Two Agent-based Approaches for Solving Multi-objective Multi-constraint Optimization Problems

Wang Hui

(B.Eng (Computer Science and Engineering), SJTU)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2004

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my Supervisor Dr. Lau Hoong Chuin for his insightful advice, guidance and encouragement. Without his help, this thesis cannot be completed successfully.

I would also say thanks to Mr. Wan Wee Chong, Mr. Lim Ming Kwang, Mr. Song Hua Wei and Mr. Neo Kok Yong for their suggestion and help during my progress.

I would take this opportunity to express gratitude to all of my friends, including Ms. Li Xiao Chen, Mr. Chen Chao, Mr. Song Xu Yang, Ms. Hou Jing and Ms. Liu Chang. It is you who give me courage and love and help me out during the most difficult time in my life. Also thanks to School of Computing, NUS, for giving me an opportunity to study in my favorite field, Computational Logistic.

And finally to everyone who helped, supported, and kept my spirits and motivations high. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Name: Wang Hui

Degree: Master of Science

Dept: Computer Science

Thesis Title: Two Agent-based approaches for Solving Multi-objective Multi-constraint Optimization Problems

Abstract

In this thesis, we propose two different agent-based approaches, the Coarse Grained Agent System (CGAS) and Fine Grained Agent System (FGAS) to solve Multi-Objective Multi-Constraint Problems (MOMCP), which represent the nature of many real life problems. CGAS gives a generic agent-model for multi-objective multi-constraint problem, while FGAS caters more for distributed multi-objective multi-constraint problem like most multiagent systems. We apply our approaches to solve the Inventory Routing Problem with Time Window (IRPTW). Experimental results indicate CGAS achieves a much better results than previous work, while FGAS runs very fast, whose run time is about one-sixth of CGAS with solution quality less than 10% poorer, which is still better than previous work.

Keywords: Agent-based approach, Multi-objective multi-constraint problem, Multiagent system

Chapter 1

Introduction

Real world optimization problems are often plagued with multiple objectives and multiple constraints. In this thesis, we term such a problem as a *multi-objective multi-constraint problem* (MOMCP). A common methodology for modeling MOMCP is by first considering the basic factors and then adding more constraints that take into account other important factors progressively. For example, the Traveling Salesman Problem (TSP) is a classic problem with only one objective and one constraint (i.e. permutation or tour constraint), which has been solved effectively. By adding more constraints, we derive a rich collection of generalizations of TSP. Among them, the most common one is Vehicle Routing Problem with Time Windows (VRPTW), which is extended with optimal fleet size objective and time window constraint from the TSP. While VRPTW is an important problem in transportation, in a logistics problem today, another important factor that should often be considered is the inventory level. By considering inventory costs across multiple periods of VRPTW, it gives rise to the Inventory Routing Problem with Time Windows (IRPTW).

In this thesis, we study IRPTW as an instance of MOMCP. IRPTW is an important problem in logistics and supply chain management. It is concerned with a distribution system with multiple infinite suppliers, capacitated warehouses, capacitated retailers, identical capacitated vehicles and unit-sized items. The items are to be transported from the suppliers to the warehouses, and subsequently delivered to the customers by vehicles. Vehicles deliver items to multiple customers within their respective time windows. Given the customers' time-varying demand over a finite planning horizon, the objective of IRPTW is to find a solution so as to minimize the total operating cost, which consists of the inventory cost and transportation cost.

Much work in optimization literature tacitly assumes that there is only one objective for each problem. Thus, an intuitive method to solve MOMCP is transforming it into a single-objective problem and then taking advantage of the existing optimization methods. Some representative methods are single-weighted method, distance functions method and min-max method [Hans, 1988].

The optimization of a single objective problem can guarantee one good solution. However, a multi-objective problem usually has a set of good solutions, which is known as Pareto-optimal set [Ben-Tal, 1979]. In real world scenario, a designer may need more alternatives to make the decision. A method that can provide a set of good solutions simultaneously is needed, which motivates a batch of researchers focusing on this area. Evolution Algorithm distinguishes itself as an excellent candidate by its nature of searching multiple solutions in parallel.

A newly emerging optimization approach is the agent-based (or multiagent) approach, which is especially good at solving problems with a distributed nature.

Agent is a concept firstly introduced in AI community. Franklin and Graesser defined an *autonomous agent* as "a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." [Franklin and Graesser, 1996]. Using agent as a self-contained problem solving unit capable of autonomous, re-active and pro-active social behavior, agent-based computing is a promising approach to develop a complex computer system.

In the optimization research community, agent-based approaches have already been applied to solve distributed optimization problems. In the existing works, such as [Yokoo et al, 1998] and [Modi et al. 2003], each agent is in charge of one or more variables. Agents are ordered into a tree with constraints as edges. Then the backtracking algorithm is applied to assign a value or values to each agent, either synchronously or asynchronously.

The existing works achieved success in solving assignment problems, but failed to extend elegantly to routing or packing problems, which is the concern of this thesis. One major reason is that existing works fail to handle the case of multiple variables per agent and the objective of minimizing number of agents in the system. That leads to our work on Fine-grained agent system (FGAS) proposed in this thesis.

A second agent system, the Coarse-grained agent system (CGAS), is also proposed in this thesis. Two motivations are behind CGAS:

1) The traditional method of converting multi-objective problem to single-objective problem has another drawback, which is the correlation between various objectives and constraints of a multi-objective multi-constraint problem is hard to express generically in one objective function. Typically, the single-objective optimization algorithm has no insight of which objective it is improving during the search. Consequently, much redundancy is incurred when optimization in one objective is undone by the optimization of subsequent objectives. Therefore, in CGAS the problem is divided along its objectives into objective agents.

2) In FGAS, since all the variables are distributed among agents, the designer faces a major difficulty of agent coordination. CGAS avoids this problem by keeping each objective agent working on the entire solution.

Note here, the concern of this thesis is not how to work out a set of Pareto-optimal solutions, but rather we focus on how to efficiently find a quality solution with a given weight for each objective. The work about Pareto-optimal set is reviewed to give readers a full picture of the literature of multi-objective optimization. In both FGAS and CGAS the weights of objective are charged by each individual agent, which can be adjusted easily by the users during the solution evolution process.

The rest of this thesis is organized as follows. In Chapter 2, we provide formal definitions needed in this thesis. Chapter 3 reviews related works in the literature. Chapter 4 and 5 present our two agent-based approaches respectively. Chapter 6 and 7 demonstrate how these two approaches can be applied to Inventory Routing Problem with Time Windows (IRPTW). Chapter 8 presents the experiment results and provides comparison and analysis of two approaches. Chapter 9 concludes the thesis.

Chapter 2

Problem Definition

In this chapter, we present a formal definition of Multi-objective Multi-constraint Problem (MOMCP) as well as the Inventory Routing Problem with Time Windows (IRPTW).

2.1 Multi-Objective Multi-Constraint Problem (MOMCP)

MOMCP consists of a set of objectives subject to a set of inequality or equality constraints.

Mathematically, the problem can be formulated as follows **[Rao, S.S. 1991]**:

Maximize/Minimize $\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x}),\ldots, g_n(\boldsymbol{x})\}$

Subject to $\{c_1(\boldsymbol{x}),c_2(\boldsymbol{x}),\ldots, c_m(\boldsymbol{x})\}$

Where $c_i(x)$ $i\in[1,m]$ takes the form of $c_i(x)\leq 0$ or $c_i(x)= 0$

Solutions to a multi-objective optimization problem are mathematically expressed in terms of non-dominated points, or Pareto-optimal solutions [Ben-Tal, 1979]. The Pareto-optimal set consists of all those solutions such that their components cannot be all simultaneously improved.

Formally, a solution $\boldsymbol{x}$ is *Pareto-optimal* for a multi-objective minimizing problem if and only if there exists no solution x' such that $f_i(x')\leq f_i(\boldsymbol{x})$ for i=1,2,..,n with $f_j(x')< f_j(\boldsymbol{x})$ for at least one j. If it is a maximizing problem, the "≤" and "<" in the definition should be replaced by "≥" and ">", respectively.

2.2 Inventory Routing Problem with Time Windows (IRPTW)

The Inventory Routing Problem with Time Windows is an extension of Vehicle Routing Problem with Time Windows. Given a set of capacitated warehouse, a set of capacitated customers and a set of identical capacitated vehicles, the vehicles will deliver the unit-sized items from warehouse to customers within their service time window. The objective of the problem is to work out a transportation plan and a distribution plan over a finite time period such that the total operating cost is minimized. The total operating cost consists of inventory cost, backlogging cost and transportation cost.

Stated formally [Lau et al. 2000]:

An IRPTW instance consists of the following inputs,

C: set of customers;

T: consecutive days in the planning period $\{1,2,\ldots,n\}$;

d_{it}: demand of customer i on day t;

q_v: vehicle capacity;

q_w: storage capacity of customer i;

W_{it}: time window of customer i on day t;

c_i^h: holding cost per unit item per day at customer i;

c_i^b: backlog cost per unit item per day at customer i;

The outputs are as follows:

1. the distribution plan, which is denoted by x_{it}; integral flow amount from the warehouse to customer i on day t;

2. the set of daily transportation routes Φ, which carry the flow amounts in 1 from the warehouse to the customers such that the sum of the following linear costs is minimized;

   a) Inventory cost at the warehouse

   b) Inventory cost at the customer

   c) Backlog cost

   d) Transportation cost from the warehouse to the customers

We use indices i and t for customers and days respectively.

The distribution plan must obey the demands and storage capacity constraints, and the transportation routes must obey the standard routing, vehicle capacities and time windows constraints.

Chapter 3

Literature Review

The multi-objective multi-constraint problem is well-studied in the literature. All the past works fall into two categories:

1) Converting a multi-objective problem to a single-objective problem and then applying the single objective optimization method which is rich in literature.

2) Treating different objectives separately and generating a set of optimal results, which is called Pareto-optimal set.

We look at several classic methodologies in sections 3.1 and 3.2. In section 3.3 we review the multiagent systems for optimization.

3.1Converting Multi-Objective Problem to Single-Objective Problem

The methods in this categories [Hajela and Lin 1992] use different formulas to integrate multiple objectives into one objective function. Then a single objective optimization algorithm is applied.

3.1.1 Single Weighted Objective Method

In this method, the idea is to project all the objectives onto a single objective function by adding a weight to each objective. The weight of an objective indicates the importance of the objective.

Mathematically, the new function is written as:

$$g(\overline{x}) = \sum_{i=1}^{k} w_i g_i(\overline{x})$$

where $0 \leq w_i \leq 1$ and $\sum_{i=1}^{k} w_i = 1.$

Since the coefficients of the objective function are determined statically, one cannot guarantee whether the chosen weights can reflect the importance of each original goal appropriately.

3.1.2 Distance Functions Method

In this method, the single objective function derived from multiple objectives is as follows:

$$Z(x) = [\sum_{i=1}^{N} |g_i(x) - \overline{y}_i|^r]^{1/r}, \qquad 1 \leq r \leq \infty,$$

where $x \in X$ (the feasible region)

$\overline{y}$ is a demand-level vector which has to be specified by the decision maker.

Typically, a Euclidean metric r=2 is chosen. [Hans, 1988].

