
Agent	Agents are computational pieces that has unique characteristic and states independently.

Table 2.1 JACK Intelligent Agents^{TM} ‘s key constructs (Howden et al., 2001)(continue)

Team	Teams are used to contain coordination views of different behavior of agents.

JACK Intelligent Agents^{TM} has several tool and components. JACK Intelligent Agents^{TM} is composed of the following modules (Howden et al., 2001):

- **JACK Runtime Environment:** The kernel that contains a communication layer with a simple agent calling service assists the execution of JACK.
- **JACK Compiler:** It is a tool that compiles codes into JAVA codes and reach JAVA compiler for producing executable programs.
- **BDI Agent Model:** In addition to the language syntax and the runtime kernel, BDI agent model adds assistance for BDI reasoning.
- **SimpleTeam Model:** This gives assistance for team-based reasoning, through additions of kernel and extensions of languages.
- **Agent Development Environment:** It is a Graphical User Interface (GUI) development environment for seeing and making changes on JACK applications.
- **Agent Debugging Environment:** It includes an agent interaction display for seeing messages among agents through the changes to the kernel by displaying internal execution states.
- **JACOB:** It is an effective object modeling toolkit to assist transportation of objects and application interactions in C++ and JAVA compilers. It provides flow of the objects in a user readable textual format, a fast binary format in XML.

2.3 Agent Based Solution Strategies for Dynamic Optimization Problems

There exist a lot of studies that is based on agent-based modeling approaches in the literature for dynamic optimization. It is not possible to cover all of these papers in this thesis, but some relevant papers some of them are also covered by Baykasoğlu and Durmuşoğlu (2014) are summarized in this section.

Pelta et al. (2009 a,b) proposed a multi-agent decentralized cooperative strategy (MAD-COS) for solving dynamic problems. In their study, cooperative agents try to find solutions by moving over a grid. Different communication mechanisms and methods for preserving explicit diversity are investigated and tested via moving peaks as a benchmark problem. In their research, they tried to find how different communication strategies affect the search space. Moreover, they proved that their proposed agent model is sufficient for the test problems by applying diversity strategies.

Wang and Liu (2010) solved dynamic travelling salesman problem by using an agent-based evolutionary search algorithm (AES). In their research, an agent refers for an alternative point in the search space. Agents are positioned in a lattice-like environment. They combined perturbation learning strategy and local updating procedure. They tested AES on dynamic version of KroA100TSP. The results show that AES has very outstanding findings on the tested problems. It surpasses standard genetic algorithm (SGA) under the performance measures of convergence duration and recording optimum points in dynamic environments.

Billiau and Ghose (2008) presented a novel approach for solving distributed constrained optimization problems (DCOPs). They suggested an algorithm named as Support Based Distributed Optimization (SBDO). SBDO differs from other DCOP algorithms by its using agent goals instead of weighted or soft constraints. They make a comparison on success of their proposed algorithm with asynchronous distributed optimization (ADOPT) and distributed pseudo tree optimization procedure (DPOP) with 120 meeting scheduling test problem. The results revealed that their proposed algorithm outperformed ADOPT and showed relatively faster convergence than DPOP.

Mahr et al. (2010) made a comparison about variety of planning methods, such as on-line optimization method and agent based solution approach for drayage operations in an uncertain environment. For comparing performance of different planning methods they dealt with dynamic vehicle routing problem associated with two kinds of uncertainty, i.e., service and arrival time. Results showed that agents' flexibility brought them additional benefits.

Voos (2009) studied on dynamic resource allocation problems. He proposed a multi agent based strategy that divides problem into single optimization problems. In that research, agents took the duty of local optimizing and canalized self-findings to a total proper solution. Test experimental results indicated that the proposed multi agent based strategy are applicable to deal with dynamic resource allocation problems.

Li and Li (2009) presented a method which was called as intercommunication job-sharing hybridization to deal with hard problems. The proposed method decomposed complex optimization problems into smaller sub-problems and through the cooperation and intercommunication of agents, these problems were solved as a piece of the complex problem. First, they designed a pre-model that named as Agent International, to improve decision-making process of international marketing. Preliminary results were promising.

Tang et al. (2004) proposed a dynamic optimization system. The presented model was used for solving automobile load makeup planning problem. System was composed of three kinds of agents. The first one is load agent, the second one is yard agents and the third one the truck scheduled agent. These three agents represented by different actors in the system. For example, yard agents represented the transferring yard and the truck-scheduled agent referred to the shipping firm. Lastly, load agent represented the truck during the planning state. They implemented three heuristic namely empirical method (EM), minimum spanning tree (MST) and vehicle routing optimization (VRO). They also combined MST and dynamic optimization system and evolved a new method called as MST Dyn. For testing the success of the developed algorithm, all algorithms have run on the same scenario for 120 days. The results displayed that EM produced the worst result and MST Dyn generated the most proper results.

Berro and Duthen (2001) proposed an approach for optimization in dynamic environments. In that approach, agents seek good solutions and give reactions changes in the state of the problem in a timely manner. The proposed approach tested on multimodal and multi-objective functions. The experimental findings were compared with Genetic algorithm based methods. The experimental results revealed that proposed method were competitive in terms of time and accuracy.

Jiang and Han (2008) proposed a simulated annealing (SA) based algorithm with multiple agents for solving real-time decision-making problem. Since there were no test instances before, a random generator was used for generating test problems. The success of the presented method was compared with variable elimination (VE) algorithm in terms of scalability and relative payoff. Results showed that SA was a more proper method to select actions selection in complex cooperative and autonomous systems.

Zhou et al. (2008) presented a method for simulating a real time job shop by combining multi agent systems (MAS) and discrete event systems. In the proposed method, agents are referred elements of a job shop. All agents had different duties and followed up these duties. The proposed system tested in terms of some performance criteria machine utilization, average and maximum number in queue, average time in system, average daily throughput and total time in queues, and maximal size of work in process. The results illustrated that the proposed system had the benefits of the its distributed structure.

Garcia et al. (2010) proposed a new centralized cooperative strategy based on taboo search to solve DOPs. In their study, there were many solvers for implementing different resolution algorithms for the examined problem. There was also a coordinator which was responsible for processing information. There were three main steps in the proposed strategy. First, every solver sent its information related to the performance to the coordinator, then coordinator processed this information and finally directives were sent to the solvers by the coordinator. Moving peaks benchmark problem and three mostly applied multimodal real test functions were used for testing the performance of proposed model. Performance of the proposed strategy evaluated according to offline error. The presented study outperformed other methods.

Lepagnot et al. (2010) proposed a novel approach based on agent’s exploration of search space for multi agent dynamic optimization (MADO). There were three main modules namely, agent manager, the coordinator, and memory module. The Number of agents changed in time but not exceed the predefined value. To determine the success of the considered model, moving peaks benchmark problem was chosen and offline error with standard deviation was chosen as success criteria. The results showed that MADO was a usable model for dynamic optimization problems.

Yan et al. (2010) presented an agent based evolutionary search (AES) approach. Agents represented the potential solutions. The researchers applied two diversity acquiring strategies namely random immigrants and adaptive dual mapping, for improving the success of AES for DOPs. Dynamic 0-1 optimization problems that were produced by XOR generator were selected as test problems. The researchers generated variety of test instances. The solution environment was integrally altered in each pre-specified iteration. Success of the AES was compared with SGA, primal dual GA, and the GA with random immigrant according to the best mean generation fitness. Obtained results revealed that AES had better performance than the other approaches. The researchers also stated that dynamic characteristic of the environment can influence the success of the presented method.

Hanna and Cagan (2009) proposed a new method called as evolutionary multi-agent system (EMAS) for adaptive optimization. Agents represented ways of generating solutions. Solution generation strategies were recombined, altered, and removed through genetic operations of genetic and evolutionary methods. EMAS differs from other approaches by its property of cooperation dimension. Cooperation was satisfied by employing all of the strategies in an autonomous agent and communication of agents. Travelling salesman problem was used for testing the success of the considered method. The experimental findings indicated that EMAS had better solutions than other approaches in terms of closeness to the optimal solution. The researchers stated that the reason behind the EMAS' better solutions was its ability of evolving the best team of agents dynamically. Moreover, they also indicated that utilizing EMAS in the presented way was leaded to lower duration while acquiring high qualified results.

Xian and Lee (2008) presented an agent-based dynamic scheduling method, which depends on ant colony intelligence (ACI) through local agent coordination. The aim of proposed research was to develop a dynamic manufacturing system by MAS. Agents were modeled via knowledge of their aims and functions. Through MAS, parallel execution of commands was provided. Agents also negotiated for enhancing system performance. Foraging and division of labor of an ant colony inspired the agent coordination mechanism. There were mainly five types of agents namely order agents, job agents, shop floor agents, work center agents, and machine agents. They stated that, their study differs from other researches in terms of a making realistic assumptions and producing more general manufacturing. For solving task scheduling problems and task allocation problems in a separate way, ACI was combined job and machine agents together. Method based on ACI (MAS+ACI) and FIFO dispatching rule (MAS+FIFO) were compared according to success criteria namely, average flow time, average tardiness, throughput, buffer size, and machine utilization. Results revealed that MAS+ACI perform well than MAS+FIFO.

Wang and Usher (2002) proposed an agent-based approach that uses contract-net protocol for negotiation of agents. Machine cell agents and job agents were used within a hierarchical control environment. They tested the presented model on a job shop scheduling problem with variety of simulation levels in terms of mean flow and queue time. The experiments showed that when the system was light, collaboration element did not have significant influence on the performance. Hence, in the heavily loaded system, collaborative factor provide dramatic decreases in the mean flow time. Moreover, when they evaluated average queue time when the system was heavily loaded, it is stated that negotiation mechanism prevented the high WIP levels of the bottleneck machine. According to experimental findings, it is stated that collaboration element had positive effect on the success of the contract net-based negotiation scheme for agent-based scheduling problems.

Wang et al. (2008) presented a multi-agent based method for scheduling problems in dynamic environments. The considered approach is cooperated with filtered beam search (FBS). In order to provide dynamism in the system, new jobs arrivals were generated. The system consisted of two kinds of agents namely system optimal agent (SOA) and cell coordinated agents (CCAs) and five different modules called as cooperation and coordination, human interface, execution and monitoring, human interface, and FBS for the decision-making process. The proposed approach was compared with two different dispatching rules according to the weighted quadratic tardiness values. The presented method outperformed dispatching rules. According to results, researchers stated that the presented system was applicable for real life problems.

Sghir et al. (2015) presented an agent based optimization method for solving quadratic assignment problem. Their proposed model includes different types of cooperated agents namely “crossover agent”, “local search agents” and “perturbation agents”. In their algorithm, they evolved a system which based on reinforcement learning. According to the test results, the presented model performed well on the tested benchmark instances in terms of solution quality.

Barenji et al. (2016) developed a multi-agent based dynamic scheduling system for manufacturing flow lines. Their proposed system is evolved for rescheduling manufacturing lines by considering dynamic customer demands and internal disturbances. The model is applicable for dynamic and static manufacturing lines. According to the simulation results, the presented model could increase the production rate and uptime productivity of flexible flow line manufacturing systems.

Erol et al. (2012) proposed a multi-agent based model for simultaneous and dynamic scheduling of machines and AGVs in manufacturing systems. In their study, AGVs and machines are scheduled simultaneously first time in the literature. Moreover, the proposed model is tested against dispatching rules. The results indicated that the proposed MAS model outperformed all of the classical dispatching rules.

Baykasoğlu and Kaplanoğlu (2011) presented a multi-agent based system for load consolidation. In their proposed model, agents are responsible for load consolidation decisions for the less-than-truckload orders. Negotiation mechanism is used for assigning the less-than-truckload orders to the trucks. In addition, the developed model includes load acceptance/rejection, load assignment, reassignment, routing and scheduling decisions.

Baykasoğlu et al. (2011) introduced a multi-agent based model for load consolidation problems of third-party logistic companies. Their proposed system composed of three different types of agents namely “order agent”, “truck agent” and “regional load consolidation agent”. They modeled the system against different scenarios. Agents make variety of consolidation decisions in order to meet customer demands in these scenarios.

Kaplanoğlu et al. (2015) proposed a multi-agent based scheduling approach that considers AGV breakdowns for AGVs and machines within a manufacturing system. The presented model was developed for solving problems in a manufacturing system without stopping the manufacturing process. In their model, real-time problems were solved throughout bidding and negotiating mechanisms between agents.