There are two disadvantages to this method:

1) The ideal solution should be known, otherwise a demand level is assumed.

2) If the wrong demand level is chosen, the result will be non-Pareto-optimal solutions.

3.1.3 Min-Max Formulation Method

This method assumes every objective function is equally important. It attempts to minimize the relative deviations of the single objective functions from individual optimum. That is, it tries to minimize the objective conflict.

For a minimization problem, the corresponding min-max problem is formulated as follows:

$$\text{Minimize } F(x) = \max\,[Z_j(x)],\ j=1,2,\ldots, N$$

where $x \in \mathrm{X}$ (the feasible region) ;

$Z_j(x)$ is calculated for a nonnegative target optimal value $\bar{g}_j(x) > 0$ as follows:

$$Z_j(x) = \frac{g_j(x) - \bar{g}_j(x)}{\bar{g}_j(x)},\ j = 1,2,...,N.$$

This method can be modified to include priority of each objective by introducing dimensionless weights in the formulation.

3.2 Treating Objectives Separately

Usually, the Pareto-optimal set consists of more than one solution. However, single – objective method can provide only one Pareto-optimal solution each time. Moreover, these methods become expensive as they call for the designer's thorough understanding of the relative importance of objectives before optimization. Those reasons motivate researchers to find a method which can provide a set of Pareto-optimal solutions without exact knowledge of objective weights.

Evolutionary algorithms (EAs) is recognized to be well-suited to multi-objective optimization. Multiple individuals can search for multiple solutions in parallel, eventually derived a set of good solutions. The ability to handle complex problems, involving features such as discontinuities, disjoint feasible spaces and noisy function evaluations, reinforces the potential effectiveness of EAs in multi-objective optimization.

The research of EA mainly concerns with three aspects, fitness assignment, diversity preservation, and elitism.

The first studies on multi-objective evolutionary algorithms (MOEAs) were mainly concerned with the problem of guiding the search towards the Pareto-optimal set. In contrast to single-objective optimization, where objective function and fitness function are often identical, both fitness assignment and selection must allow for several objectives with multi-criteria optimization problems. Criterion-based and Pareto-based fitness assignment strategies have been studied in the history.

Criterion-based methods switch between the objectives during the selection phase. Each time an individual is chosen for reproduction, potentially a different objective will decide which member of the population will be copied into the mating pool.

For example, [Schaffer 1985] proposed filling equal portions of the mating pool according to the distinct objectives, while [Kursawe 1991] suggested assigning a probability to each objective which determines whether the objective will be the sorting criterion in the next selection step -- the probabilities can be user-defined or chosen randomly over time.

The idea of calculating an individual's fitness based on Pareto dominance was proposed in [Goldberg 1989]. Some approaches, such as [Fonseca and Fleming 1993], use the dominance rank, i.e. the number of individuals by which an individual is dominated, to determine the fitness values. Others make use of the dominance depth. An example is [Srinivas and Ddb 1994]. Alternatively, also the dominance count, i.e., the number of individuals dominated by a certain individual, can be taken into account. For instance, [Zitzler and Thiele 1999] and [Zitzler et al. 2001] assign fitness values on the basis of both dominance rank and count.

Maintaining diversity along the current approximation of the Pareto set is the second important issue. Most work incorporates density information into the selection process: an individual's chance of being selected is decreased when the density of individuals in its neighborhood is greater. Some representative methods include Kernel methods, Nearest neighbor techniques and Histograms [Silverman 1986]. Kernel methods define the neighborhood of a point in terms of a so-called Kernel function K which takes the distance to another point as an argument. Nearest neighbor techniques take the distance of a given point to its k^{th} nearest neighbor into account in order to estimate the density in its neighborhood. Histograms define a third category of density estimators that use a hypergrid to define neighborhoods within the space.

Although fitness assignment and diversity preservation techniques aim at guiding the population towards the Pareto-optimal set, still good solutions may be lost during the optimization process due to random nature. A common way to deal with this problem is to maintain a secondary population, the so-called *archive*, to record the promising solutions in the population.

Due to memory and run-time limitations, the size of *archive* is limited. Therefore, criteria have to be defined for which the solutions should be kept in the *archive*. The dominance criterion is most commonly used, i.e., dominated archive members are removed and the archive comprises only the current approximation of the Pareto-optimal set. However, as this criterion is in general not sufficient, additional information is taken into account to reduce the number of archive members further. Examples are density information [Zitzler and Thiele 1999] and [Knowles and Corne 1999] and the time that has been passed since the individual entered the archive [Rudolph and Agapie 2000].

It should be mentioned that not all elitist MOEAs explicitly incorporate an archive, e.g., NSGA-II [Deb et al. 2000]. However, the basic principle is the same: during environmental selection special care is taken to not loose non-dominated solutions.

3.3 Agent-Based Approaches

3.3.1 Overview

Recently, the agent-based approach has become popular in various branches of computer science. Multiagent system can naturally represent the decentralized nature of the problem, the multiple location control, the multiple perspectives, or the competing interests. Moreover, a multiagent system is good at expressing the interaction between agents, either to achieve their individual objectives or to manage the dependencies coming from existing in a common environment. These interactions range from simple semantic interoperation (the ability to exchange comprehensible communications), through traditional client-server type interactions (the ability to request that a particular action is performed), to rich social interactions (the ability to cooperate, coordinated and negotiate about a course of action).

Multiagent systems research brings together a diverse set of research disciplines and thus there is a wide range of ideas currently being explored. In our research, we focus on the multiagent systems applications on optimization/constraint satisfaction, since we are dealing with multi-objective multi-constraint optimization problem. Typically, in MAS for optimization each agent represents a partial solution.

3.3.2 Agent Models

Several agent models in the literature have been proposed to study the optimization problem.

The most widely-used *Belief-Desire-Intention* (*BDI*) agent model was proposed in [Rao and Georgeff 1995]. In that work, *Belief* represents the state of the environment that a BDI agent roams. *Desire* represents the motivation of agents. *Intention* represents the course of action taken by an agent. However, this work only explained the general principles for constructing an agent system, while does not give any indication for multiagent interaction.

The interaction issue is first studied by Chainbi in [Chainbi et al. 2001]. A *Belief-Goal-Role* (*BGR*) agent model is proposed. In *BGR* model, agents can have different local goals and roles in achieving a common global goal.

[Liu and Tang 2002] proposed an *Environment-Reactive rules-Agents* (*ERA*) agent model for tackling constraint satisfaction problems. Competitive collaboration among agents and “survival-of-the fittest” principle is discussed in that work.

3.3.3 Algorithms for DCSP and DCOP

Definitions of DCSP and DCOP

Two classic problems are highly related to MAS for optimization. One is Distributed Constraint Optimization Problem (DCOP), the other is Distributed Constraint Satisfaction Problem (DCSP). DCSP techniques have been used for coordination and conflict resolution in many multiagent applications. DCOP extends from DCSP by further considering optimization of the objective.

Here are the definitions of these two problems:

DCSP

A Constraint Satisfaction Problem (*CSP*) is defined by a set of *n* variables, $X=\{x_1,\ldots,x_n\}$,each element associated with value domains $D_1,\ldots, D_n$ respectively, and

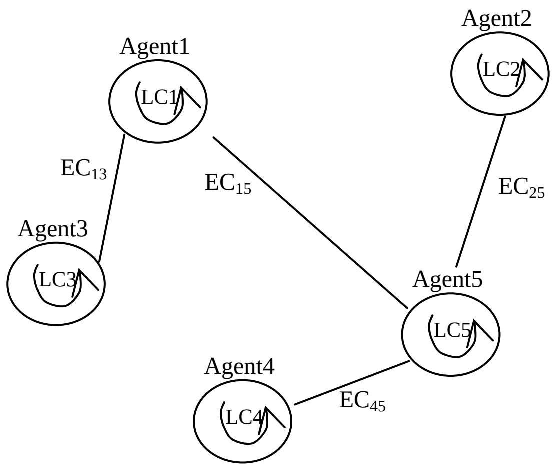


Figure 3.1 Agents and Constraints in DCSP and DCOP

a set of k constraints, $C=\{C_1, \ldots, C_k\}$. A solution in CSP is the value assignment for the variables which satisfies all the constraints in C.

A DCSP is a CSP in which variables and constraints are distributed among multiple agents. Formally, there is a set of m agents, $A_g=\{A_1, \ldots, A_m\}$. Each variable (x_i) belongs to an agent A_j. There are two types of constraints based on whether variables in a constraint belong to a single agent or not:

- For a constraint $C_r \in C$, if all the variables in C_r belong to a single agent $A_j \in A_g$, it is called a *local constraint (LC).*

- For a constraint $C_r \in C$, if variables in C_r belong to different agents in A_g, it is called an *external constraint (EC).*

Figure 3.1 illustrates an example of a DCSP. Each agent can have multiple variables. There is no limitation on the number of local/external constraints for each agent. Solving a DCSP requires that agents not only satisfy their local constraints, but also communicate with other agents to satisfy external constraints.

DCOP

A Distributed Constraint Optimization Problem (DCOP) consists of n variables $X=\{x_1,\ldots,x_n\}$ each assigned to an agent, where the values of the variables are taken from finite, discrete domains $D_1,\ldots, D_n$ respectively. Only the agent who is assigned a variable has control of its value and knowledge of its domain. The goal is to choose values for variables such that a given objective function is minimized or maximized. The objective function is described as an aggregation over a set of cost functions, or valued constraints.

Algorithms for DCSP & DCOP

Asynchronous Weak Commitment (AWC) Algorithm

AWC search algorithm is known to be the best published DCSP algorithm [Yokoo et al. 1998]. One limitation of the Asynchronous Backtracking (ABT) Algorithm [Yokoo et al. 1992], a famous DCSP algorithm before AWC, is that the agent/variable ordering is statically determined. If the value selection of a higher priority agent is bad, the lower priority agents need to perform an exhaustive search to revise the bad decision. In AWC search algorithm, the min-conflict heuristic is introduced to reduce the risk of making bad decisions. Furthermore, the agent ordering is dynamically changed so that a bad decision can be revised without performing an exhaustive search.

To simplify the description of the algorithm, suppose that each agent has exactly one variable and the constraints between variables are binary. We represent a DCSP as a network, where variables are nodes and constraints are links between nodes. For example, in Figure 3.2 there are three agents, x_1,x_2,x_3, with variable domains {1, 2}, {2}, {1, 2} respectively, and constraints $x_1 \neq x_3$ and $x_2 \neq x_3$.

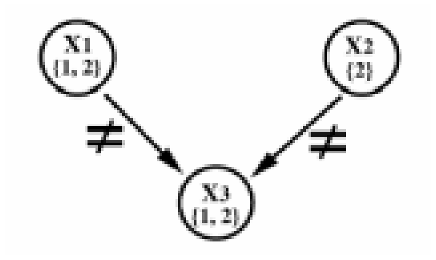


Figure 3.2 Agents in AWC

In AWC, agents asynchronously assign values to variables, and communicating the values to neighboring agents with shared constraints. Each variable has a non-negative integer priority. When the value of an agent's variable is not consistent with the values of its neighboring agents' variables, there can be two cases:

(i) a *good* case where there exists a consistent value in the variable's domain;

(ii) a *nogood* case that lacks a consistent value.

In the *good* case with one or more value choices available, an agent selects a value that minimizes the number of conflicts with lower priority agents, which is known as min-conflict heuristic. On the other hand, in the *nogood* case, an agent increases its priority to *max*+1, where *max* is the highest priority of its neighboring agents, and selects a new value that minimizes the number of conflicts with all of its neighboring agents. This priority increase makes previously higher agents select new values. Agents avoid the infinite cycle of selecting non-solution values by saving the *nogood* situations.

When a value assignment in which every variable is consistent is found, we got the solution. The proof of the algorithm completeness can be also found in [Yokoo et al. 1998].

Asynchronous Distributed Optimization (*Adopt*)

Adopt [Modi et al. 2003] is the first algorithm for DCOP that can find either an optimal solution or a solution within a user-specified distance from the optimal, using only localized asynchronous communication and polynomial space at each agent.

In *Adopt*, each agent is in charge of a single variable. Agents are prioritized in a Depth-First Search (DFS) tree in which each agent has a single *parent* and multiple *children.* Two agents are neighbors if they have a constraint between them. Constraints are allowed between an agent and any of its ancestors or descendents, but there can be no constraints between nodes in different subtrees of the DFS tree (See Figure 3.3)

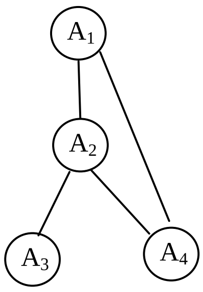


Figure 3.3 DFS tree in *Adopt*

The global objective function in Adopt has been modeled as *valued constraints*, that is, constraints that are described as functions that returned a range of values.

Adopt performs *distributed backtrack* search using an "opportunistic" best-first search strategy. Each agent always chooses the variable value with smallest lower bound. Using "opportunistic" best-first search strategy, we may abandon partial solutions before they have proved the solution is definitely suboptimal. Therefore, reconstruction of a previously explored solution is needed. The idea of *Adopt* is using a stored lower bound as a backtrack threshold. The program will stop when the interval between agent's stored lower bound and upper bound meet the user's requirement.

Details of *Adopt* can be seen in Figure 3.4.

```
# Currentvw: Current view of linked ancestors' values
# $x_i/d_i$:        Local variable/value
# $c(d)$:           Current lower bound on cost for subtree rooted at child, given $x_i$ chooses
                    value $d$
```

```
# proc  Initialize:
        Currentvw ← { }; $d_i$ ←null; threshold ← 0
        for all d in $D_i$:
        c(d) ←0;
        context(d) ←{}
```

```
# proc  Hill_climb:
        for all d in $D_i$:
                e(d) ← $\delta$ ($x_i$, Currentvw U{($x_i$,d)}) +c(d);
        choose d that minimizes e(d)
                prefer current value $d_i$ in case of tie;
        if e($d_i$)>threshold
                $d_i$←d;
        childLimit←max(c($d_i$),threshold- $\delta$ ($x_i$, Currentvw $\cup${($x_i$, $d_i$)}))
        SEND VALUE (($x_i$, $d_i$), childLimit) to descendents
        # only choose variables relevant to local cost
        Neighborvw = {($x_j$, $d_j$) in Currentvw | $x_j$ neighbor of $x_i$}
        viewContext←Neighborvw $\cup${union of contexts of $d_i$ in D}
        # to preserve completeness – VIEW is for best value d, not current value $d_i$
        SEND VIEW, viewContext, e(d)) to parent;
```

```
# proc when_received_value ($x_j$,$d_j$,limit):
         update Currentvw with ($x_j$,$d_j$)
                for all d in $D_i$
                if(context(d) incompatible with Currentvw)
                        c(d) ←0;
                        context(d) ← {}
                if($x_j$ is parent)
                        threshold ←limit
        go to Hill_climb;
```

```
# proc when_received_view(vw,cost):
 d←value of $x_i$ in vw
 if vw contains ($x_i$, d) //child is my neighbor
        remove ($x_i$,d) from vw;
 if vw compatible with Currentvw and cost >c(d) then
        c(d) ←cost;
        context(d) ← vw;

else  //child is not my neighbor
        for all d' in $D_i$ :
                if vw compatible with Currentvw and cost >c(d') then
                        c(d') ←cost;
                        context(d') ←vw;
 end if;
 if( c($d_i$) changed) then
        go to Hill_climb;
```

Figure 3.4 Algorithm of Adopt

Handling multiple local variables

The major disadvantage of the above algorithms is that each agent has only one local variable. There are two major ideas of extending the method to apply to the situation where one agent has multiple local variables.

Method 1: each agent finds all solutions to its local problem first. By finding all solutions, the given problem can be re-formalized as a DCSP, in which each agent has one local variable whose domain is a set of obtained local solution. Then, agents can apply algorithms for the case of a single local variable. The drawback of this method is that when a local problem becomes large and complex, finding all the solutions of a local problem becomes virtually impossible. A work employing this idea is [Armstrong and Durfee 1997]

Method 2: an agent creates multiple virtual agents, each of which corresponds to one local variable, and simulates the activities of these virtual agents.

For example, if agent *k* has two local variables x_i,x_j, we assume that there exist two virtual agents, each of which corresponds to either x_i or x_j. Then, agent *k* simulates the concurrent activities of these two virtual agents. In this case, each agent does not have to predetermine all the local solutions. However, since communicating with other agents is usually more expensive than performing local computations, it is wasteful to simulate the activities of multiple virtual agents without distinguishing the communications between virtual agents within a single real agent, and the communications between real agents. An example of this idea is [Yokoo and Hirayama 1998].

3.3.4 Cooperativeness-Based Strategies (CBS)

Although AWC is one of the most efficient DCSP algorithms, real time and dynamism in multiagent domains demands very fast conflict resolution.

While AWC relies on the min-conflict heuristic that minimizes conflicts with other agents, the CBS [Jung et al. 2001], enhanced by local constraint communication, consider how much flexibility (choice of values) is given towards other agents by a selected value. By considering neighboring agents' local constraints, an agent can generate a more locally cooperative response, potentially leading to faster conflict resolution convergence.

The local cooperativeness goes beyond merely satisfying constraints of neighboring agents to accelerate convergence. That is, an agent A_i cooperates with a neighbor agent A_j by selecting a value for its variable that not only satisfies the constraint with A_j, but also maximized A_j's flexibility. Then A_j has more choices for a value that satisfies A_j's local constraints and other external constraints with its neighboring agents, which can lead to faster convergence. To elaborate this notion of local cooperativeness, the followings definition was given,

Definition 1: For a value $v \in D_i$ and a set of agents $N_i^{sub} \subseteq N_i$ (a set of neighboring agents), a flexibility function is defined as $f(v, N_i^{sub}) = \sum_j c(v, A_j)$ such that $A_j \in N_i^{sub}$ and $c(v, A_j)$ is the number of values of A_j that are consistent with *v*.

Definition 2: For a value *v* of A_i, *local cooperativeness* of *v* is defined as $f(v, N_j)$. That is, the local cooperativeness of *v* measures how much flexibility is given to all of A_i's neighbors by *v*.

As an example of the flexibility function $f(v, N_i^{sub})$, suppose agent A_1 has two neighboring agents A_2 and A_3, where a value v leaves 70 consistent values to A_2 and 40 to A_3 while another value v' leaves 50 consistent values to A_2 and 49 to A_3. Now, assuming that values are ranked based on flexibility, an agent will prefer v to v': $f(v,\{A_2,A_3\})=110$ and $f(v',\{A_2, A_3\})=99$. These definitions of flexibility function and local cooperativeness are applied for the cooperative strategies defined as follows:

- S_{basic}: Each agent A_i selects a value based on min-conflict heuristics(the original strategy in the AWC algorithm);
- S_{high}: Each agent A_i attempts to give maximum flexibility towards its higher priority neighbors by selecting a value v that maximum $f(v, N_i^{high})$;
- S_{low}: Each agent A_i attempts to give maximum flexibility towards its lower priority neighbors by selecting a value v that maximizes $f(v, N_i^{low})$;
- S_{all}: each agent A_i selects a value v that maximizes $f(v, N_j)$, i.e. max flexibility to all neighbors.

These four strategies can be applied to both the *good* and *nogood* cases. In the *nogood* case, neighboring agents are grouped into higher and lower agents based on the priorities before the priority increase in AWC. Therefore, there are sixteen strategy combinations for each flexibility base.

Furthermore, in [Jung and Tambe 2003], flexibility function was extended to accommodate different types of possible local cooperativeness:

Definition 3: For a value $v \in D_i$ and a set of agents $N_i^{sub} \subseteq N_i$ (a set of neighboring agents), a flexibility function is defined as $f^{\oplus}(v, N_i^{sub}) = \oplus(c(v, A_j))$ where (i) $A_j \in N_i^{sub}$; (ii) $c(v, A_j)$ is the number of values of A_j that are consistent with v and (iii) $\oplus$ referred to as a *flexibility base*, can be *sum*, *min, max, product*, etc.

Chapter 4

Fine-Grained Agent System (FGAS)

This chapter explains how FGAS works.

Typically a multiagent system (MAS) exhibits the following merits [Stone and Veloso 1997]:

1. A MAS can naturally represent the decentralized nature of the problem and is good at expressing the complex interaction between agents;

2. The system is dynamic. For real-time practical applications, there are often unexpected changes to the problem instances over time. Since the agents are inherently modular and autonomous, it is easy for agents to enter and leave this system without affecting operations which have already been performed;

3. Rather than tackling the whole problem with a centralized agent, the computation task is split and assigned to each agent. Consequently, the computation complexity and programming difficulty are both reduced;

4. Computation efficiency is brought by concurrent computing. By keeping agents working in parallel, the running time is greatly reduced.

FGAS, inheriting all the merits from a typical MAS, is proposed in this thesis to solve MOMCP.

4.1 Main Features of FGAS

In a typical MAS used for optimization, each variable belongs to an agent. There are two types of constraints, local constraint (LC) and external constraint (EC). The agents will work through communication and negotiation to optimize the objectives of the problem.

Two major difficulties make the existing MAS for optimization inadequate to solve routing and packing problems:

1) *They fail to handle multiple variables per agent*. Although the definition states that each agent can have multiple variables, most existing works assume each agent only takes responsibility for one variable. While those methods perform satisfactorily on the class of assignment problems, they fail to extend to the packing or routing problem, which usually has multiple variables per agent. There are also some attempts to extend these works to multiple variables per agent [Armstrong and Durfee 1997] and [Yokoo and Hirayama 1998]. Unfortunately, they are found to be neither effective nor scalable to large-scale problems.

2) *They cannot model the objective of minimizing number of entities in the system*. In routing problems the entities are vehicles, while in packing problems the entities are bins. In the existing MAS for optimization, agents always represent variables. Therefore, they never consider the issue of agents' number in the system.

FGAS is extended to handle the case of multiple variables per agent. With intelligent agents in the system, FGAS is capable of solving routing and packing problems.