Şahin et al. (2015) proposed a multi-agent based system for simultaneously scheduling of flexible machine groups and material handling system working under a dynamic manufacturing environment. They designed the presented model by Prometheus methodology and programmed it in JACK. In their model, they used negotiation mechanism for problem solving. Each agent in the system had the ability of negotiating. The proposed model tested on dynamic and static environments. The results indicated that proposed model is effective on both environments.

Baykasoğlu and Görkemli (2015) presented an agent-based dynamic part family formation for cellular manufacturing applications. In their study, they evolved a novel agent-based clustering algorithm that deals with dynamic demand changes. Their presented model composed of three cooperated agents namely “part”, “manager” and “part family”. Since there were no data for dynamic benchmark data for part family problems in the literature, they tested their proposed algorithm on static problems. Despite the proposed model was not an optimization algorithm, the results showed that the algorithm has promising results.

Baykasoğlu and Görkemli (2016) developed a new dynamic virtual manufacturing approach through agent-based modeling. The presented method could able to carry out part family formation, virtual cell formation and scheduling simultaneously. The experimental results indicated that the proposed approach is able to manage dynamic part demand arrivals besides providing promising solutions.

CHAPTER THREE

STOCHASTIC DIFFUSION SEARCH (SDS) ALGORITHM

In this chapter, detailed information related to SDS algorithm is presented. Firstly, origins of SDS algorithm are given. Then, the natural phenomenon which this metaheuristic algorithm inspires is mentioned. In addition, this chapter includes information about different strategies and specifications of agents' duties for each strategy of SDS algorithm. Moreover, procedural flow of the SDS algorithm is also displayed in this chapter.

3.1 Origins of Stochastic Diffusion Algorithm

Origins of SDS depend on two methods for invariant pattern recognition, pattern identification within a larger data structure. The first method is a sequential algorithm namely Template Matching and the second one is a connectionist model called Hinton Mapping. SDS was introduced in and subsequently applied to a variety of real-world problems: locating eyes in images of human faces (Bishop et al, 1992); lip tracking in video films (Grech-Cini, 1995); self-localization of an autonomous wheelchair (Beattie and Bishop, 1998) and site selection for wireless networks (Hurley and Whitakar, 2002) . Furthermore, a neural network model of SDS using Spiking Neurons has been proposed (Nasuta et al, 1999).

3.1.1 Template Matching

Template Matching is most widely used in the context of 2D image matching. A template image is available and needs to be identified within a larger input image. The correlation among the transformed template and the corresponding region of the input image is computed. The solution is specified with the transformation which yields the highest correlation.

3.1.2 Hinton Mapping

A Hinton Mapping network contains two sets of detectors. These are canonical and retinocentric, respectively. In these methods, retinocentric feature patterns are mapped into canonical feature patterns.

3.2 Stochastic Diffusion Search

Stochastic Diffusion Search algorithm is first proposed in 1989 and can be seen as the member of swarm intelligence algorithms (Bishop, 1989). Opposed to other nature-inspired algorithms, it has a powerful mathematical structure defining its behavior and convergence. Moreover, it is a multi-agent optimization and global search algorithm based on some iterative actions among the agents. Each agent has a hypothesis about a feasible solution and tests its hypothesis individually. Successful agents implement this hypothesis testing procedure repeatedly while canalizing the unsuccessful agents by communicating directly. Throughout this mechanism, fast convergence of agents to the valuable solutions in the solution space can be ensured.

3.2.1 Mining Game

In order to simplify the explanation of SDS algorithm, Al-Rifaie and Bishop (2010) introduced a metaphor which is called “Mining Game” (Al-Rifaie and Bishop, 2010). A group of miners desire to obtain the information for finding gold on the hills of a mountain range but they do not have knowledge about its distribution. They divide the mountain into discrete hills and each hill contains different amount of mine. Miners try to dig the best seam that has the maximum amount of gold. At hand, they do not know the best seam. For solving this problem they apply SDS. The steps of SDS procedure can be shown as in Table 3.1.

Table 3.1 The steps of SDS procedure (Al-Rifaie and Bishop, 2010)

- In the beginning, all miners are allocated a hill to mine randomly.
- In each day every miner is allocated to selected seam on his hill to mine randomly.
- At the end of each day, the probability of a miner's happiness is related with the amount of gold he finds in that day.
- At the end of each day, all miners get together and unhappy miners select one of the other miners randomly to negotiate. If the negotiated miner is happy, he shares the information of hill that he dig. On the other hand, if the selected miner is unhappy, miner does not share information to other miner and the original miner selects randomly a hill for the following day.

3.3 Stochastic Diffusion Search Architecture

SDS algorithm includes three stages. Initial stage is initialization phase. In this phase, all of the agents get their first solutions. After this phase, iterative steps will begin. There are two new phases in these iterative steps. One of them is test phase and the other one is diffusion phase. In the test phase, agents are evaluated through their fitness function values. According to their fitness values, their states are determined (active or passive). After the test phase, diffusion phase will start. In diffusion phase, agents are recruited for matching other agents for communication. Throughout the different recruitment strategies, agents diffuse their solutions to other agents. The basic SDS architecture is demonstrated in Table 3.2.

Table 3.2 The basic SDS architecture

```
Initialize (Agents);
Repeat
Test (Agents);
Diffuse (Agents);
Until (Terminating Criteria)
```

3.4 Recruitment Strategies in Stochastic Diffusion Search

Different recruitment strategies can be applied in diffusion stage of SDS algorithm (Myatt et al., 2006). An agent can be in the form of two states. The first state as named as active state, If agent achieve success in the test stage, it will be in the form of active state. On the other hand, if agent fails in the test stage, it will be the form of inactive. It is engaged if it is communicated with an agent. Five different recruitment strategies namely "passive recruitment mode", "active recruitment mode", "dual recruitment mode", "context sensitive mechanism", "context free mechanism" are applicable in diffusion phase (Myatt et al., 2006).

3.4.1 Passive recruitment mode

The basic SDS algorithm (Bishop,1989) uses passive recruitment mode. In this mode, if the agent is inactive, other agent is randomly chosen and if randomly chosen agent is active, passive agent get other agent's hypothesis and changes its state as active. Besides, if the chosen agent is passive, the agent produces new hypothesis (Myatt et al., 2006)

Table 3.3 Passive recruitment mode

```
For agent = 1 to No_of_agents
        If ( agent.activity()==passive )
           random_agent = choose a random agent()
             If ( random_agent.activity()==active )
             agent.setHypothesis( random_agent.getHypothesis() )
             Else
             agent.setHypothesis( randomHypothesis() )
             End If/Else
        End If
End For
```

3.4.2 Active recruitment mode

In active recruitment mode, active agents are responsible for negotiation. They are on duty in diffusion phase. An active agent randomly chooses an agent if the chosen agent is inactive and unengaged with other active agent, then there will be information flow through active agent to inactive one and inactive agent is flagged as engaged. All active agents repeat the same process. If there is still unengaged inactive agent, these agents will produce new hypothesis (Myatt et al., 2006).

Table 3.4 Active recruitment mode

```
For agent = 1 to No_of_agents
 If ( agent.activity()==active )
  random_agent = choose a random agent()
   If (random_agent.activity()==passive && random_agent.getEngaged()==false)
           random_agent.setHypothesis(agent.getHypothesis())
           random_agent.setEngaged(true)
   End If
  End If
End For
```

3.4.3 Dual recruitment mode

In dual recruitment mode, both active and passive agents choose another agent randomly. If the chosen agent is inactive and unengaged, the hypothesis of active agent is passed through inactive one. Besides, if the agent is not active, it also chooses other agent randomly, if the chosen agent is active, agents will negotiate and active agent shares its knowledge with the inactive agent and inactive agent is flagged as engaged. If there is still unengaged and inactive agent, these agents choose randomly new hypothesis (Myatt et al., 2006).

Table 3.5 Dual recruitment mode

```
For agent = 1 to No_of_agents
  If ( agent.activity()==active )
    random_agent = choose a random agent()
    If(random_agent.activity()==passive && random_agent.getEngaged()==false)
           random_agent.setHypothesis(agent.getHypothesis())
           random_agent.setEngaged(true)

      End If
Else
      random_agent= choose a random agent()

        If (random_agent==active && agent.getEngaged()==false)

          agent.setHypothesis(random_agent.getHypothesis())

          agent.setEngaged(true)
```

Table 3.5 Dual recruitment mode (continue)

```
        End If
 End If/Else
End For
For **agent**=1 to No_of_agents
  If (**agent.activity==passive &&  agent.getEngaged()==false**)
     **agent**.setHypothesis(randomHypothesis)
  End If
End For
```

3.4.4 Context Sensitive Mechanism

In active recruitment mode, robustness and greediness of the algorithm will decrease. Fortunately, in dual recruitment mode, these two attributes increases. Hence, greediness of the dual recruitment mode decreases the robustness of the algorithm. In order to prevent this reduction, context sensitive mechanism can be useful. In other words, context sensitive mechanism provides the global search exploration. In this mechanism, if an active agent chooses an agent randomly. If the chosen agent is also active and has the same hypothesis with the active agent, this agent changes its state to inactive and chooses another hypothesis randomly (Myatt et al., 2006).

Table 3.6 Context sensitive mechanism

```
For agent = 1 to No_of_agents
  If ( agent.activity() )
   random_agent = choose a random agent ()
      If (random_agent.activity()==active
         If (agent.getHypothesis()==random_agent.getHypothesis())
            agent.setActivity ( false )
            agent.setHypothesis ( randomHypothesis() )
         End If
      End If
  End If
End For
```

3.4.5 Context Free Mechanism

The context free mechanism has almost the same structure with the context sensitive mechanism. Differently from the context sensitive mechanism, if the chosen agent is active; choosing agent changes its state to inactive and chooses another hypothesis randomly regardless of selected agent's hypothesis (Myatt et al., 2006).

Table 3.7 Context free mechanism

```
For agent = 1 to No_of_agents
If ( agent.activity() )
   random_agent = choose a random agent ()
      If (random_agent.activity()==active
            agent.setActivity ( false )
            agent.setHypothesis ( randomHypothesis() )
      End If
  End If
End For
```

CHAPTER FOUR
REALIZING STOCHASTIC DIFFUSION SEARCH ALGORITHM IN JACK MULTI-AGENT ENVIRONMENT

In this chapter, the design of the proposed multi-agent based SDS algorithm is performed. This chapter of the thesis explains agent types, their interactions, plans, events and how these components of the multi agent systems are implemented into JACK platform.

4.1 Agent Types and Agent Specifications

The proposed system is composed of four main types of agents namely “Manager Agent”, “Position Update Agent”, “Trigger Agent” and “Solution Agent”. Each of the agent types has special duties and is appointed for different plans and events.

4.1.1 Manager Agent

Manager agent is responsible for initializing the system, keeping all of the solution agent data and displaying solution agents’ solutions. Manager agent’s descriptors are summarized in Table 4.1.

Table 4.1 Manager agent’s descriptors

Name: Manager agent
Description: Initializes the simulation process, keeps all the solution agent data.
Lifetime: Initialized when a run command reaches to the system from JAVA. Demised when simulation finishes.
Initialization: Obtains the run command from JAVA.
Demise: Finishes all of the simulation process.
Functionalities included: System initialization functionality, keeping data functionality.
Uses data: Solution agent data

Table 4.1 Manager agent's descriptors (continue)

Produces data: Solution agents' solutions report data
Goals: Initialize system, display results of the simulation.
Percepts responded to: Arrival of a new run command to the system.
Actions: Change in the persistent database of solution agents.
Protocols and interactions: Start request with solution agents.

The simulation process will begin with the manager agent's start request to the solution agents. This method is illustrated in Figure 4.1.

```
public agent ManagerAgent extends Agent {
   #posts event StartRequest startRequestET;
   #handles event StartRequest;
   #sends event Start ev;
   #posts event DisplayRequest displayRequestET;
   #handles event DisplayRequest;
   #uses plan SendStartRequest;
   #uses plan Display;

   public ManagerAgent(String name)
   {
      super(name);
   }


   public void submitStartRequest(){
         String solutionAgentName=null;

         for(int i=0;i<Main.vector.size();i++){

            solutionAgentName=Main.returnSolutionAgents().get(i).toString();
            postEvent(startRequestET.request(solutionAgentName,1));

      }
      postEvent(displayRequestET.request());
   }

}
```
Figure 4.1 Manager agent start request

4.1.2 Position Update Agent

Position update agent is responsible for determining and changing solution agents' positions. A solution agent may take place in saloon where it is ready to negotiate and may be in hill for searching new solutions. Position update agent calls each solution agent to the saloon, in order to negotiate with the other solution agents. It also sends match agents command to the trigger agent. Its descriptors are summarized in Table 4.2 and its update request method is depicted in Figure 4.2.

Table 4.2 Position update agent descriptors

Name: Position update agent
Description: Changes solution agents' states.
Lifetime: Initialized when a run command reaches to the system from JAVA. Demised when simulation finishes.
Initialization: Obtains run command from JAVA.
Demise: Finishes all the simulation process.
Functionalities included: Changing solution agents' states functionality.
Uses data: Solution agent name data
Produces data: Solution agents' state data
Goals: Call solution agents back to the saloon.
Percepts responded to: Arrival of a new run command to the system.
Actions: Change states of solution agents.
Protocols and interactions: Update with solution agents.

```
public agent PositionUpdateAgent extends Agent {
   #posts event UpdateRequest updateRequestET;
   #sends event Update ev;
   #handles event UpdateRequest;
   #uses plan SendUpdateRequest;
   public PositionUpdateAgent(String name)
   {
      super(name);
   }
   public void submitUpdateRequest()
          {
             postEvent(updateRequestET.request());
          }
}
```

Figure 4.2 Position update agent's update request

4.1.3 Trigger Agent

Trigger agent is responsible for matching solution agents for communicating and triggering solution agents so as to negotiate their solutions during the communication period. Trigger agent selects a solution agent, then, match it with another solution agent that is not in communication at that moment. Its descriptors are summarized in Table 4.3 and its definition on JACK is depicted in Figure 4.3.

Table 4.3 Trigger agent descriptors

Name: Trigger agent

Description: Matches solution agents.

Lifetime: Initialized when a run command reaches to the system from JAVA. Demised when simulation finishes.

Initialization: Obtains run command from JAVA.

Demise: Finishes all the simulation process.

Functionalities included: Matching solution agents functionality.

Uses data: Solution agent name data, solution agent negotiation lock data.

Produces data: Matched pairs of solution agents data.

Goals: Match solution agents.

Percepts responded to: Go to saloon and negotiate plan.

Actions: Match solution agents.

Protocols and interactions: Trigger with solution agents.

```
public agent TriggerAgent extends Agent {
   #handles event TriggerEvent;
   #sends event SendPositionUpdateEvent ev;
   #uses plan TriggerPlan;

   public TriggerAgent(String name)
   {
      super(name);
   }

}
```

Figure 4.3 Trigger agent

4.1.4 Solution Agent

Solution agents are main actors of the simulation process. Each solution agent represents a feasible solution in the system. These solution agents try to find optimal solution of the given problem through negotiating with each other and searching the solution area by themselves. Solution agents desire to find a negotiable solution agent (solution agent that is not making a conversation at that moment) during negotiating process. After negotiating process, solution agents either do crossover (it means that the solution agent communicate with an agent which has a better solution) or do local search (it means that the agent could not find an agent which has a better solution). Its descriptors are summarized in Table 4.4 and its definition on JACK is depicted in Figure 4.4.

Table 4.4 Solution agent descriptors

Name: Solution agent

Description: Represents a single solution of the given problem.

Lifetime: Initialized when start request reaches to it from manager agent. Demised when simulation finishes.

Initialization: Start request command from manager agent.

Demise: Finishes all the simulation process.

Functionalities included: Searching solution area functionality, negotiating functionality.

Uses data: Self solution data, fitness and solution data of its pair.

Produces data: New solution of the given problem.

Goals: Finding optimal solution of the given problem.

Percepts responded to: Initialize agents plan, do crossover plan, negotiate with others plan and go to hill after self-reasoning plan.

Actions: Find possible solutions of the given problem through communicating other agents or self-reasoning.

Protocols and interactions: ---

```
public agent SolutionAgent extends Agent {
   #posts event Update ev;
   #handles event Update;
   #handles event Start;
   #handles event SendPositionUpdateEvent;
   #posts event TriggerEvent ev1;
   #handles event TriggerEvent;
   #uses plan GoToHillWithSelfReasoning;
   #uses plan GoToSaloonAndNegotiate;
   #uses plan InitializeAgents;
   #uses plan NegotiateWithOtherAgents;
   #uses plan GoToHillSelfReasoningAfterSaloon;
   #uses plan DoCrossOver;

   public SolutionAgent(String name, SolutionE eclipseS)

   {
      super(name);
      solution=eclipseS;

   }
```

Figure 4.4 Solution agent

4.2 Plans and Events

Each type of agents in the system has several duties. These agents carry out these duties through plans and events. In this part of the chapter, plans and events of each agent are listed.

4.2.1 Manager Agent's Plans and Events

Manager agent is responsible for two plans and three events. By handling these plans and events, manager agent sends start request to the solution agents and display solution agent data.

a. **Manager Agent's Plans:** Manager agent has the following plans:

   i. *Send start request plan:* Manager agent sends the command of start to the solution agents through this plan.
   ii. *Display plan:* Manager agent shows solution agents' solution via display plan.

b. **Manager Agent's Events:** Manager agent has the following events:

i. *Start request event:* It is the event of specifies the manager agent's request functionality.
ii. *Display request event:* It is the event of specifies the manager agent's display functionality.
iii. *Start event:* It is the event of specifies the manager agent's start functionality.

4.2.2 Position Update Agent's Plans and Events

Position update agent is responsible for making a plan and two events. By handling these plan and events, position update agent sends the message of comeback to saloon to the solution agents.

a. **Position Update Agent's Plans:** Position update agent has the following plan:

i. *Send update request plan:* Position update agent sends the command of update position to the solution agents through this plan.

b. **Position Update Agent's Events:** Position update agent has the following events:

i. *Update request event:* It is the event of specifies the position update agent's update request functionality.
ii. *Update event:* It is the event of specifies the position update agent's update functionality.

4.2.3 Trigger Agent's Plans and Events

Trigger agent is responsible for a plan and two events. By handling these plan and events, trigger agent matches solution agents with each other for negotiating their solutions.

a. **Trigger Agent's Plans:** Trigger agent has the following plan:

i. *Trigger plan:* Trigger agent matches the solution agents for negotiating through this plan.

b. **Trigger Agent's Events:** Trigger agent has the following events:

i. *Trigger event:* It is the event of specifies the trigger agent's triggering solution agents for matching functionality.
ii. *Send match event:* It is the event of specifies the manager agent's match functionality.

4.2.4 Solution Agent's Plans and Events

Solution agent is responsible for six plans and three events. Through handling these plans and events, solution agents search solution space and try to find the optimal solution for the present problem.

a. Solution Agent's Plans: Solution agent has the following plans.

i. *Initialize agents plan:* Solution agents initialize themselves and get their first solutions through this plan.
ii. *Go to saloon and negotiate plan:* Solution agents come back to saloon and get ready for matching through this plan.
iii. *Go to hill with self-reasoning plan:* Solution agents try to find new solutions after initialization through this plan.
iv. *Negotiate with other agents plan:* Solution agents negotiate with other solution agents and share their best solutions through this plan.
v. *Do crossover plan:* Solution agents implement lox-crossover with a better solution agent through this plan.
vi. *Go to hill self-reasoning after saloon plan:* Solution agents implement local search through this plan.

b. Solution Agent's Events: Solution agent has the following events.

i. *Start event:* It is the event of specifies solution agent's get the message of start functionality.
ii. *Update event:* It is the event of specifies solution agent's get the message of update functionality.
iii. *Trigger event:* It is the event of specifies solution agent's get the message of trigger functionality.

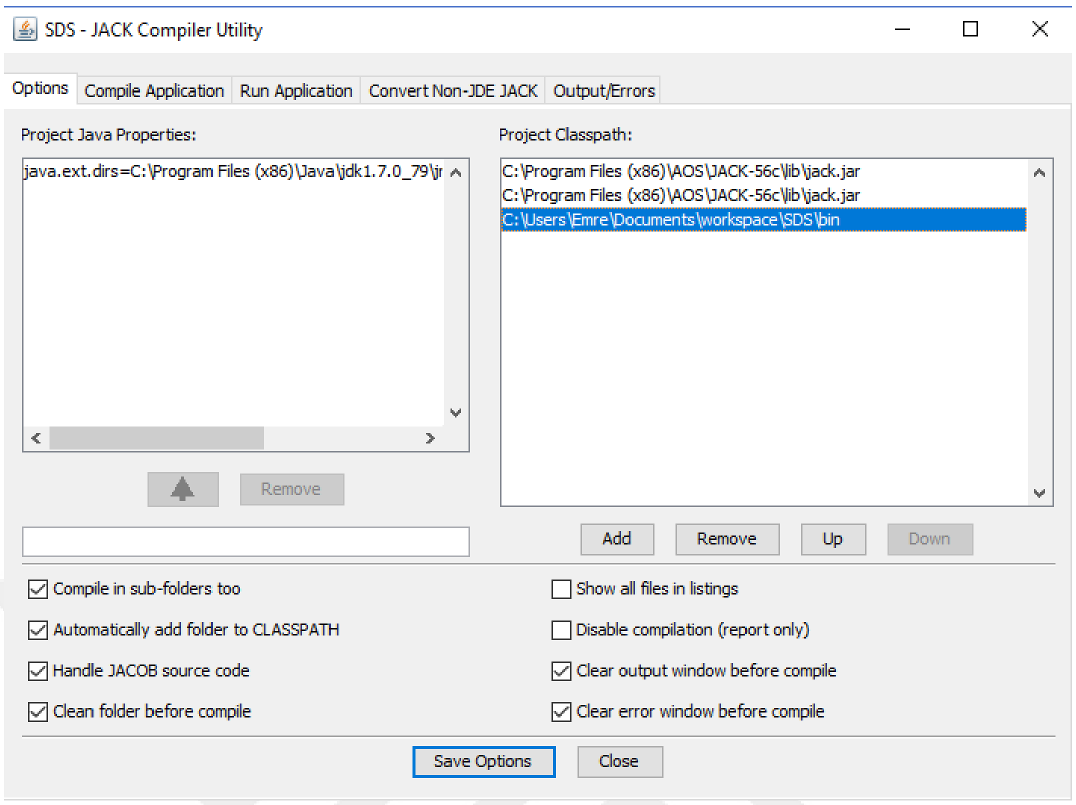
4.3 Connection between JACK and Eclipse JAVA Platform

JACK multi-agent based simulation platform and Eclipse Java platform can be cooperated and get connected. Design of the proposed model is implemented in JACK and agent's procedural applications are implemented in Java.

4.3.1 JACK and Eclipse JAVA Platform Connection Steps

In order to connect JACK and Java the following steps should be applied:

i. In JACK, compiler utility tab on the tools menu should be clicked and class path of the Java project should be added on the options tab project class path area on the pop-up screen as it is illustrated in Figure 4.5.
ii. In Eclipse Java platform, class path of the JACK project should be added as an external class file on the java build tab on properties menu as it is depicted in Figure 4.6.