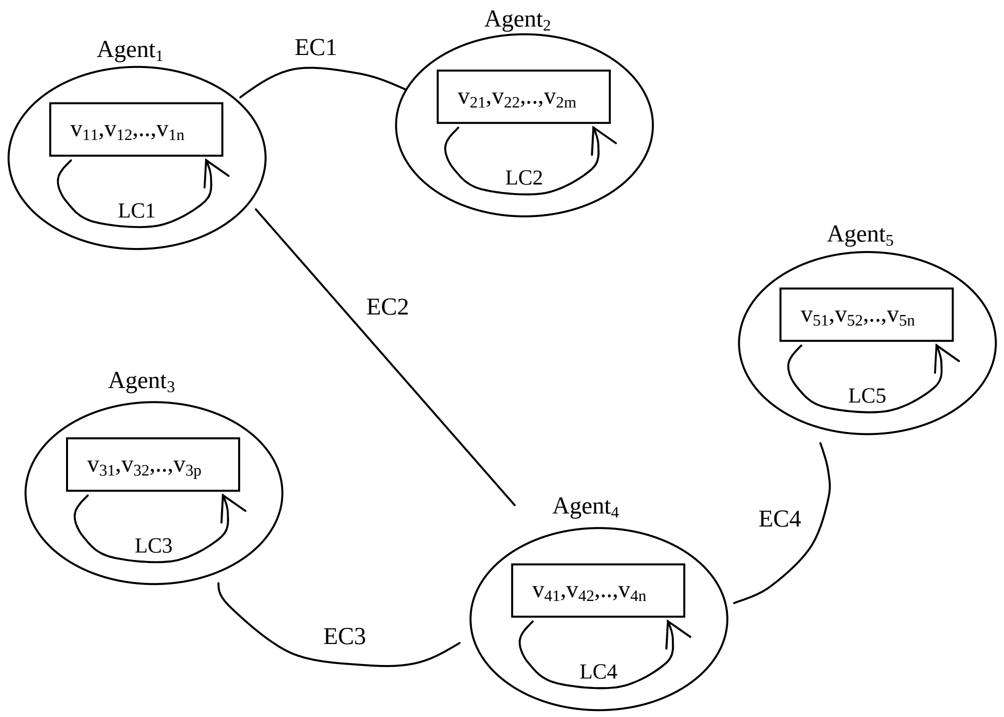
4.1.1. Multiple variables per agent

The main reason why existing work cannot be applied to the multiple variables per agent is that the conflict resolution strategies employed require the knowledge of all possible alternative assignments or states of the agent.

FGAS solves this problem by adopting the local-search idea which has been successfully used to solve large-scale optimization problems [Lao 2002]. Instead of listing all the alternatives, FGAS only proposes a move list based on the predefined moves. Thus, agents in FGAS are able to host multiple variables (Figure 4.1).

4.1.2. Characteristics of an agent

In FGAS, agents represent meaningful entities. They are cooperative to achieve good overall objective. Different from existing work, agents in FGAS are endowed with



Agent$_1$
$v_{11},v_{12},..,v_{1n}$
LC1
EC1
Agent$_2$
$v_{21},v_{22},..,v_{2m}$
LC2
EC2
Agent$_5$
$v_{51},v_{52},..,v_{5n}$
LC5
Agent$_3$
$v_{31},v_{32},..,v_{3p}$
LC3
Agent$_4$
$v_{41},v_{42},..,v_{4n}$
LC4
EC3
EC4

Figure 4.1 Agents in FGAS and the constraints between and within them.

more intelligence. They have their own opinion towards guiding the solution improvement. When a change contradicts with their opinion, agents tend to be selfish but still willing to make compromise.

1) *Intelligence*

Agents in FGAS are able to detect or discover trends for getting good or bad future solutions from their local information, including the status of itself and suggestion from constraint agent. Such future expectation will be converted to their desires of encouraging or discouraging the corresponding changes. With such desires, agents are able to handle a class of objectives which is usually affected by a series of changes. We termed such objective as *estimated objective*, which we will explain later.

> Example: In VRPTW, a single relocate usually cannot reduce one vehicle. But several relocates together can lead to the reduction of one vehicle. In this case, if an agent detects it stands a chance of reducing one vehicle, it will convert it into its desire of encouraging future "relocate".

The constraint agent mentioned above forms the second type of agents in FGAS. The degree of satisfying or violating a constraint provides useful information for agents' state changing.

In FGAS, there are several types of agents. We have seen the need for a constraint agent above. Henceforth, to minimize confusion, we use the term *subagent* to refer to the standard agent in a multiagent system.

2) *Selfishness*

Subagents are selfish in the sense that they are reluctant to accept the changes that contradict with their desires. The desires are quantified into a value, called the *desire value*. The *desire value* describes a subagent's opinion of a change's quality and indicates how much the subagents desire this change. In FGAS, a *desire value* of agent consists of three parts:

(a) the change of overall objective value, b) its future expectation of getting good or bad result, and (c) the objective balance, which will be explained in section 4.3.3 in detail.

Introducing selfishness of subagents speeds up the solution convergence.

3) *Cooperativeness*

When the desires of two selfish subagents conflict, the subagents tend to be cooperative. Selecting the right conflict resolution strategy is an important issue in MAS for optimization. Extending the idea of generating a locally cooperative response from cooperativeness-based strategy (CBS) [Jung and Tambe 2003], agents in FGAS negotiates based on the *desire value*.

4.1.3. Agent Coordination

Drawing inspiration from natural systems in which complex global structures and behaviors result from local interactions among many simple elements, we proposed a physics-motivated [Shehory et al. 1999] coordination mechanism for FGAS that treats subagents as randomly moving, locally interacting entities. The similar scheme has already successfully used in the area of coalition formation for Large-Scale Electronic Markets [Lerman and Shehory 2000].

Note that in E-markets there exist few external constraints, the agents can randomly move and interact. However, in our scenario, there may exist some external constraints which limit the concurrent computation. For example, if the change of subagent *a* will affect the local knowledge of subagent *b*, then they cannot be computed concurrently. Otherwise, the *b* will make wrong decision based on the obsolete information. Therefore, we introduce FORBID function to describe the external constraints between subagents which limit the concurrent computation and the *arbitration agent* to prevent the situation of error arising from concurrent computation with inter-dependent agents.

4.1.4. Dynamic System

The number of subagents in FGAS can be dynamic. When the new demand or request comes, a new subagent can be generated and added to FGAS. When an objective is to minimize number of agents, the existing agents will try to achieve this objective by destroying itself. This feature enables FGAS to model the objective of minimizing number of agents, which is an important objective of packing or routing problem.

4.1.5. Handling Multiple Objectives

In MOMCP, a fundamental issue is how to handle multiple objectives. There are four kinds of objectives:

1) *local computable objective*: objectives that can be exactly computed with local knowledge. This kind of objective is delegated to subagents.

2) *local estimated objective*: objectives which usually cannot be affected by one move. However, a move's effect can be estimated using local knowledge. This kind of objective is delegated to subagents.

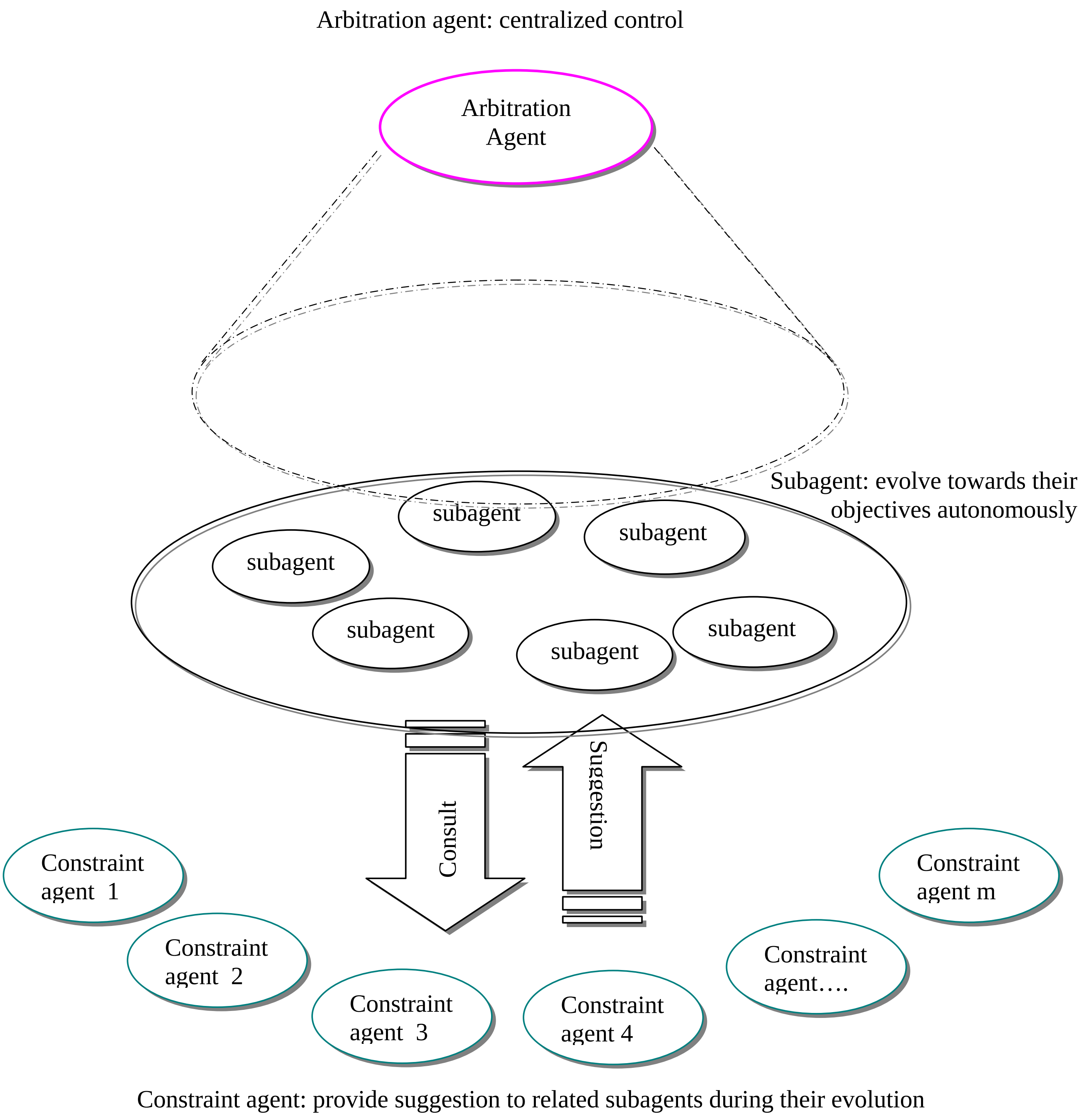


Figure 4.2 Relationships between the agents

3) *global computable objective*: objectives that can be only computed with global knowledge. The *arbitration agent* (*AA*) is used to handle this objective.

4) *global estimated objective*: objectives that can be only estimated with global knowledge. The *arbitration agent* (*AA*) is used to handle this objective.

A situation which should be prevented is the over optimization of one objective at the expense of another. In FGAS, a threshold is set for each objective such that when the value in one objective decreases beyond a certain threshold, that change will be only accepted with only small probability.

Using FGAS, objectives of the problem is either decomposed to subagents or handled by the arbitration agent. We will explain the details in Section 4.3.3 and the example of IRPTW is given in Section 6.1.2.