Project Java Properties:

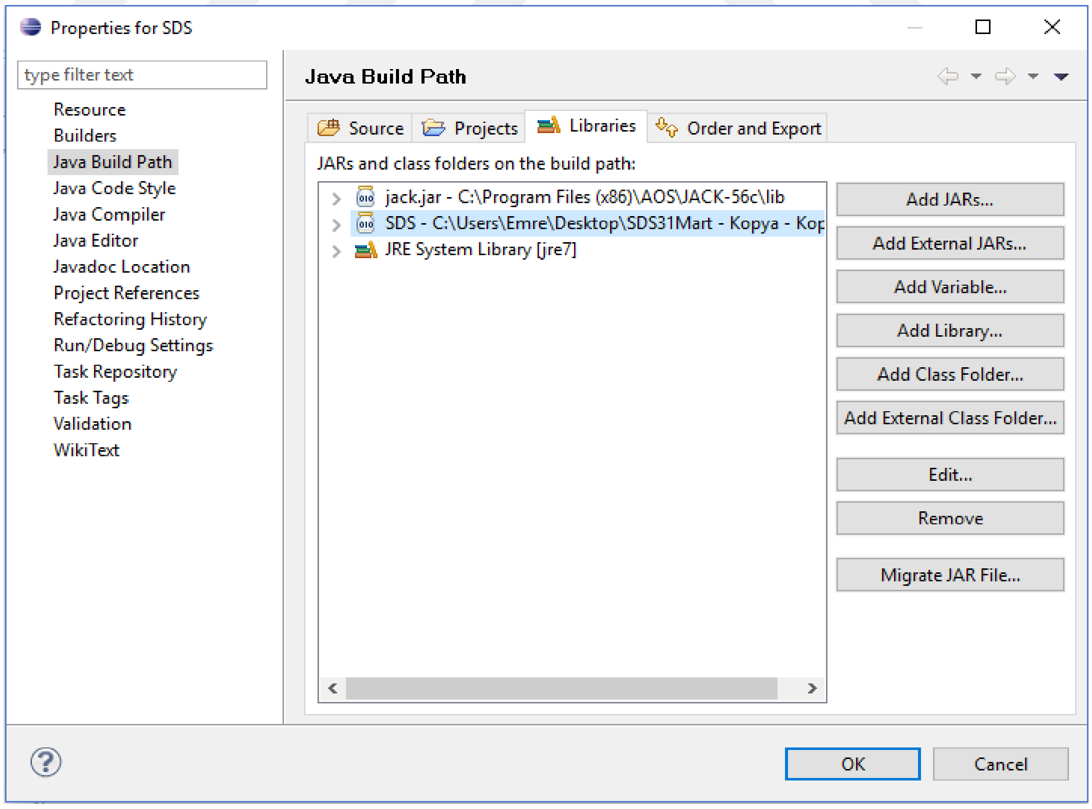
```
java.ext.dirs=C:\Program Files (x86)\Java\jdk1.7.0_79\jr
```

Project Classpath:

```
C:\Program Files (x86)\AOS\JACK-56c\lib\jack.jar
C:\Program Files (x86)\AOS\JACK-56c\lib\jack.jar
C:\Users\Emre\Documents\workspace\SDS\bin
```

- ☑ Compile in sub-folders too
- ☑ Automatically add folder to CLASSPATH
- ☑ Handle JACOB source code
- ☑ Clean folder before compile
- ☐ Show all files in listings
- ☐ Disable compilation (report only)
- ☑ Clear output window before compile
- ☑ Clear error window before compile

Figure 4.5 Connection operations in JACK side



JARs and class folders on the build path:

```
> jack.jar - C:\Program Files (x86)\AOS\JACK-56c\lib
> SDS - C:\Users\Emre\Desktop\SDS31Mart - Kopya - Kop
> JRE System Library [jre7]
```

Figure 4.6 Connection operations in JAVA side

4.3.2 Operations on JAVA Platform for the Proposed Model

The following operations of the proposed model are carried out on JAVA platform.

i. *Creating solution agent's constructor:* Solution agent's constructor is defined on JAVA. Solution agent constructor is demonstrated in Figure 4.7.

```
        public class SolutionE implements Serializable {
        public String solutionName;
        public Status agentStatus;
        public boolean NegotiationLock;
        public int fitness;
        public int bestFitness;
        public Vector<Operation> solution = new Vector<Operation>();
        public Vector<Operation> temporarySolution = new Vector<Operation>();
        public Vector<Operation> bestSolution = new Vector<Operation>();
        public Vector<Operation> crssolutions = new Vector<Operation>();
        public boolean isLast;
        public SolutionE(String solutionName1, boolean
        agentNegotiationLock,
        int agentFitness,boolean agentisLast) {
        solutionName = solutionName1;
        NegotiationLock = agentNegotiationLock;
        fitness = agentFitness;
        isLast=agentisLast;
}
```

Figure 4.7 Solution agent constructor

ii. *Determining solution representation:* Operation class is defined on JAVA for solution representation. Each solution agent has a solution operation type solution vector. Operation class constructor is shown in Figure 4.8.

```
public class Operation implements Serializable {
public int No;
public int ptime;
public int ddate;
public int weight;
public boolean check;
public Operation(int OperationNo,int Operationptime,int Operationweight,int
Operationddate){
No=OperationNo;
ptime=Operationptime;
weight=Operationweight;
ddate=Operationddate;
                }
}
```

Figure 4.8 Operation class

iii. *Crossover Operation:* Lox-crossover operation (Falkenauer and Bouffouix, 1991) is coded on JAVA. By the help of connection between JACK and JAVA, solution agents call crossover function on JACK. In other words, all functions that are coded on JAVA are executable on JACK. Crossover operation is shown below in Figure 4.9. For more details on Lox-crossover mechanism one can see Falkenauer and Bouffouix's study (Falkenauer and Bouffouix, 1991).

```
public Vector<Operation> crossOver(Vector<Operation> parent1,
Vector<Operation> parent2) {
int sayi1 = 0;
int sayi2 = 0;
while(sayi2<=sayi1){
sayi1=1+Main.RandomNumber(parent1.size()-1);
sayi2=1+Main.RandomNumber(parent1.size()-1);
}
for (int i = sayi1; i <= sayi2; i++) {
parent1.get(i).check = true;
for (Operation element : parent2) {
if (element.getNo() == parent1.get(i).getNo()) {
element.check = true;
}
}
}
```

Figure 4.9 Crossover operation

```
    Vector<Operation> offSpring = new Vector<Operation>();
    for (Operation element : parent1) {
    if (!element.check) {
    for (Operation element2 : parent2) {
    if (!element2.check) {
    offSpring.add(element2);
    element2.check = true;
    break;
    }
    }
    } else
    offSpring.add(element);
    }

    System.out.println("offSpring");
    for (Operation element : offSpring) {
    System.out.print(element.getNo());

    }

    for (Operation element : parent1) {

    element.check = false;
    }

    for (Operation element : parent2) {

    element.check = false;
    }
    parent2 = offSpring;
    System.out.println("parent2");
    for (Operation element : parent2) {
    System.out.print(element.getNo());

    }
    offSpring = null;

    System.out.println();
    return parent2;

}
```

Figure 4.9 Crossover operation (continue)

iv. *Local Search Mechanisms:* Solution agents' local search mechanisms are also coded on JAVA. Five different local search mechanisms namely "*one-block swap", "two-block swap", "three-block swap", "one-block insertion" and "two-block insertion"* are executable by solution agents in JACK throughout the connection between JACK and JAVA. For more details on these local search mechanisms see the following reference (Anderson, 1996).

v. *Overall Control of the System:* System operations such as creating agent instances, output reports, entering problem data are implemented via JAVA. Simulation runs are also controlled on JAVA. Implementation of these operations is also illustrated in Figure 4.10.

```
public static void main(String[] args) {

int numberOfAgents=50;
String solutionAgentName;
for (int i=0;i<numberOfAgents;i++){
solutionAgentName="SolutionAgent"+i;
SolutionE v=new SolutionE(solutionAgentName,false,10000,false);
v.setName(solutionAgentName);
SolutionAgent sA=new SolutionAgent(v.getName(),v);
sA.deneme();
Main.vector.add(v);
}
//Main.returnSolutionAgents();

TriggerAgent triggerAgent=new TriggerAgent("trigger");
ManagerAgent managerAgent=new ManagerAgent("Manager Agent");
managerAgent.submitStartRequest();
PositionUpdateAgent updateAgent=new PositionUpdateAgent("Position
Agent");
updateAgent.submitUpdateRequest();
}
```

Figure 4.10 Implementation of system operations

4.4 Flow of the Proposed Simulation Model

In this section, overall model of the proposed simulation model is explained in detail. Agent's types, agent plans and events, interactions between the different agents and implementation of SDS algorithm in JACK are expressed in a detailed manner. The general overview of the proposed model is illustrated below in Figure 4.11

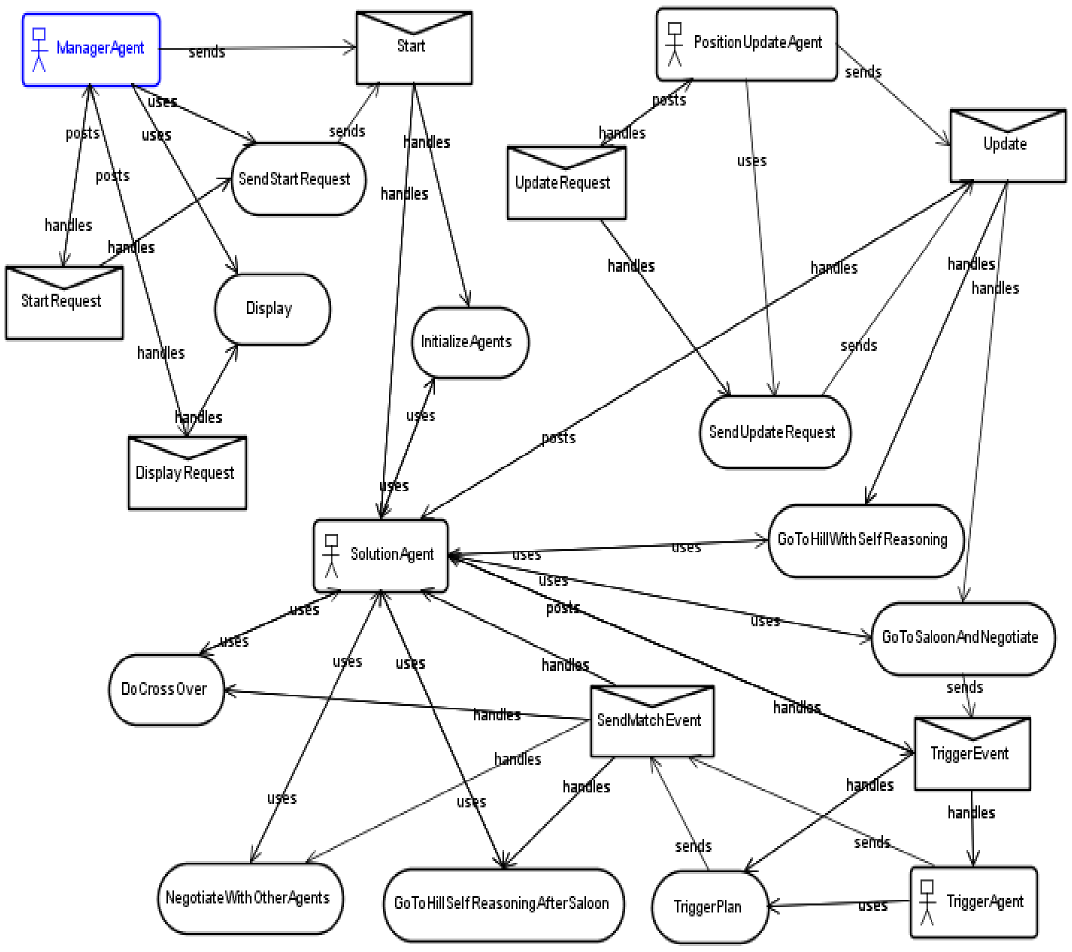


Figure 4.11 General overview of the proposed model

4.4.1 Initialization of the System

The proposed simulation model is initialized by the manager agent and solution agents. The general overview of initialization process and interaction between manager and solution agents are illustrated in Figure 4.12.

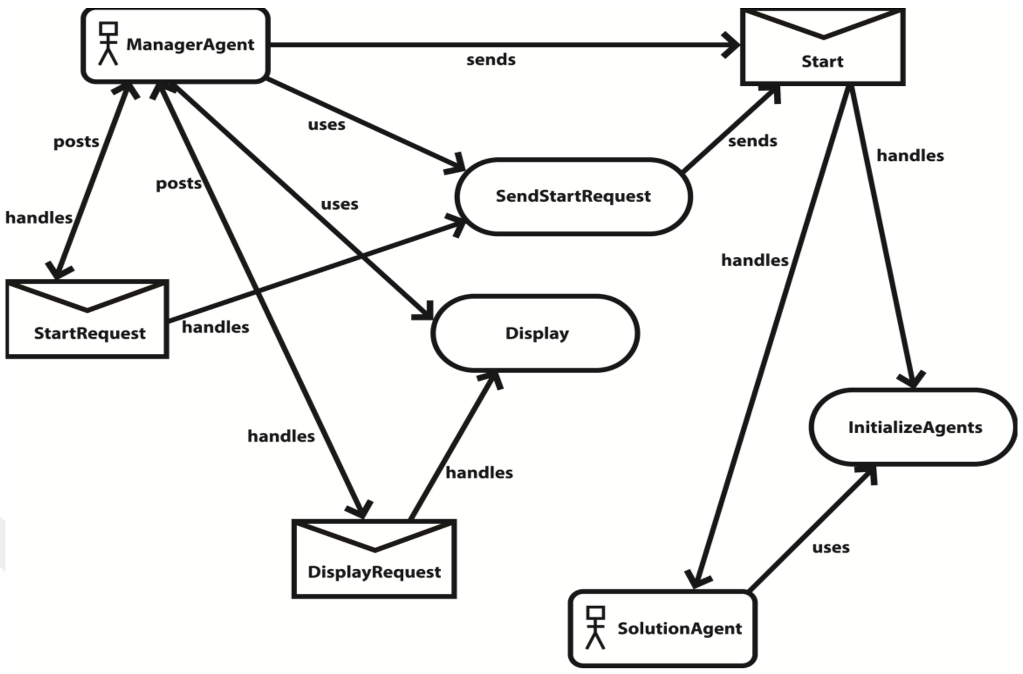


Figure 4.12 General overview of initialization process

An instance of manager agent is created on JAVA and this agent send start request to the solution agents through "*SendStartRequest*" plan which is illustrated in Figure 4.13.

```
public plan SendStartRequest extends Plan {

   #handles event StartRequest startRequestET;
   #sends event Start startET;

   static boolean relevant(StartRequest ev)
   {
      return true;
   }
   context()
   {
      true;
   }
   #reasoning method
   body()
   {
      @send(startRequestET.solution,startET.startMethod(startRequestET.planNo));
      }
}
```

Figure 4.13 "SendStartRequest" plan

In this plan, request method which is defined in "*StartRequest*" event is executed and manager agent sends start message to the all of the solution agents within the system. The details of request method can be seen in Figure 4.14.

```
public event StartRequest extends BDIGoalEvent {
public String solution;
public int planNo;

#posted as
request(String s, int p)
{
solution=s;
planNo=p;
}

}
```

Figure 4.14 Request method

Solution agents that received the start message, generate their first solutions by implementing "*InitializeAgents*" plan. Throughout this plan, solution agents execute "setSolution" and "calculateFitness" procedures on JAVA. By making use of these procedures, initialization of system process is completed. Moreover, throughout "Output" class on JAVA, all of the solution agents' findings are also recorded. This plan is shown in Figure 4.15.

```
package solution;
import elements.Main;
import elements.Output;
public plan InitializeAgents extends Plan {
   #handles event Start ev;
   #uses interface SolutionAgent self;
   static boolean relevant(Start ev)
   {
      return true;
   }
```

Figure 4.15 "InitializeAgents" plan

```
        context()
        {
            true;
        }
        #reasoning method
        body()
        {
            System.out.println(self.solution.getName());
             int best=999999;
                for (int i=0;i<1;i++){
                    self.solution.solution.clear();
                    self.solution.setSolution();
                    self.solution.setSolution();
                    self.solution.showSolution(self.solution.solution);
                    int fit=self.solution.CalculateFitness(self.solution.solution);
                    System.out.println(fit);
                    if (best>fit){
                        self.solution.bestSolution=self.solution.solution;
                        best=fit;
                        self.solution.bestFitness=best;
                }
          }
          Main.statistics.add(new
Output(self.solution.getName(),self.solution.bestSolution,self.solution.bestFitness));
        }
}
```

Figure 4.15 “InitializeAgents” plan (continue)

4.4.2 Position Update Mechanism

Position update mechanism is under the responsibility of position update agent and trigger agent. A general overview of position update mechanism is illustrated in Figure 4.16.

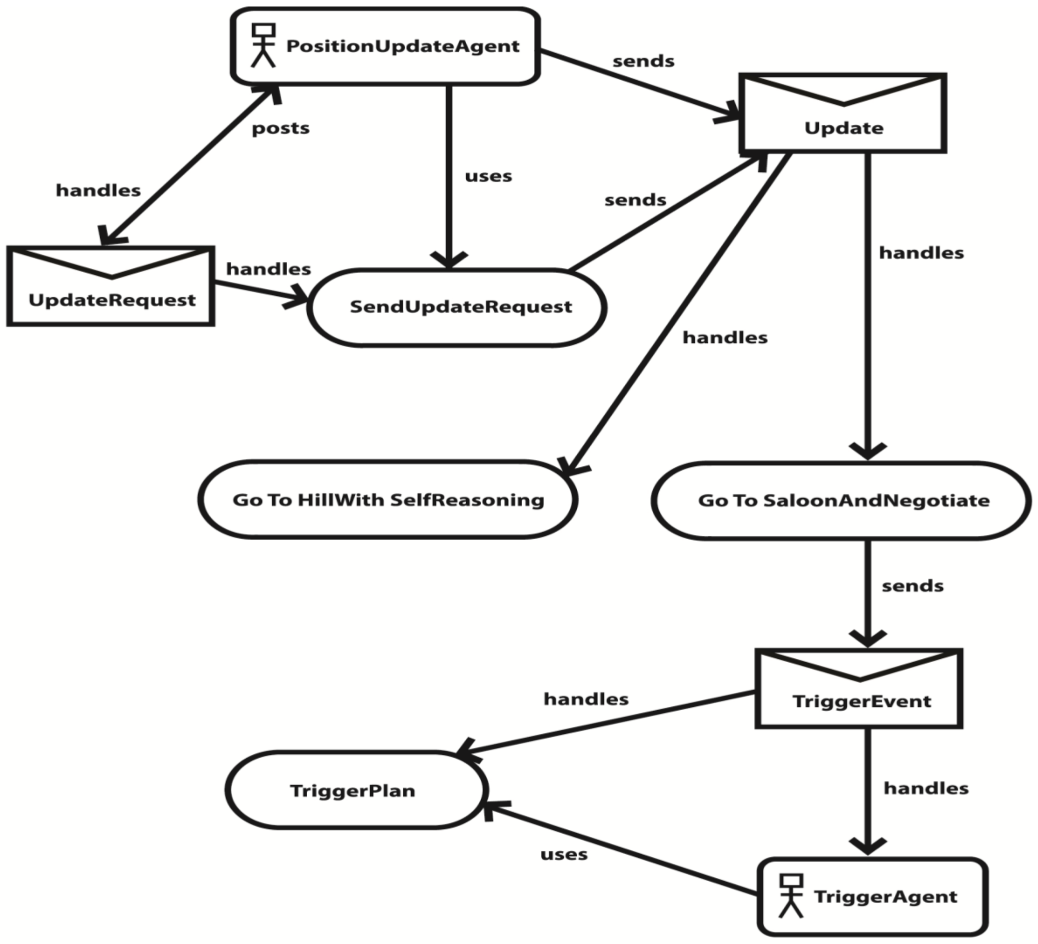


Figure 4.16 General overview of position update mechanism

Position update agent sends update request to solution agents through "SendUpdateRequest" plan which is shown in Figure 4.17.

```
public plan SendUpdateRequest extends Plan {
   #handles event UpdateRequest updateRequestET;
   #sends event Update updateET;
   static boolean relevant(UpdateRequest ev)
   {
      return true;
   }
   context()
   {
      true;
   }
```

Figure 4.17 "SendUpdateRequest" plan

```
   #reasoning method
   body()
   {
           int numberOfSolutionAgent=Main.vector.size();
           String solutionAgentName;
                 int j=0;
                 int time=0;
                 int ctime=0;
                 Main.numberOSolutionAgentsInSaloon=numberOfSolutionAgent;
                  while (j<20)
                  {

                       time=0;
                       while (time<15){
                           for (int i=0;i<numberOfSolutionAgent;i++)
                    {
                      solutionAgentName="SolutionAgent"+i;

@send(solutionAgentName,updateET.updateMethod(2,"trigger",solutionAgentName
));
                    ctime=1+Main.RandomNumber(3);
                    time=time+ctime;
                    @sleep(ctime);
                   }
                   }
                     for (int i=0;i<numberOfSolutionAgent;i++)
                    {
                      SolutionE ssagent=(SolutionE)Main.returnSolutionAgents().get(i);
                      ssagent.isLast=true;
                      ssagent.NegotiationLock=false;
                      solutionAgentName="SolutionAgent"+i;

@send(solutionAgentName,updateET.updateMethod(2,"trigger",ssagent.getName())
);
                    }
                     j++;
                     Main.numberOSolutionAgentsInSaloon=0;
                  }
   }
```

Figure 4.17 “SendUpdateRequest” plan (continue)

After taking the message of position update (comeback to the saloon) via update method of “Update” event, all of the solution agents discontinue their solution searching process and get ready for communication and negotiation with other solution agents. Detailed information of update method is illustrated in Figure 4.18.

```
public event Update extends BDIMessageEvent {
    public int planNo;
    public String trigger;
    public String solutionAgentName;

    #posted as
    updateMethod(int p, String t, String s)
    {
      planNo=p;
      trigger=t;
      solutionAgentName=s;

    }

}
```

Figure 4.18 Update event

This communication and negotiation process is carried on for a certain time period. During this period, all of the solution agents try to find a negotiable solution agent (solution agent that is not communicating with another solution agent at that specific moment) via help of trigger agent. If a solution agent finds another solution agent to talk, communication time proceeds for minimum time of ongoing conversations. The solution agents that take update message, executes trigger method of “Trigger” event on “GoToSaloonAndNegotiate” plan. Detailed information of “GoToSaloonAndNegotiate” plan and trigger method are illustrated in Figure 4.19 and 4.20 respectively.

```
public plan GoToSaloonAndNegotiate extends Plan {
    SolutionE crossOverSolutionAgent;
    #handles event Update updateET;
    #sends event TriggerEvent triggerEventET;
    static boolean relevant(Update ev)
    {
        return true;
    }
    context()
```

Figure 4.19 “GoToSaloonAndNegotiate” plan

```
   {
      updateET.planNo==2;
   }
   #reasoning method
   body()
   {
      for(int i=0;i<5;i++){
         if (self.solution.NegotiationLock==false)
         {
@send(updateET.trigger,triggerEventET.triggerMethod((SolutionE)Main.returnSolut
ionAgents().get(i), self.solution));
      }
      }
   }
}
```

Figure 4.19 “GoToSaloonAndNegotiate” plan (continue)

```
public event TriggerEvent extends BDIMessageEvent {
   public SolutionE crossOverSolutionAgent;
   public SolutionE solutionSenderObject;
   #posted as
   triggerMethod(SolutionE s, SolutionE ss)
   {
      crossOverSolutionAgent=s;
      solutionSenderObject=ss;
   }

}
```

Figure 4.20 Trigger method

When the communication process is completed, all solution agents’ “isLast” area are marked as true for denoting the end of the conversations. The last message is sent to the solution agents for making decisions about the next move (All solution agents go their rooms for deciding the next move).

4.4.3 Negotiation and Determining Next Move Mechanism

Trigger agent has the duty of matching negotiable solutions agents and triggering matched solution agents to get ready for negotiation. Solution agents that is matched and triggered for negotiation begin to negotiate. General overview of the negotiation and determination of the next move mechanism is illustrated in Figure 4.21.