The relationships between subagents, arbitration agent and constraint agents are illustrated in Figure 4.2.

4.2 Formulation

Stated formally, we are given an n-objective problem Q, the goal is to

$$\text{Maximize/Minimize } g_i(x) \ i \in [1,n] \ \text{ s.t. } C = \{c_1, c_2, ..., c_m\}$$

where g_i is one objective of Q, $i \in [1,n]$

$\boldsymbol{x}$ is the vector of decision variables

C is the constraint set consisting of constraints c_1 to c_m

Adopting the idea in [Chainbi et al. 2001], agents in FGAS interact with each other. They have different roles and local goals. We define:

A team of agents $\alpha =< AA, SA, CA, FORBID >$, in which

- *AA* is an *arbitration agent* that provides centralized control. This agent may not be necessary for every problem. The arrangement of *AA* enables FGAS to solve more problems.

- *SA* is a set of *subagents*, which operate interactively to change their states and finally reach an "optimal" solution for the problem.

  $SA = \{sA_1, sA_2, ..., sA_p\}$

  where p is the number of subagents

  $sA_i =< X_i, G_i, RC_i, D_i, Sta_i >$

  X_i is the set of variables in sA_i

  G_i is the objectives of sA_i

  RC_i returns the set of constraint agents related to sA_i

  D_i is the desire value of sA_i

  Sta_i is the state of sA_i

  If Sta_i =A (i.e. active) sA_i is changing its state

  If Sta_i =NA (i.e. non-active) sA_i is not changing its state

$X_1 \cup X_2 \cup ... \cup X_p = S$ and $X_i \cap X_j = \Phi \;\; i, j \in [1, p]$, *S* is the solution of *Q*.

- *CA* is a set of *constraint agents*. cA_i representing constraints c_i of the problem. $CA = \{cA_1, cA_2, ..., cA_m\}$

- *FORBID* is function of subagent pairs, defined as follows,

  If FORBID(sA_i, sA_j)=TRUE, then at least one of Sta_i and Sta_j should be in the state of NA. This function describes the external constraints, which limit the concurrent computation, between the subagents, as we mentioned in Section 4.1.

4.3 Agent Functionality

In this section, we provide details of the functionality of each kind of agent. Recall that there are three types of agents, i.e. Arbitration Agent *AA*, Constraint Agent *CA* and subagent *SA*.

4.3.1 Arbitration Agent *AA*

AA is a virtual agent in FGAS. It is not necessary for every problem. But with *AA*, FGAS is able to solve problems which cannot be solved by traditional MAS.

Example. Without *AA*, the FGAS can successfully solve VRPTW but fail to handle IRPTW, because the inventory cost across the planning period makes some subagents cannot compute concurrently.

As previously mentioned, *AA* will monitor the subagents' work and ensure that no agents pair which makes FORBID return true is computed simultaneously.

AA also takes the responsibility of handling *global objective.*

Example. In IRPTW, the value of the inventory cost cannot be computed with only local knowledge by the subagents. In this case, *AA* is required to participate in every interaction of subagents to compute that objective.

4.3.2 Constraint agent *CA*

Most works in the literature solve MOMCP based on optimizing objectives while ignoring the contribution of constraints. We observe that from examining the degree of violating or satisfying constraints, subagents can get some insights into the future solution quality, which is very important for handling *estimated objectives*.

Example. In VRPTW, if a vehicle capacity is 200 and its load will become 5 after some action, such action will probably be accepted since it will probably realize the objective of minimizing number of vehicle. The constraint agent will return a value containing such information to the subagent.

4.3.3 Subagents *SA*

Subagents form the backbone of FGAS. Each subagent has only a local view of the environment. Through communication, negotiation and cooperation, they work together to optimize objectives of the problem.

Subagent and constraint agent

For each $sA_i \in SA$, $RC_i(sA_i) = CA_i \subseteq CA$ are the constraint agents related to sA_i.

Besides optimizing the local objective, the responsibility of satisfying constraints is also assigned to each individual subagent. To be more precise, each subagent knows all its related constraint agents. As mentioned in section 4.3.2, it will get important information from those constraint agents for handling the *estimated objectives*.

Objectives of subagent

The objectives of subagent can be described as

$$G_i = \{g_{1i}, g_{2i}, \ldots, g_{xi}\}, x \leq n$$

such that g_{ji} is the localized objective g_j of subagent sA_i, where n is the number of objectives.

Usually, the localized objectives are same as the overall objectives, while there are sometimes overall objectives should be modified to be the localized objective of subagents

> For example, in VRPTW, one objective is to minimize the total travel distance. Then, if a subagent represents one route, its goal is also to minimize travel distance. However, the objective of minimizing number of vehicle should be modified before it is localized to each subagent. Details could be found in Section 6.1.2.

Each subagent usually has more than one objective, since we are dealing with MOMCP.

Desire of subagent

Desire of subagents is determined by the objectives of the problem. It represents the motivational aspect of FGAS, which is measured by a *desire value*. A desire value consists of three parts:

- *Changes of overall objective.* It represents the cooperative aspect of a subagent. The more a move contributes to the overall objective, the more a subagent desires the move.

- *Future expectation.* This component is the information of future expectation, which is discovered intelligently by the subagent, as explained in section 4.1.2.

- *Balance of multiple objectives of the subagent.* This component is introduced to prevent over optimizing of one objective. A threshold $t_{i,k}$ is defined for each objective *k*. When the decrease in an objective value surpasses its threshold value, the desire for the move will be greatly decreased.

Formally, a desire value is defined as:

$$D_i(m) = \alpha_0 * u(m) + \alpha_1 e_i(X_i, S_{CAi}) + \sum_{k=2}^{y+1} \alpha_k f_{i,k}(m,k)$$

$$f_{i,k}(m,k) = \begin{cases} 0 \quad \text{when } u_{i,m} < t_{i,k} \\ \\ f_{i,k}(u_{i,m}) \quad \text{when } u_{i,m} > t_{i,k} \end{cases}$$

where i is the index of the subagent;

m is the move;

$$\sum_{j=0}^{y+1} \alpha_j = 1;$$

$u(m)$ is the overall objective change by applying the move m ;

$e_i(X_i, S_{CAi})$ is the function of future expectation;

S_{CAi} is the value obtained from related constraint agents;

$f_{i,k}(m,k)$ is the function of objective balance. When the change in one objective k surpasses its threshold $t_{i,k}$, it will return a value which is usually negative. Otherwise, it will return 0.

The desire value can be thought of as a subagent's willingness to execute a move. The selfishness of a subagent makes the solution improve according to its desire. It increases the speed of improvement in objectives of subagents, and hence leads to a fast solution convergence speed. However, excessive emphasis on selfishness will sacrifice the overall objective. Hence the negotiation for possible cooperation between subagents is very important.

Negotiation between subagents

The negotiation strategy of FGAS is extended from cooperativeness-based strategies [Jung and Tambe 2003]. It chooses the best move according to the *desire value*.

We define

- *desired agents* for a move *m* are those subagents with their desire values $D_i(m)>0$

- *non-desired agents* for a move *m* are those subagents with their desire values $D_i(m)<0$
- *conflict* is the situation where a best move includes some non-desired agents.

Negotiation is introduced into FGAS to deal with the conflict between subagents. By negotiation, we mean all the *non-desired agents* will compromise also according to three parts, overall objective change *u*, the move's future expectation *e* and the objective balance *f*. Different from generating a desire value, when a subagent calculates the probability for compromise, more weight is put on the overall objective value change. If there is **any** subagent who still cannot accept the move, the move is rejected.

Mathematically, the probability is defined as:

$$p_i(m) = P(u(m), e_i(m), \sum f_{i,k}(m))$$

where $u(m)$ is the overall contribution of best move *m*,

$e_i(m)$ is the future expectation of move *m*

$\sum f_{i,k}(m)$ is the aggregation of the objective balance for all the objectives, *k* is the index of objective

4.4 FGAS Workflow

After illustrating the main components of FGAS, an overall picture of how FGAS works is given as follows:

First, *AA*, *sAs* and *cAs* initialize. The initialization of *AA* and *cAs* are straightforward. The initialization of *sAs* uses some generic algorithm, such as Greedy Algorithm, to generate an initial solution for the problem.

Subsequently, the subagents start working. These subagents will run into each other on a random basis. They either improve their internal status by themselves or interact with the subagents they met. *AA* will keep as many un-FORBID subagents as possible working in parallel.

Several types of moves are defined for each specific problem. When two or more *sAs* interact, a list of move will be proposed, which follows the idea of local search. Actually, when the size of subagent is considerably small, brute-force idea can be applied to achieve better results. Among all the moves, the one with maximum contribution to the overall objective is evaluated by the related subagents. If conflict occurs, negotiation will be held. The interaction of subagents is illustrated by Figure 4.3.

The overall objective value is recorded by *AA*. When a termination criterion is met, the program will end.