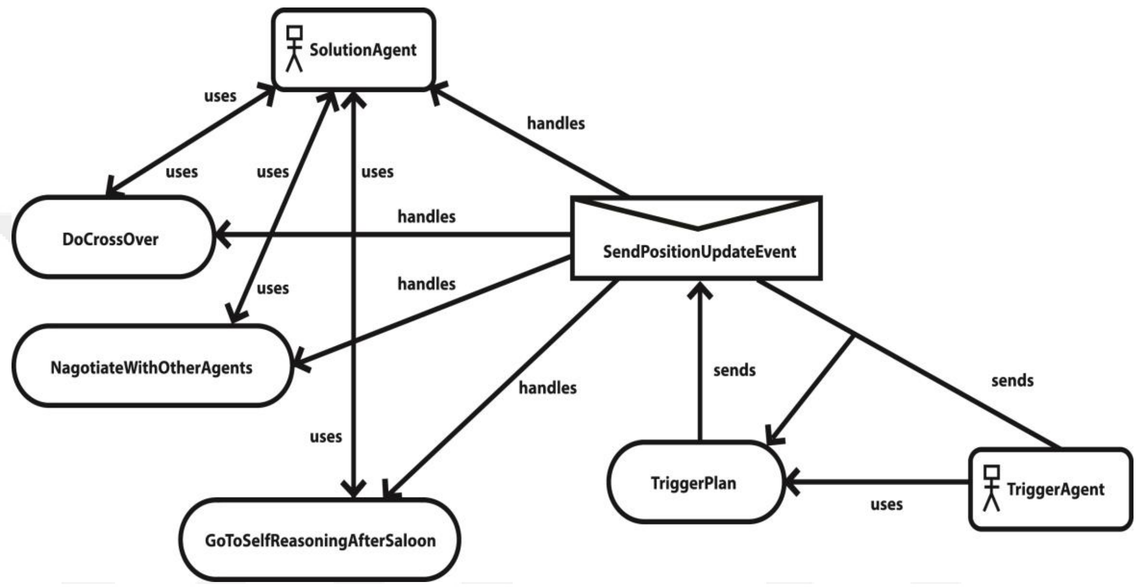


Figure 4.21 General overview of negotiation and next move mechanism

At this point, trigger agent executes trigger plan which is depicted in Figure 4.22.

```
public plan TriggerPlan extends Plan {
   #handles event TriggerEvent triggerEventET;
   #sends event SendMatchEvent sendPositionUpdateEventET;
   static boolean relevant(TriggerEvent ev)
   {
      return true;
   }
   context()
   {
      true;
   }
   #reasoning method
   body()
   {
send(triggerEventET.solutionSenderObject.getName(),sendPositionUpdateEventET.send
Method(triggerEventET.crossOverSolutionAgent,
triggerEventET.solutionSenderObject));
   }
}
```

Figure 4.22 Trigger plan

In context of trigger plan, trigger agent uses send request method of “SendMatchEvent”. Throughout this method, solution agents learn their negotiation partner’s name. In this method, “solutionSenderObject” represents the matched pair of solution agent that named as “crossOverSolutionAgent”. Details of the send request method is shown in Figure 4.23.

```
public event SendMatchEvent extends BDIMessageEvent {
   public SolutionE crossOverSolutionAgent;
   public SolutionE solutionSenderObject;
   #posted as
   sendMethod(SolutionE s, SolutionE ss)
      {
         crossOverSolutionAgent=s;
         solutionSenderObject=ss;
    }
}
```

Figure 4.23 Send method

This negotiation process depends on some conditions. If these conditions are satisfied, solution agents execute “NegotiateWithOtherAgents” plan. In order to execute this plan, the following three conditions must be satisfied.

1. Both of the agents “crossOverSolutionAgent” and “solutionSenderObject”, must be ready to negotiate. In other words, both of the agents’ negotiation locks must be marked as false.

2. Conversation among the “crossOverSolutionAgent” and “solutionSenderObject” agents must be held at specified negotiation time interval. In other words, “crossOverSolutionAgent” ’s “isLast” area must be marked as false. Because, after the negotiation time completion, all solution agents’ “isLast“ area marked as true in order to clarify the end of negotiation time.

3. Another issue is that, “solutionSenderObject” agent’s best fitness must be better than “crossOverSolutionAgent” agent’s one. In other words, a

solution agent does not gain anything by negotiating with a less successful agent than it.

After providing these three conditions, negotiation between the solution agents can be begun. At this point, “crossOverSolutionAgent” may face one of the following three cases.

i. **Case 1 “crossOverSolutionAgent” has not negotiated with a better solution agent until that time:** In this case, “crossOverSolutionAgent”’s “crssolution” area is null. So that “crossOverSolutionAgent” copies “solutionSenderObject”’s best solution directly to its “crssolution” area.

ii. **Case 2 “solutionSenderObject”’s best solution is better than the solution of “crossOverSolutionAgent”’s “crssolution”:** In this case, “crossOverSolutionAgent” ‘s “crssolution” area is not null. Thus “crossOverSolutionAgent” should clear its “crssolution” area before copying “solutionSenderObject”’s best solution to its “crssolution” area. Because it is not meaningful to keep a worse solution of the pre-negotiated solution agent after negotiating a better solution agent.

iii. **Case 3 “solutionSenderObject”’s best solution is worse than the solution of “crossOverSolutionAgent”’s “crssolution” area :** In this case, “crossOverSolutionAgent”’s “crssolution” is better than *“solutionSenderObject”’*s best solution. Therefore,“crossOverSolutionAgent” ends the conversation process and do nothing. Because, it is not meaningful to copy*“solutionSenderObject”*‘s best solution, if “crossOverSolutionAgent” has already negotiated a better solution agent than “solutionSenderObject” agent.

All of these pre-conditions for negotiation and solution agents’ options are illustrated in Figure 4.24 and Figure 4.25 respectively.

```
public plan NegotiateWithOtherAgents extends Plan {
   #handles event SendMatchEvent sendPositionUpdateEventET;
   #uses interface SolutionAgent self;
   static boolean relevant(SendMatchEvent ev)
   {
      return true;
   }
   context()
   {
   (sendPositionUpdateEventET.crossOverSolutionAgent.getBestFitness()<
     self.solution.getBestFitness()       &&
  sendPositionUpdateEventET.crossOverSolutionAgent.NegotiationLock== false
&& self.solution.NegotiationLock==false &&
       self.solution.isLast==false);
   }
```

Figure 4.24 Negotiation conditions

```
if (self.solution.crssolutions.size()==0){
self.solution.crssolutions.addAll(self.solution.getcrssolutions(sendPositionUpdateEv
entET.crossOverSolutionAgent.getBestSolution()));
     System.out.println(self.solution.crssolutions.size());
      }else if (self.solution.CalculateFitness(self.solution.crssolutions) >
self.solution.CalculateFitness(self.solution.getcrssolutions(sendPositionUpdateE
ventET.crossOverSolutionAgent.getBestSolution())))
      {
         self.solution.crssolutions.clear();
self.solution.crssolutions.addAll(self.solution.getcrssolutions(sendPositionUpdateEv
entET.crossOverSolutionAgent.getBestSolution()));
       }else {
          System.out.println("ajan daha iyi bir ajanla konuşmuş");
       }
```

Figure 4.25 Negotiation cases

When the time for negotiation ends, all of the solution agents' "isLast" area marked as true and all solution agents get ready for determining their next moves. In detail, miners should decide for the next hill for digging.

Solution agents have two plans namely "DoCrossOver" and "GoToHillSelfReasoningAfterSaloon" to execute for determining their next moves. Solution agents select a plan according to their findings to execute during the negotiation period. Solution agents record their findings "crssolutions" area. After negotiation process, if a solution agent's "crssolutions" area is null, it means that, solution agent has not communicated with another solution agent with a better solution than its solutions. In this case, solution agent executes "GoToHillSelfReasoningAfterSaloon" plan and determines its next move by applying one of the following local search mechanisms.

i. ***One-Block Swap:*** One-block swap-move tswp (Vector<Operation> X) swaps the positions of two elements in the current vector X which is given in Figure 4.26. Element x_i at position i is exchanged with element x_j at position j. The other elements all other positions are unaffected by this move. Assume that the randomly chosen positions are *i*=3 and *j*=7.

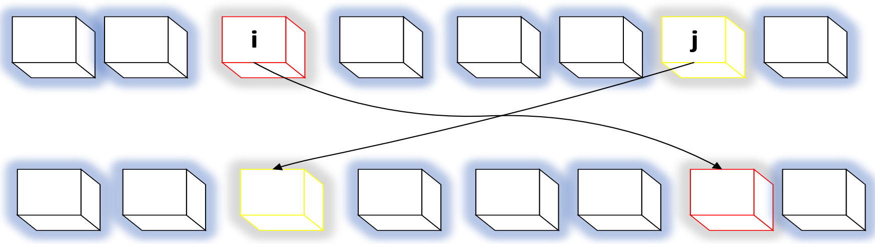


Figure 4.26 One-block swap

ii. ***Two-Block Swap:*** Two-block swap is based on replacing two different blocks. The positions are chosen randomly for two blocks. The most important point is that the chosen block's lengths must be equal. Figure 4.27 demonstrates an example of this neighborhood generation model. For instance, if the randomly chosen positions are 2 and 5, it explains that the two framed blocks (7,5) and (3,6) which are depicted in Figure 4.27 are randomly chosen. Then, these chosen blocks are replaced to generate a new solution vector.

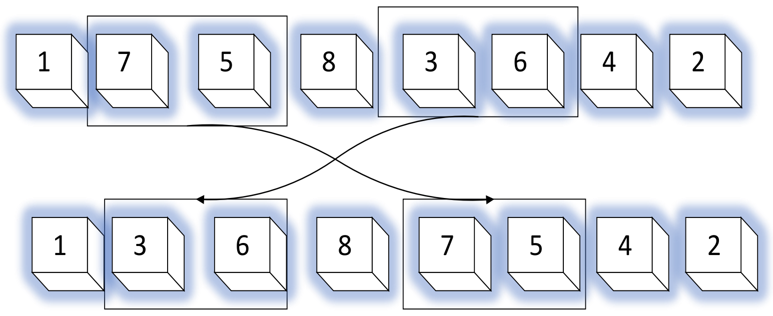


Figure 4.27 Two-block swap

iii. ***Three-Block Swap:*** Three-block swap is based on replacing two different blocks with an equal length three. The positions of the blocks are randomly chosen. Figure 4.28 illustrates an instance of this neighborhood mechanism. For example, if the chosen positions are 2 and 6, it explains that the two framed blocks (5,7,8) and (1,3,2), which are depicted in Figure 4.28. Then, these two blocks are replaced to generate a new solution.

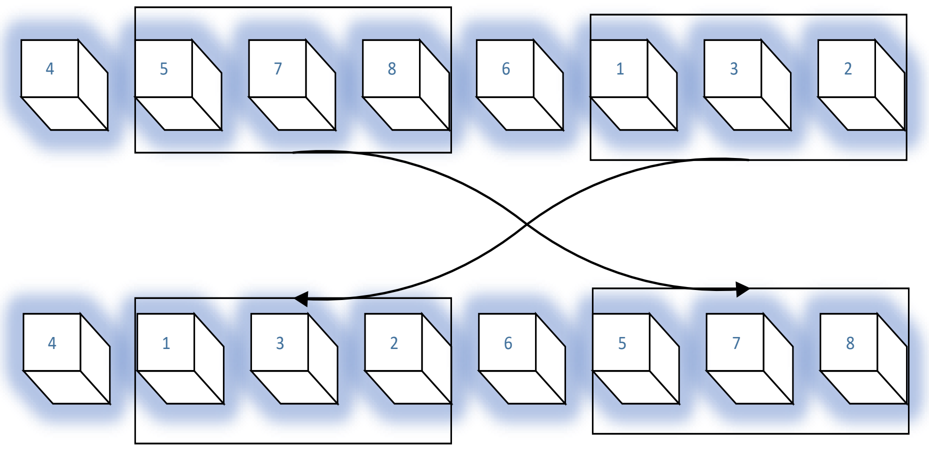


Figure 4.28 Three-block swap

iv. ***One-Block Insertion:*** One-block insertion **tinsertion** (Vector<Operation> X) inserts a selected element in the current vector X in to a randomly selected position, which is depicted in Figure 4.29. Element x_i at position *i* is inserted at position *j*. Assume that the randomly chosen positions are *i*=3 and *j*=7 respectively.

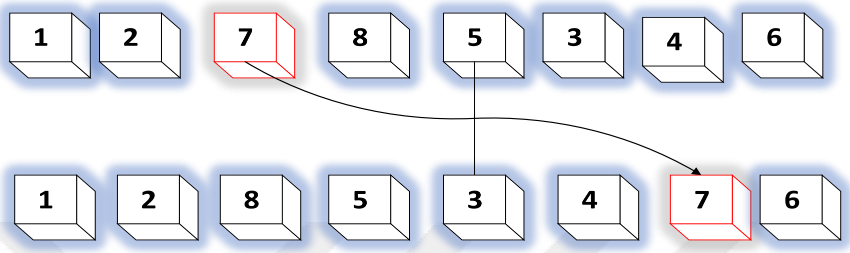


Figure 4.29 One-block insertion

v. ***Two-Block Insertion:*** Two-block insertion is aimed at inserting a block with a length two in the solution. The position of the block is randomly selected. Figure 4.30 shows an example of this neighborhood structure.