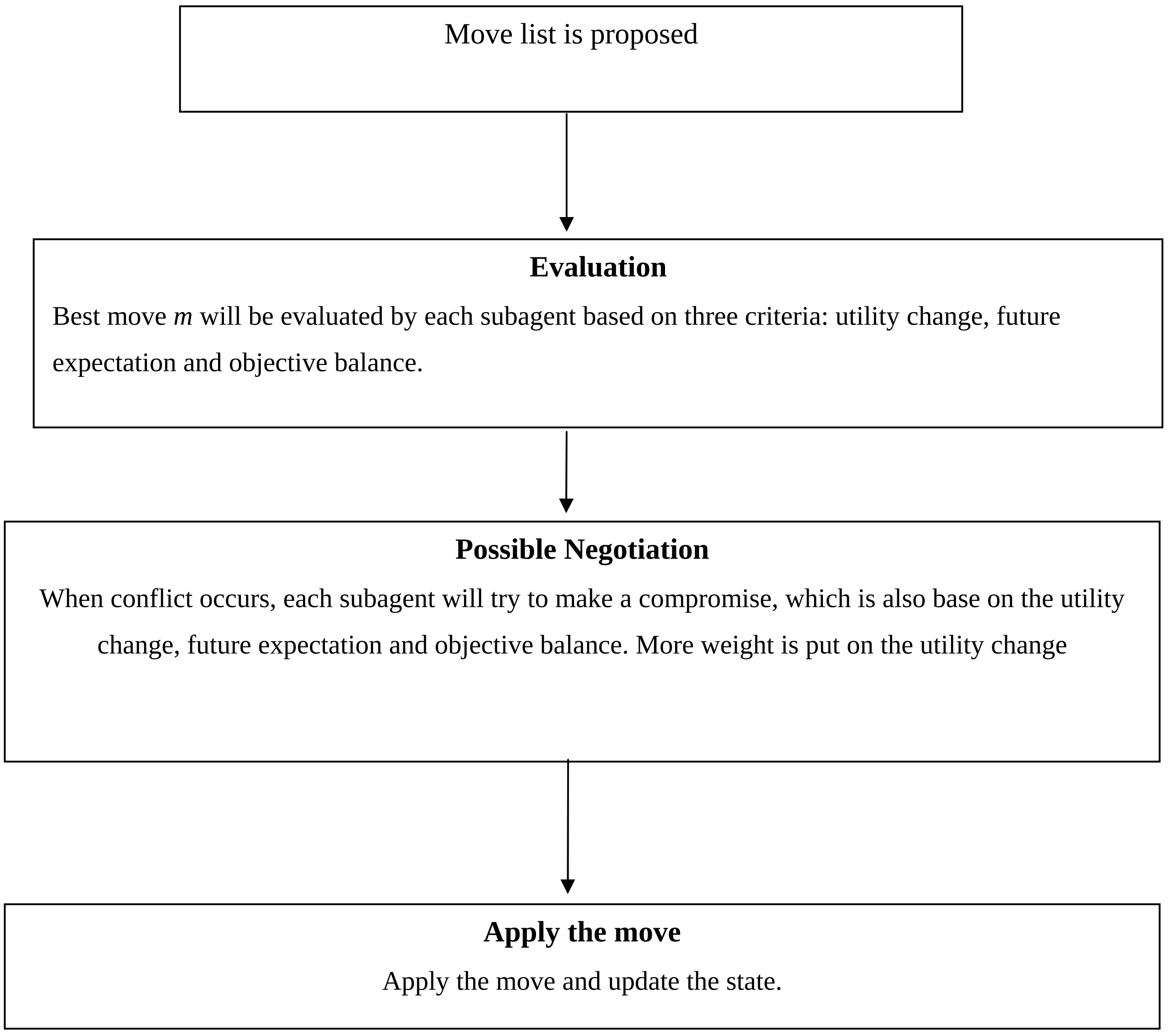

Move list is proposed
Evaluation
Best move *m* will be evaluated by each subagent based on three criteria: utility change, future expectation and objective balance.
Possible Negotiation
When conflict occurs, each subagent will try to make a compromise, which is also base on the utility change, future expectation and objective balance. More weight is put on the utility change
Apply the move
Apply the move and update the state.

Figure 4.3 Interaction of subagents

Chapter 5

Coarse-grained Agent System (CGAS)

In this chapter, we discuss the second agent-based approach which we term the Coarse-grained Agent System (CGAS). As discussed in Chapter 3, the classic method of solving multi-objective problem is combining multiple objectives to form one objective, and then apply the traditional single objective optimization method. Unfortunately, the optimization algorithm has no insight to which objective it is improving during the search. Consequently, much redundancy is incurred. To avoid such redundancy, in Coarse-grained Agent System (CGAS), we model the whole problem as several independent objective agents where each of them only takes responsibility of one objective.

5.1 Overview of CGAS

For a MOMCP, the problem is broken along its objectives. Each objective is assigned to an objective agent. The objective agents work collaboratively to achieve the overall objectives.

Stated formally, we are given an *n*-objective problem Q, the goal is to

$$\text{Maximize/Minimize } g_i(x)\ i \in [1,n] \text{ s.t. } C = \{c_1, c_2, ..., c_m\}$$

where g_i is one objective of *Q*, $i \in [1,n]$

$\boldsymbol{x}$ is the vector of decision variables

C is the constraint set consisting of constraints c_1 to c_m

we solve Q using *n objective agents*, where the objective of i^{th} objective agent is defined by:

$$\text{Maximize/Minimize } g_i(x) \text{ s.t } C_i \subseteq C$$

such that $C_1 \cup C_2 \cup ... \cup C_n = C$ and $C_i \cap C_{i+1} \neq \Phi$.

5.2 Objective Agent

An *objective agent (OA)* improves the solution in the perspective of its own objective. One feature of *OAs* in CGAS is it only subject to a subset of constraints. Thus, more freedom is allowed for an *OA* to find potentially good solution. Although the output of the intermediate *OA* may not be a feasible solution for the problem, the final solution, after optimized by all the *OAs*, will satisfy all the constraints. Since the constraint partition is highly problem-specific, we are not trying to propose a methodology to partition constraints, instead, we propose the following guidelines for constraint decomposition among *OAs:*

(1) Within one objective agent, the constraints should be those that are directly related to its objective function.

(2) Between two consecutive objective agents (in CGAS, the objective agent works sequentially, so we say “consecutive” here), the overlapping constraints are typically those are common to both problems such that the output from one sub-problem will not be over optimized at the expense of another.

(3) All the constraints should be satisfied after the solution is optimized by all *OAs*.

We will further explain the constraint decomposition in the context of IRPTW in Chapter 7.

5.3 CGAS Workflow

In CGAS, *OA*s work sequentially and iteratively. The idea of greedy algorithm is adopted-- the more important objective has more priority to be optimized. After optimized by one objective agent, the solution structure will be changed, and therefore the solution can be passed on to the next objective agent for further improvement. At the end of each iteration, the solution will be checked and the program will end when a terminating criteria is met.

The working flow can be illustrated by figure 5.1. The whole solution is passed along the direction of the arrow.

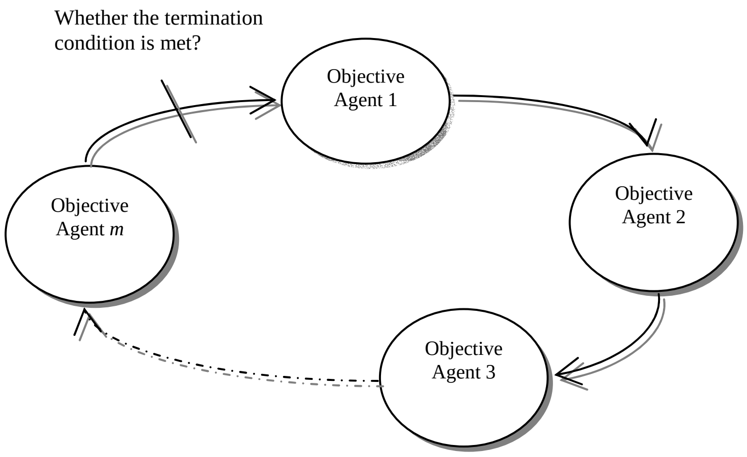


Figure 5.1 work flow of CGAS

Chapter 6

Solving IRPTW using FGAS

In this chapter, we discuss how to solve IRPTW using FGAS.

6.1 Mapping from IRPTW to FGAS

In a supply chain optimization problem such as IRPTW, the optimal solution is usually established through the cooperation and competition among the parties within the supply chain system. FGAS has an innate advantage in modeling and representing such supply chain optimization problem, since the subagent in FGAS also has a cooperative yet competitive nature.

6.1.1 Identifying agents

Subagent in IRPTW

In IRPTW, those parties are a set of routes. Each route is delegated to a route agent, i.e. the subagent in FGAS.

More formally, for IRPTW, a solution instance of IRPTW is viewed as a set of routes [Lao 2002], each serving a certain subset of customers at a sequence of particular time points.

Solution Plan $S=\{R_1,R_2,\ldots R_n\}$

where Routing Plan $R_i=\{(C_{i1},q_{i1},t_{i1})\rightarrow(C_{i2},q_{i2},t_{i2})\rightarrow\ldots\rightarrow(C_{in}, q_{in}, t_{in})\}$

q_{ik} $k \in [1, n]$ is the supply amount to customer C_{ik}

t_{ik} $k \in [1, n]$ is the service time to customer C_{ik}

Mapping to FGAS, a route agent RA_i is assigned to route R_i

Arbitration Agent in IRPTW

A route agent has only local knowledge of the environment. While it knows how much distance it has changed through a move, it is unable to compute the change of the overall inventory cost. Hence, the task of computing overall inventory is left to the arbitration agent.

If two route agents (route *a* and *b*) from different days (day *m* and day *n*) are adjusting their inventory interactively, the route agents between day *m* and *n* (say route set *R*) who serve the same customer with *a* or *b* cannot be active concurrently. The reason is inventory level of the customers in *R* will be affected or changed by the state change of *a* and *b*. Monitoring and preventing such FORBID interactions is the task of arbitration agent.

Constraint agent in IRPTW

There are three constraint agents in IRPTW: vehicle capacity constraint agent, time window constraint agent and customer holding capacity agent. For a route agent, it will related to one vehicle capacity constraint agent, one time window constraint agent and a set of customer holding capacity agents.

The basic function of constraint agents in IRPTW is to check whether the constraints are violated.

As discussed in Chapter 4, more information can be acquired from constraint agents beyond the basic “satisfy or not satisfy a constraint”. For example, in order to realize the local objective of minimizing number of vehicle (which will be explained in the next section) in IRPTW, the detail information of how much the move can change the vehicle transportation load is required. Such information is obtained from the vehicle capacity constraint agent. In FGAS, the return from vehicle capacity constraint agent is the exact value reflecting the effect of the move rather than simply “true” or “false”.

6.1.2 Handling objectives and constraints

The division of IRPTW is motivated and improved upon the work of [Lau et al. 2002], in which IRPTW is divided into VRPTW and the Dynamic Lot-sizing Problem (DLP). In addition, we also consider the decomposition of VRPTW into two single-objective sub-problems, following the scheme of [Gambardella et al. 1999]. Hence, we derived three objectives of IRPTW:

(1) minimizing number of vehicles used;

(2) minimizing total distance traveled;

(3) minimizing inventory and backlog costs.

Subject to

(1) Customer time windows;

(2) Vehicle capacity;

(3) Customer holding capacities.

Now we explain how to handle the objectives for IRPTW:

(1) Minimize number of vehicles—local estimated objective

This objective is translated to every route agent's desire of destroying itself by pushing out its transportation load to other route agents. However, such direct translation is impractical because when a route (say *a*) transfers its transportation load to others (say *b*), the load of *b* will increase. Route agent *b* will refuse such action because it will violate its desire of decreasing its own transportation load.

Hence, each route agent's objective is modified as: 1) when a route agent detect that it stands a chance of emptying its transportation load, it will encourage the action which can help it achieve this desire; 2) when a route is nearly full, it has the desire to top up its transportation load without violating capacity constraint. The second part of the local objective is based on the observation that when the total demand is fixed, the fewer vehicles are used, the more one vehicle will carry. This local estimated objective can be expressed using the curve in figure 6.1.

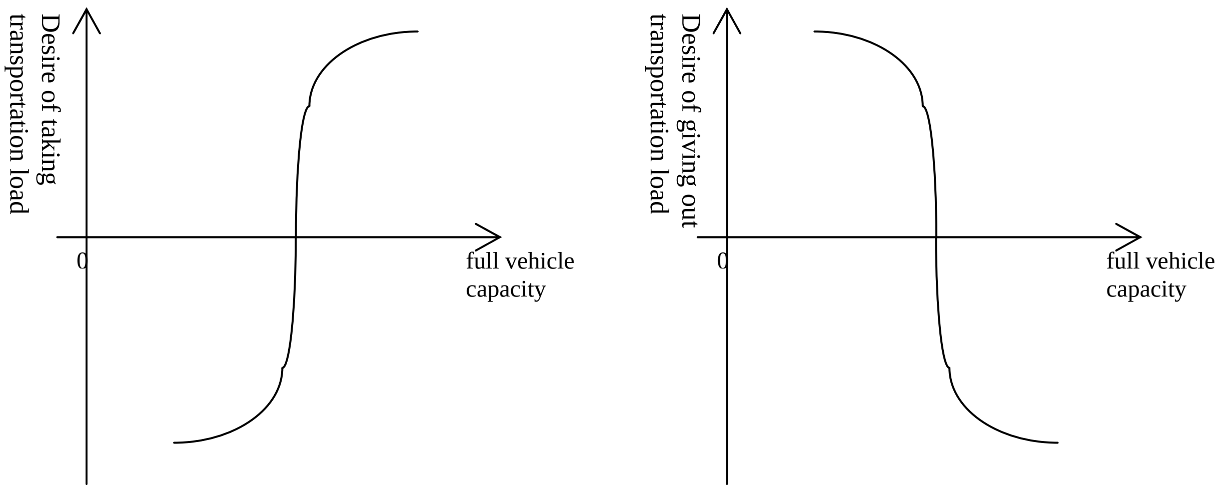

Desire of taking transportation load
0
full vehicle capacity
Desire of giving out transportation load
0
full vehicle capacity

Figure 6.1 Function of localized *minimize number of vehicle*

Two kinds of information are needed for this local objective: 1) the current transportation load of the route agent; 2) the change of the transportation load contributed by the move. The second piece of information is obtained from **vehicle capacity constraint agent**.

(2) Minimizing travel distance—local computable objective

This objective is handled in a straight forward manner. Each route agent is willing to contribute to the improvement of overall objective of minimizing travel distance.

(3) Minimizing inventory cost—global computable objective

The inventory cost is impossible to compute locally. Suppose a move shift the supply amount for customer *k* from day *i* to day *j*, all the inventory level of customer *k* between day *i* to day *j* will be affected. Therefore, the task of computing inventory cost is left to *arbitration agent* instead of making this objective localize to each route agent.

6.2 Solution Initialization

The purpose of initialization is to generate an initial set of route plans. Our method is treating the whole problem as multiple independent VRPTW instances, where each instance represents the routing plan in a different day. We apply a greedy algorithm to generate the initial route set. Obviously, the inventory cost of the initial solution is 0.

6.3 Interaction between Route agents

The purpose of interaction between route agents is to change their states in order to improve objective value. We adopt the local search strategy. All possible local moves listed below will be considered and the best move will be chosen and evaluated. The moves are defined as follows:

1) SelfRelocate —move involving only one route agent

Without changing the members of a route, the route agent tries to rearrange the sequence of the customers to reach a better configuration. A *SelfRelocate* operation describes the act of changing one customer's position in the route. Figure 6.2 illustrates this operation. The time complexity of this operation is O(K^2), where K is the number of customers in one route, which is bound by total customer number N.

This operation is the simplest one. It only affects the objective of minimizing traveling distance, so there is no need to consider the balance between different objectives. In addition, it only involves one route agent. So, no conflict will happen and no negotiation is needed.

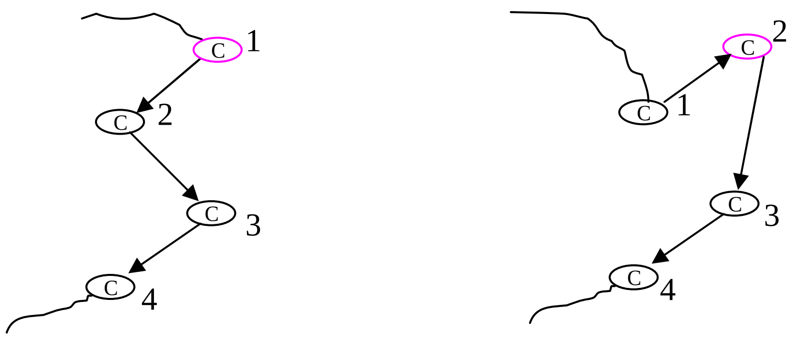


Figure 6.2: Route Agent: *SelfRelocate* Operation

2) *n-m-exchange—move between two route agents within the same day*

Within the same day, we introduce a complex move between the route agents—*n-m* exchange. This move describes the act of exchanging a segment of length n in one route with a segment of length m in another (see Figure 6.3). Both traveling distance

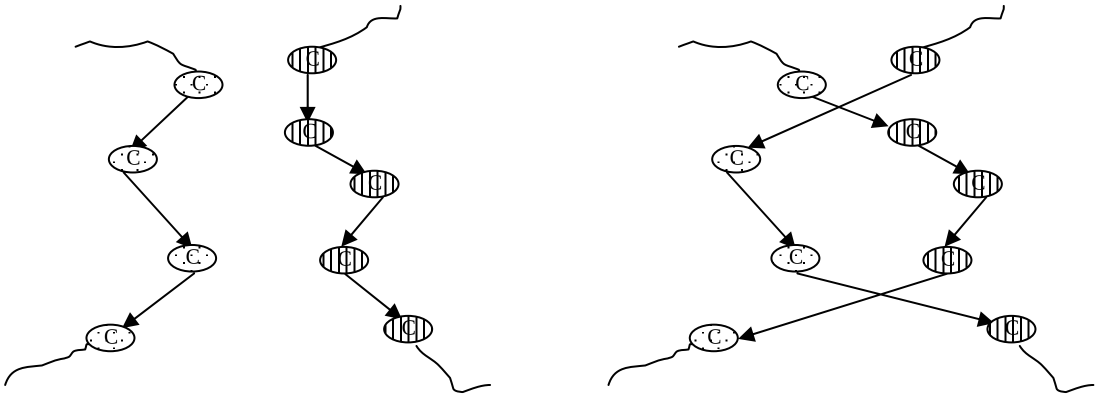


Figure 6.3: Route Agents in the same day : *n*-*m* move

This example above is a 2-3 move

and number of vehicle can be affected by this move. When the *n* or *m* is equal to the length of the route, the number of vehicle can be reduced by one.

In fact, this move can be seen as the combination of several standard moves in VRPTW. When either *n* (change all variables to italics throughout this thesis) or *m* is equal to 0, the move is reduced to the move “relocation”. When *n*=1 and *m*=1, the move is the traditional “exchange”. When either *n* or *m* is equal to the route length, this move can be seen as a “distribution”.

If we set *n* and *m* to be any value between 0 and their route length, all the possible moves are listed based on the brute-force idea. However, considering the real situation, brute-force is sometimes over powered. Usually an upper bound *L* is set for the route segment length. Therefore, the time complexity of this move is $O(L^2(L+K)^2)=O(K^2)$, where *K* is the length of the route.

3) Merge—move between two route agents in different days

When two route agents serve the same customer on different days, the two services can be merged into one with a larger supply. Although the inventory cost will increase, we can expect a better overall objective since the inventory cost is usually smaller than the saved transportation cost. It's the arbitration agent's task to prevent the over increase in the inventory cost. The time complexity of this move is O(*K*), where *K* is the length of the route.

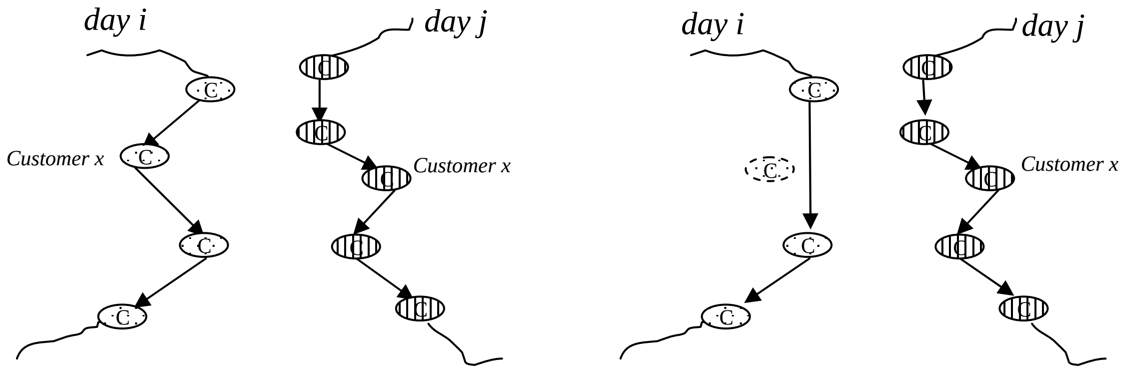


Figure 6.4: Route Agents in the different days: merge

4) InvRelocate—move between two route agents in different days

The aim of introducing this move is to adjust the inventory between different days. This move describes the act of relocating one customer's supply from a route in one day to a route in another day. Using this move, the inventory cost is decreased. Also, there is a chance of reducing one vehicle. Time complexity of this move is O(*K*), where *K* is the length of the route.

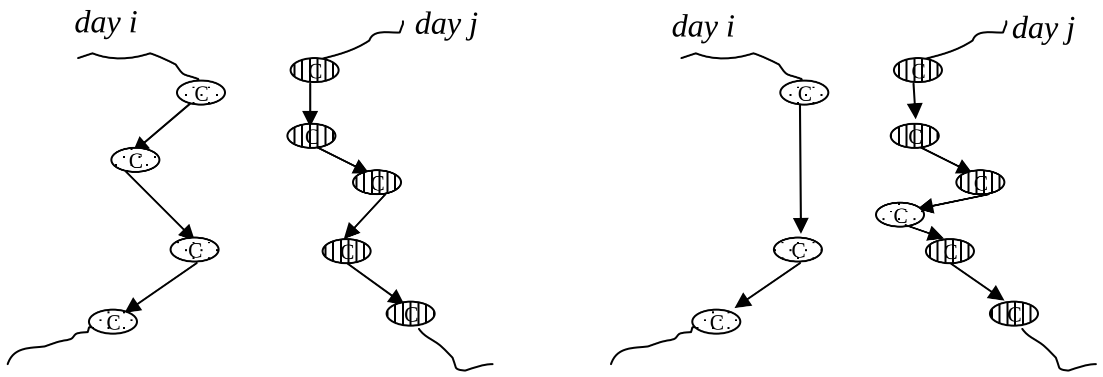


Figure 6.5: Route Agents in the different days: *InvRelocate*

6.4 Agent Coordination

One route agent will encounter another on a random basis. When two[1] route agents interact with each other, a list of moves will be proposed and the best one will be evaluated. The desire value of a route agent is defined by:

$$D_i(m) = \alpha_1 u_m + \alpha_2 e_{veh} + \alpha_3 f_{dist}$$

where $\sum_{i=1}^{3} \alpha_i = 1$

u_m is the change of overall objective by move *m*

e_{veh} is future expectation regarding the number of vehicle.

f_{dist} is the balance of the objective of minimizing travel distance.

$$f_{dist} = \begin{cases} 0 \quad \text{when} \quad \Delta D_{dist} < t_{dist} \\ \\ f_{dist}(\Delta D_{dist}) \quad \text{otherwise;} \end{cases}$$

ΔD_{dist} is the change of travel distance

Similarly, the desire value for *AA* can be defined, since *AA* handles inventory cost and participates in every interaction.

When the best move is not desired by any of the route agent or arbitration agent, they are willing to negotiate by accepting the move with a probability based on the move's overall contribution and its own desire:

[1] In our implementation, we only consider the case in which only two route agents interact with each other.

$$p = P(u_m, v_i)$$

where u_m is the overall objective change by applying move *m,*

v_i is the desire value of route agent *i* or arbitration agent *AA*

If the negotiation fails, the move will not be accepted.