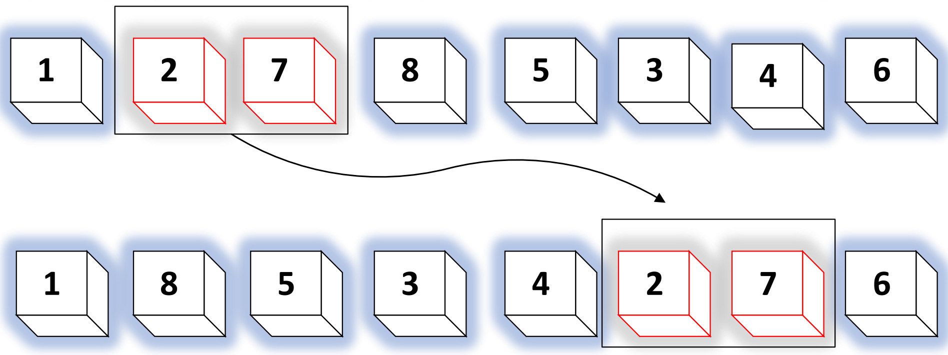


Figure 4.30 Two-block insertion

On the other hand, if a solution agent's "crssolutions" area is not null, it means that, solution agent have communicated with another solution agent with a better solution than its solutions. In this case, solution agent executes "DoCrossOver" plan and determines its next move by applying LOX-Crossover with the solution that is recorded in "crssolutionsarea".

The steps of crossover LOX which was firstly developed by Falkenauer and Bouffouix (1991) can be listed as in the following way

Step 1. Choose randomly a subsequence of operations from one parent.
Step 2. Generate a proto-offspring by duplicating the subsequence into the related positions of it.
Step 3. Erase the operations that exist in the subsequence from the second parent.
Step 4. Put the operations into the unfixed positions of the proto-offspring from left to right related to the rank of the sequence to generate an offspring. This method is demonstrated in Figure 4.31. It also shows an instance of producing one offspring.

Crossover LOX tries to protect both the relative positions between genes and the absolute positions according to the extremities of parents as much as possible. The extremities correspond to the high- and low-priority operations.

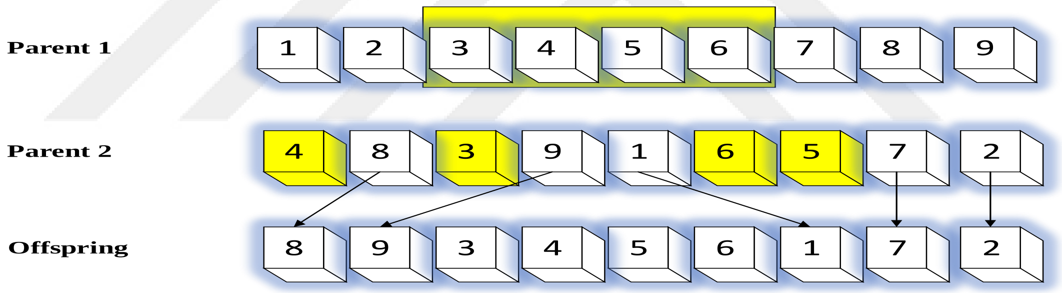


Figure 4.31 Lox crossover move

CHAPTER FIVE
MULTI AGENT-BASED STOCHASTIC DIFFUSION SEARCH ALGORITHM FOR OPTIMIZATION PROBLEMS: THE SINGLE MACHINE TOTAL WEIGHTED TARDINESS CASE

In this chapter, the presented algorithm is tested on the single machine total weighted tardiness problem (SMTWT). SMTWT can be defined as in the following way: a group of independent n jobs is to be completed on a single machine that is able to process only a job at the same time without any interruption. Every job has a processing time p_i, a due date d_i, and a positive weight w_i. All the jobs are ready for processing at the beginning. According to order of the jobs, the completion time C_i and tardiness T_i =max $\{0, C_i - T_i\}$ can be calculated for every job. If a job i is completed after its due date, then a weighted tardiness penalty $w_i * T_i$ will occur. The goal is to find a job order S to minimize the sum of the weighted penalties: $\sum_{i=1}^{n} w_i T_i$ (Ding et al., 2016).

The problem is very easy to understand. Despite the easy understanding of the problem, the problem has high computational complexity and can be stated as NP-hard (Lenstra et al., 1977). In other words, there is no algorithm to solve this problem in a polynomial time. Most of the researchers acquired good performance of SMTWT as a proof of the algorithm’s usefulness.

5.1 Test Instances and Experimental Protocol

To evaluate the success of the developed multi agent-based SDS algorithm, experiments are conducted on two sets of test problems. The problem instances which are easily solved by Geiger (Geiger, 2009) are taken as the first set from the OR-library. The first benchmark set includes three different problems namely “40”,”50”, and “100” with different sizes. Each size involves 125 instances, so the proposed algorithm is tested on 375 different problems totally. Optimal solutions of each instance are obtained from Pan and Shi’s study (Pan and Shi, 2007).

The second set of the test problems contains larger size problems. There are 250 instances with two different sizes 150 and 200 respectively. According to the available literature, these instances are solved optimally Tanaka et al. (2009).

The primary experiments are conducted on the first problem instance. In detail, 20 independent runs are applied for each instance. According to the existing literature, the following values are reported in Table 5.1 for the performance parameters of the proposed algorithm.

Table 5.1 Reported performance values

pd: The percentage relative deviation of a solution value found by an algorithm from the optimal or best-known solution value.

pd=100(Obtained-Optimum)/Optimum (When Optimum=0; pd=100 Obtained is used)

ad: The average pd value for a sample of 125 instances in 20 independent runs.

md: The maximum pd out of a sample of 125 instances in 20 independent runs.

no:The number of optimal or best-known solution values found out of a sample of 125 instances.

$\mathbf{n_{hit}}$**=** The average hit ratio to the optimal or best-known solution for a sample of 125 instances in 20 independent runs.

There are several metaheuristic algorithms available in the literature for SMTWT problem. For comparing the success of the developed algorithm some of the state of art algorithms is chosen. The selected algorithms for performance comparison are depicted in Table 5.2.

Table 5.2 Selected algorithms

Article	Utilized Algorithm
Besten et al. (2001)	Iterated Local Search (ILS)
Avci et al. (2003)	Problem Space Genetic Algorithm (PSGA)
Bilge et al. (2007)	Tabu Search (TS)
Tasgetiren et al. (2006)	Variable Neighborhood Search (VNS)
Tasgetiren et al. (2006)	Particle Swarm Optimization (PSO)
Tasgetiren et al. (2006)	Differential Evolution (DE)
Wang and Tang (2009)	Population-based Variable Neighborhood Search (PVNS)
Geiger (2010)	Variable Neighborhood Descent (VND)
Grosso et al. (2004)	Iterated Enhanced Dynasearch (GPI-DS)
Ding et. al. (2016)	Breakout Dynasearch (BDS)

According to the experimental results, multi agent-based stochastic diffusion search algorithm (MAB-SDS) shows competitive performance in comparison to the selected algorithms. In problem set of 40 Jobs, all of the problem instances are solved optimally by the proposed algorithm. The other algorithms except ILS, VNS and VND, also solve all instances optimally. ILS, VNS and VND have high values of ad. This shows that ILS, VNS and VND are inferior to the other algorithms in terms of solution quality. The proposed algorithm found optimal results for every sample of 125 instances with 20 independent runs. MAB-SDS outperforms VNS and PVNS with hit ratio of 1 respectively. Experimental results are depicted in Table 5.3.

Table 5.3 Experimental results of 40 Job set

Algorithm	ad	md	n_{hit}	no
ILS	0.13	-	-	125
PSGA	0.000	0.000	-	125
TS	0.000	-	-	125
VNS	0.21	-	0.95	125
PSO	0.000	0.000	1	125
DE	0.000	0.000	1	125
PVNS	0.000	0.000	0.9971	125
VND	0.99	-	-	125
GPI-DS	0.000	0.000	1	125
BDS	0.000	0.000	1	125
MAB-SDS	0.000	0.000	1	125

In problem set of 50 Jobs, MAB-SDS is able to solve all instances optimally. It surpasses ILS, TS, VNS, and VND algorithms and shows competitive performance with the other algorithms in terms of ad. In terms of md, maximum pd out, the proposed algorithm outperforms PSGA. Moreover, when the n_{hit} values of the selected algorithms are compared, VNS and PVNS are inferior to the other algorithms with the ratio of 0.91 and 0.9965 respectively. Summary of the results for 50 Jobs set is illustrated in Table 5.4.

Table 5.4 Experimental results of 50 Job set

Algorithm	**ad**	**md**	$\mathbf{n_{hit}}$	**no**
ILS	0.86	-	-	125
PSGA	0.000	0.020	-	125
TS	0.001	-	-	125
VNS	0.20	-	0.91	125
PSO	0.000	0.000	1	125
DE	0.000	0.000	1	125
PVNS	0.000	0.000	0.9965	125
VND	1.45	-	-	125
GPI-DS	0.000	0.000	1	125
BDS	0.000	0.000	1	125
MAB-SDS	0.000	0.000	1	125

In problem set of 100 Jobs, MAB-SDS can obtain optimal solutions for all the problem instances. The proposed algorithm is much better than ILS, PSGA, TS, VNS and VND with 0.000 value of ad. The results for 100 Jobs are also summarized in Table 5.5.

Table 5.5 Experimental results of 100 Job set

Algorithm	**ad**	**md**	$\mathbf{n_{hit}}$	**no**
ILS	14.50	-	-	125
PSGA	0.020	0.300	-	125
TS	0.007	-	-	125
VNS	0.11	-	0.90	125
PSO	0.000	0.000	1	125
DE	0.000	0.000	1	125
PVNS	0.000	0.000	-	125
VND	0.98	-	-	125
GPI-DS	0.000	0.000	1	125
BDS	0.000	0.000	1	125
MAB-SDS	0.000	0.000	1	125

In order to better assess the performance of the MAB-SDS, larger sized instances that is solved by Tanaka et al.(2009)'s exact algorithm are also considered. The test results are compared with the GPI-DS and BDS algorithms that is proposed by Ding et al. (2016).

In 150 Jobs set, the MAB-SDS algorithm is able to solve the selected problems with an ad value of 0.043 from the optimal solutions respectively. In addition, the proposed algorithm solved problems in 200 Jobs set with an ad value of 0.127. The proposed algorithm is able to provide comparable and acceptable solutions. The detailed results of 150 and 200 Jobs sets are depicted in Table 5.6 and 5.7 respectively.

Table 5.6 Experimental results of 150 Jobs set

Algorithm	**ad**	**md**	**n_{hit}**	**no**
GPI-DS	0.001	0.031	0.9553	125
BDS	**0.000**	**0.047**	**0.9897**	125
MAB-SDS	0.043	0.265	0.9047	125

Table 5.7 Experimental results of 200 Job set

Algorithm	**ad**	**md**	**n_{hit}**	**no**
GPI-DS	0.068	0.867	0.9064	125
BDS	**0.032**	**0.885**	**0.9281**	125
MAB-SDS	0.127	0.932	0.8435	125

CHAPTER SIX
CONCLUSION

6.1 Summary

This dissertation is conducted to model a metaheuristic algorithm, namely SDS, in a multi-agent based modeling environment. To the best of our knowledge, this is the first study that a metaheuristic algorithm specifically, SDS is truly realized in a multi-agent based environment. The algorithm is not modeled as classic procedural algorithms. It is modeled as an ongoing vivid system. All agent movements, their interactions are modeled in flowing time axis. While realizing the SDS algorithm in JACK multi agent-based modeling environment, dynamic nature of the algorithm is mimicked truthfully.

We firstly have defined the agent types and their specifications. Then, each type of agents' events and plans are modeled. In order to handle static vector operations, JACK multi agent-based modeling environment and eclipse JAVA platform are connected and cooperated. Throughout this connection, all of the agents kept their personal information such as agent name, solution vector, agent status, fitness etc. on pre-defined fields in eclipse JAVA platform. In addition, the proposed model is also controlled on eclipse JAVA platform.

SMTWT is selected for testing the performance of the proposed multi-agent based SDS algorithm. Although the main goal in this thesis is not to provide the best possible results but to realize a metaheuristic algorithm truthfully, however the proposed algorithm provides comparable and acceptable solutions.

6.2 Contribution of the Study and Future Works

In most of the real-life applications, the environment changes continuously, actually the number of static problems is very few in real-life cases. Despite of the dynamic structure of the real-life settings, most of the researchers try to solve these problems by evolving static methods. It is very hard to produce appropriate solutions of these dynamic problems by applying static methods. For this reason, most of the researchers fail while solving the real-life optimization problems, optimally in dynamic environments.

In this thesis, a dynamic method is evolved for solving a static problem. In further researches, it is planned to apply this dynamic method to the dynamic optimization problems. The modification of the proposed algorithm for the dynamic problems can also be scheduled as a future work with the help of the proposed algorithm's inherent structure.

REFERENCES

Al-Rifaie, M. M., & Bishop, M. J. (2010). The mining game: a brief introduction to the stochastic diffusion search metaheuristic. *Q: The magazine of AISB*, (130), 8-9.

Aleman, D. M., Wibisono, T. G., & Schwartz, B. (2009). Accounting for individual behaviors in a pandemic disease spread model. In *Winter Simulation Conference,* 1977-1985.

Anderson, E. J. (1996). Mechanisms for local search. *European Journal of Operational Research*, *88*(1), 139-151.

Avci, S., Akturk, M. S., & Storer, R. H. (2003). A problem space algorithm for single machine weighted tardiness problems. IIE Transactions, 35(5), 479–486.

Axtell, R. (2000). Why agents?: on the varied motivations for agent computing in the social sciences. In *Workshop on Agent Simulation: Applications, Models, and Tools*, 3.

Barenji, A. V., Barenji, R. V., Roudi, D., & Hashemipour, M. (2016). A dynamic multi-agent-based scheduling approach for SMEs. *The International Journal of Advanced Manufacturing Technology*, 1-15.

Baykasoglu, A., & Gorkemli, L. (2015). Agent-based dynamic part family formation for cellular manufacturing applications. *International Journal of Production Research*, *53*(3), 774-792.

Baykasoglu, A., & Gorkemli, L. (2016). Dynamic virtual cellular manufacturing through agent-based modelling. *International Journal of Computer Integrated Manufacturing*, 1-16.

Baykasoglu, A., & Kaplanoglu, V. (2011). A multi-agent approach to load consolidation in transportation. *Advances in Engineering Software*, *42*(7), 477-490.

Baykasoğlu, A., & Kaplanoğlu, V. (2015). An application oriented multi-agent based approach to dynamic load/truck planning. *Expert Systems with Applications*, *42*(15), 6008-6025.

Baykasoglu, A., Kaplanoglu, V., Erol, R., & Sahin, C. (2011). A multi-agent framework for load consolidation in logistics. *Transport*, *26*(3), 320-328.

Baykasoglu, A., & Durmusoglu, Z. D. (2014). A classification scheme for agent based approaches to dynamic optimization. *Artificial Intelligence Review*, *41*(2), 261-286.

Beattie, P. D., & Bishop, J. M. (1998). Self-localisation in the 'senario'autonomous wheelchair. *Journal of Intelligent and Robotic Systems*, *22*(3-4), 255-267.

Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with JADE* (Vol. 7). Liverpool: John Wiley & Sons.

Berro, A., & Duthen, Y. (2001). Search for optimum in dynamic environment: a efficient agent-based method. In *GECCO'2001 Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, San Francisco, California*, 51-54.

Bilge, Ü., Kurtulan, M., & Kıraç, F. (2007). A tabu search algorithm for the single machine total weighted tardiness problem. *European Journal of Operational Research*, *176*(3), 1423-1435.

Billiau, G., & Ghose, A. (2008). Robust, flexible multi-agent optimisation using SBDO. *Decision Systems Lab/Center for Software Engineering, Report*, *Report No: 2008–TR03.*

Bishop J. (1989). *Anarchic techniques for pattern classification.* PhD thesis, University of Reading, Reading, UK.

Bishop, J. M., & Torr, P. (1992). The stochastic search network. In *Neural networks for vision, speech and natural language* (pp. 370-387). Amsterdam:Springer Netherlands.

DePasquale, D., Charania, A. C., & Olds, J. R. (2006). Agent-based economic modeling of commercial transportation services to the international space station. In *Space 2006*, 7226.

Ding, J., Lü, Z., Cheng, T. C. E., & Xu, L. (2016). Breakout dynasearch for the single-machine total weighted tardiness problem. *Computers & Industrial Engineering*, *98*, 1-10.

Erol, R., Sahin, C., Baykasoglu, A., & Kaplanoglu, V. (2012). A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles in manufacturing systems. *Applied Soft Computing*, *12*(6), 1720-1732.

Falkenauer, E., & Bouffouix, S. (1991). A genetic algorithm for job shop. *In Robotics and Automation*, 1991. Proceedings., *1991 IEEE International Conference*, 824-829.

Farahvash, P., & Boucher, T. O. (2004). A multi-agent architecture for control of AGV systems. *Robotics and Computer-Integrated Manufacturing*, *20*(6), 473-483.

Geiger, M. J. (2009). The single machine total weighted tardiness problem – Is it (for metaheuristics) a solved problem? *In Proceedings of the 8th Metaheuristics International Conference MIC 2009*, 141.1–141.10.

Gendreau, M., & Potvin, J. Y. (2005). Metaheuristics in combinatorial optimization. *Annals of Operations Research*, *140*(1), 189-213.

González, J. R., Masegosa, A. D., & García, I. J. (2011). A cooperative strategy for solving dynamic optimization problems. *Memetic Computing, 3*(1), 3-14.

Grech-Cini, E. (1995). *Locating facial features.* Phd Thesis, University of Reading,Reading.

Grosso, A., Della Croce, F., & Tadei, R. (2004). An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters, 32(1),* 68–72.

Hanna, L., & Cagan, J. (2009). Evolutionary multi-agent systems: an adaptive and dynamic approach to optimization. *Journal of Mechanical Design*, *131*(1), 110-120.

Howden, N., Rönnquist, R., Hodgson, A., & Lucas, A. (2001). JACK intelligent agents-summary of an agent infrastructure. In *5th International Conference on Autonomous Agents*.

Hurley, S., & Whitaker, R. M. (2002). An agent based approach to site selection for wireless networks. In *Proceedings of the 2002 ACM symposium on Applied Computing*. 574-577.

Jayatilleke, G. B., Padgham, L., & Winikoff, M. (2004). Towards a component-based development framework for agents. In *German Conference on Multiagent System Technologies*, 183-197.

Jiang, D. & Han, J. (2008). *Real time multi-agent decision making by simulated annealing*, Simulated Annealing, Cher Ming Tan (Ed.), InTech, Retrieved December 21, 2016, from: http://www.intechopen.com/books/simulated_annealing/real_time_multiagent_decision_making_by_simulated_annealing

Kaplanoğlu, V., Şahin, C., Baykasoğlu, A., Erol, R., Ekinci, A., & Demirtaş, M. (2015). A multi-agent-based approach to dynamic scheduling of machines and automated guided vehicles (agv) in manufacturing systems by considering agv breakdowns. *International Journal of Engineering Research & Innovation*, 32.

Kuhn, J. R., Courtney, J. F., Morris, B., & Tatara, E. R. (2010). Agent-based analysis and simulation of the consumer airline market share for Frontier Airlines. *Knowledge-Based Systems*, *23*(8), 875-882.

Laporte, G., & Nobert, Y. (1987). Exact algorithms for the vehicle routing problem. *North-Holland Mathematics Studies*, *132*, 147-184.

Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, *1*, 343-362.

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

Li, S., & Li, J. Z. (2009). A multi-agent-based hybrid framework for international marketing planning under uncertainty. *Intelligent Systems in Accounting, Finance and Management*, *16*(3), 231-254.

Lopez-Sanchez, M., Noria, X., Rodríguez, J. A., & Gilbert, N. (2005). Multi-agent based simulation of news digital markets. *International Journal of Computer Science & Applications*, *2*(1), 7-14.

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 Technologies*, *18*(1), 99-119.

Malleson, N., Heppenstall, A., & See, L. (2010). Crime reduction through simulation: An agent-based model of burglary. *Computers, Environment and Urban Systems*, *34*(3), 236-250.

Menges, F., Mishra, B., & Narzisi, G. (2008). Modeling and simulation of e-mail social networks: a new stochastic agent-based approach. In *Proceedings of the 40th Conference on Winter Simulation*, 2792-2800.

Myatt, D., Nasuto, S., & Bishop, J. (2006). Alternative recruitment strategies for stochastic diffusion search. *Artificial Life X, Bloomington USA*. Colorado: Bradford Publishing.

Nasuto, S. J., Dautenhahn, K., & Bishop, M. (1999). Communication as an emergent metaphor for neuronal operation. In *Computation for metaphors, analogy, and agents* (pp. 365-379). Berlin: Springer Berlin Heidelberg.

Padgham, L., & Winikoff, M. (2005). *Developing intelligent agent systems: A practical guide* (Vol. 13). Liverpool: John Wiley & Sons.

Pan, Y., & Shi, L. (2007). On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming, 110(3)*, 543–559

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

Pelta, D., Cruz, C., & Verdegay, J. L. (2009b). Simple control rules in a cooperative system for dynamic optimisation problems. *International Journal of General Systems*, *38*(7), 701-717.

Sahin, C., Demirtas, M., Erol, R., Baykasoğlu, A., & Kaplanoğlu, V. (2015). A multi-agent based approach to dynamic scheduling with flexible processing capabilities. *Journal of Intelligent Manufacturing*, 1-19.

Sghir, I., Hao, J. K., Jaafar, I. B., & Ghédira, K. (2015). A multi-agent based optimization method applied to the quadratic assignment problem. *Expert Systems with Applications*, *42*(23), 9252-9262.

Sörensen, K., & Glover, F. W. (2013). Metaheuristics. In *Encyclopedia of Operations Research and Management Science*, 960-970.

Sycara, K. P. (1998). Multiagent systems. *AI magazine*, *19*(2), 79.

Tanaka, S., Fujikuma, S., & Araki, M. (2009). An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling, 12(6),* 575–593.

Tang, K., Kumara, S. R., Yee, S. T., & Tew, J. (2004). Wireless-based dynamic optimization for load makeup using auction mechanism. In *IIE Annual Conference. Proceedings*, 1. Institute of Industrial Engineers-Publisher.

Tasgetiren, M. F., Liang, Y. C., Sevkli, M., & Gencyilmaz, G. (2006). Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem. *International Journal of Production Research,44(22),* 4737-4754.

Tesfatsion, L. (2002). Agent-based computational economics: Growing economies from the bottom up. *Artificial Life*, *8*(1), 55-82.

Tesfatsion, L., & Judd, K. L. (Eds.). (2006). *Handbook of Computational Economics: Agent-Based Computational Economics (Vol. 2).* Amsterdam: Elsevier.

Testa, J. W., Mock, K. J., Taylor, C., Koyuk, H., Coyle, J. R., & Waggoner, R. (2012). Agent-based modeling of the dynamics of mammal-eating killer whales and their prey. *Marine Ecology Progress Series*, *466*, 275-291.

Voos, H. (2009). *Agent-based distributed resource allocation in continuous dynamic systems*, 1-21. I-Tech Education and Publishing, Jan. 2009.

Wang, D., & Liu, S. (2010). An agent-based evolutionary search for dynamic traveling salesman problem. In *International Conference on Information Engineering Vol. 1*, 111-114.

Wang, X., & Tang, L. (2009). A population-based variable neighborhood search for the single machine total weighted tardiness problem. *Computers & Operations Research, 36(6),* 2105–2110.

Wang, S. J., Xi, L. F., & Zhou, B. H. (2008). FBS-enhanced agent-based dynamic scheduling in FMS. *Engineering Applications of 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 *IIE Annual Conference. Proceedings*, 1.

Wooldridge, M. (2001). Intelligent agents: The key concepts. In *ECCAI Advanced Course on Artificial Intelligence*, 3-43.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, *10*(02), 115-152.

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

Yan, Y., Yang, S., Wang, D., & Wang, D. (2010). Agent based evolutionary dynamic optimization. In *Agent-Based Evolutionary Search,* 97-116.

Zhou, R., Lee, H. P., & Nee, A. Y. (2008). Simulating the generic job shop as a multi-agent system. *International Journal of Intelligent Systems Technologies and Applications*, *4*(1-2), 5-33.