The overall objective value is recorded by *AA*, when a termination criterion is met, the program ends.

Chapter 7

Solving IRPTW using CGAS

In this chapter, we discuss how to solve IRPTW using CGAS.

7.1 Mapping from IRPTW to CGAS

As described in section 6.1.2, we derived three objectives of IRPTW:

(1) VRPTW 1: minimizing number of vehicles used;

(2) VRPTW 2: minimizing total distance traveled;

(3) DLP: minimizing inventory and backlog costs.

It has the following constraints:

(1) Customer time windows;

(2) Vehicle capacity;

(3) Customer holding capacities.

Each objective is subject to a subset of constraints, which is illustrated in Figure 7.1. One objective agent is delegated to each sub problem. There objective agents work sequentially and iteratively. The output of one objective agent (*OA*) is the input of the

IRPTW

Minimize total number of vehicles, total distance traveled, and inventory and backlog cost

Subject to customer time windows, retailer holding capacity and vehicle capacity constraints.

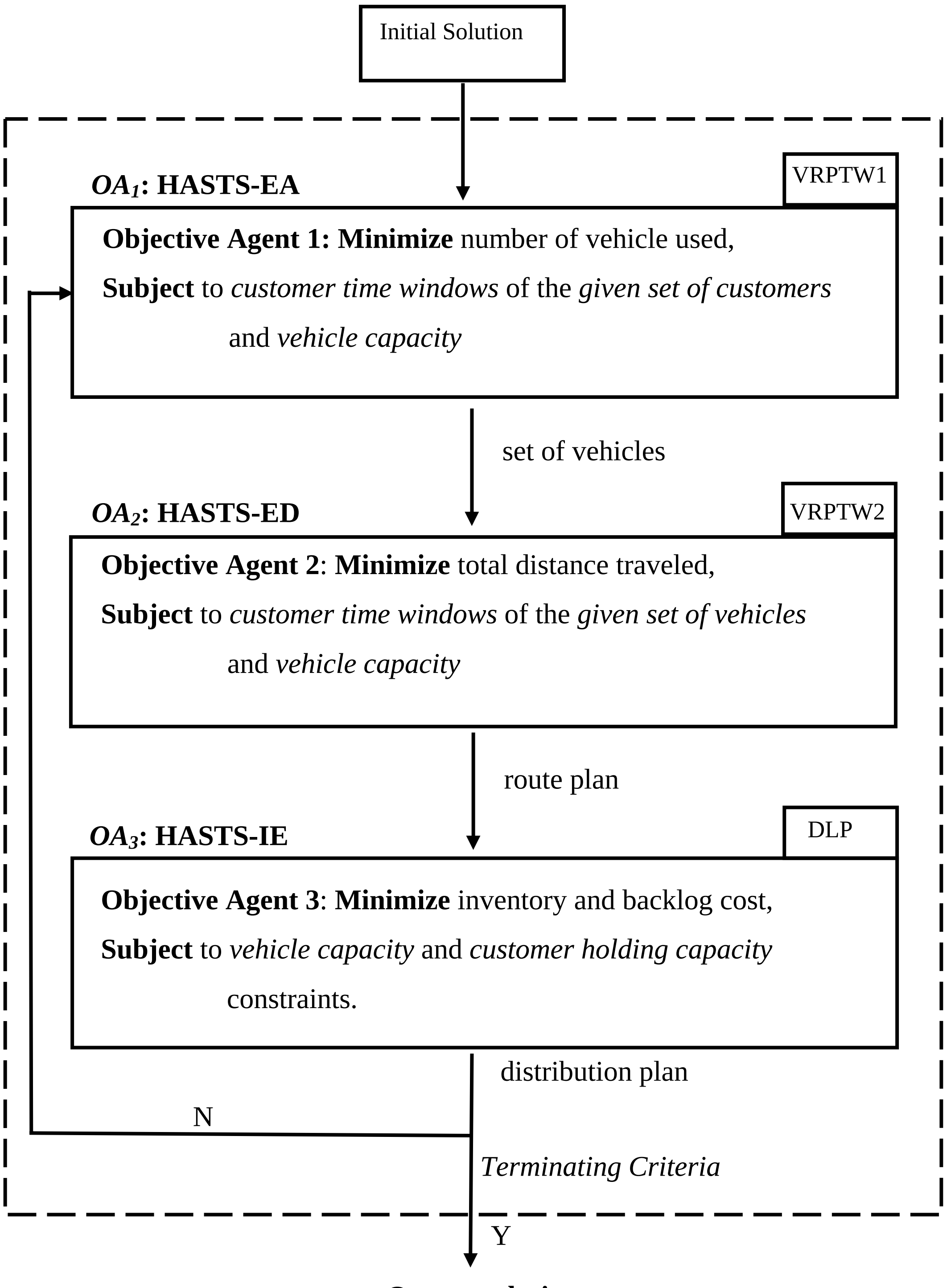

Initial Solution
OA₁: HASTS-EA
VRPTW1
Objective Agent 1: Minimize number of vehicle used,
Subject to *customer time windows* of the *given set of customers* and *vehicle capacity*
set of vehicles
OA₂: HASTS-ED
VRPTW2
Objective Agent 2: **Minimize** total distance traveled,
Subject to *customer time windows* of the *given set of vehicles* and *vehicle capacity*
route plan
OA₃: HASTS-IE
DLP
Objective Agent 3: **Minimize** inventory and backlog cost,
Subject to *vehicle capacity* and *customer holding capacity* constraints.
distribution plan
N
Terminating Criteria
Y
Output solution

Figure 7.1: Solving IRPTW using CGAS, employing HASTS as conquer technique

next one. When the termination condition is met, the final solution is output. Notice here, the program cannot stop at any middle point of the system because each *OA* only satisfy part of constraints and the only feasible solution is produced after the last *OA*'s optimization.

Another important issue is the solution convergence under such division. Fortunately, this has been proved in [Lau et al. 2000].

Now we explain how to decide the constraint set for each *OA*. For OA_1 and OA_2, since we are concerned with routing of vehicles, the natural set of constraints should be the vehicle routing constraints and not the customer storage constraints. It is not necessary to consider to customer storage capacity constraint because OA_3 can always properly take care of it and output a feasible solution. Similarly, for OA_3, the customer storage capacity constraints are more important comparing to time window constraint. In fact, the change OA_3 made will never cause the violation of time window constraint. Another point that should be explained here is the overlapping constraint between OA_2 and OA_3. If vehicle capacity constraint is removed from OA_2, an extreme case of pushing many the customers into one vehicle might happen. If such solution is passed to OA_3, in order to satisfy vehicle capacity constraint we will suffer a huge inventory cost. What make things worse is that such situation cannot be remedied by other *OAs*.

7.2 A proposed conquer technique--HASTS

What we have introduced above is the framework of CGAS. A disadvantage of this system is the framework itself cannot guarantee the quality of the solution. The optimization technique employed by the objective counts a lot.

In this thesis, HASTS - Hybrid Ants System (AS) and Tabu Search (TS), a hybrid model that contains 4 derived models, is employed as optimization technique by objective agents.

7.2.1 Ants System and Tabu Search

The standard AS builds a complete solution with each ant and the density of the pheromone trails reflects the preference of the solution structure. The pheromone trails provides information sharing and intelligence in which the quality of the solution can be optimized. As it does not require an initial solution, AS can be viewed as an excellent construction heuristic. Being a meta-heuristic, it is also not limited to a single type of problem and offers solutions of high quality. Hence, we adopt AS as a component for our hybrid model.

On the other hand, the standard TS incorporates both an adaptive memory and a responsive exploration. The adaptive memory allows TS to reduce solution cycling, and is capable of making radically changes based on past history. Responsive exploration allows TS to apply intensification and diversification strategies adaptively. However, TS is not without any weaknesses. Its effectiveness hinges on the neighborhood structure and Tabu list. Hence, TS will be trapped in a poor local optimal if it lacks an effectual neighborhood and Tabu list.

We deem AS and TS to be a good combination as the two meta-heuristics are very different and complementary in nature. HASTS hybridizes the two meta-heuristics to form 4 derived models, each adjusting the relative importance of AS and TS to cater to the needs of different sub-problems. The 4 derived models are *Empowered*

Ants (HASTS-EA), *Improved Exploitation* (HASTS-IE), *Enhanced Diversification* (HASTS-ED**)** and *Collaborative Coalition (*HASTS-CC).

7.2.2 HASTS-EA *(Empowered Ants)*

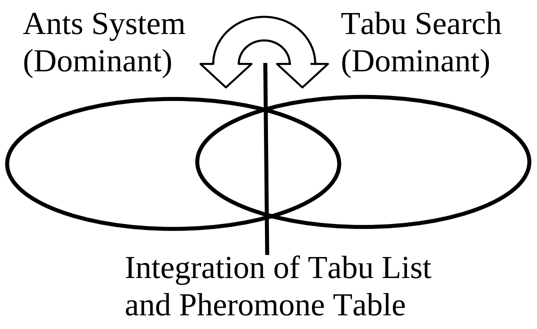


Figure 7.2 HASTS- EA (Empowered Ants)

This derived model arises from the observation that while the AS reaches near optimal solutions, it suffers from a tendency of solution cycling in the near optimum region due to their emphasis on the strong pheromone trails. By empowering the ants with memory, it reduces the chances of reconstructing the same solution. An analogy can be drawn where each ant becomes more intelligent to find a better trail by *not* following false tracks laid by previous ants. TS uses a Tabu list to reduce cycling on the same set of solutions. While the AS optimizes the solution based on its pheromone trails as a “preference” memory, solution cycling is reduced via the Tabu list. Furthermore, TS can be applied to modify the solutions radically, hence encouraging exploration that helps to escape from local optimality. In our implementation, the AS is modified to include a Tabu list, which records the solution made by each ant in a single iteration. Subsequently, each ant in the iteration would check if the next move is Tabu-ed. If it is, the move will be dropped and a new move will be generated. The Tabu list is reset at the end of the iteration. A pseudo-code of HASTS-EA is shown in the Figure 7.4

7.2.3 HASTS-IE (Improved Exploitation)

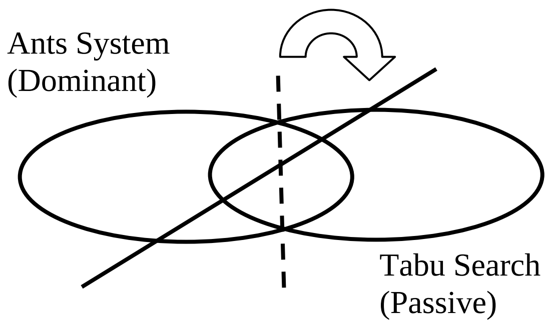


Figure 7.3 HASTS- IE(Improved Exploitation)

In this model, TS is embedded in the AS to conduct *intensification* search on the best solution. A similar design has been employed in [Stutzle and Dorigo 1999] to produce good solutions for TSP. This model offers two advantages. First, by updating the pheromone trail only after intensifying the best solution, we increase the probability of finding a better solution by subsequent ants. Second, due to the probabilistic guided nature of AS, this narrows the chances of reaching an optimal solution if it happens to be radically different from local optimum. For example, it is well known that for TSP, the AS may take a long time before it reaches optimality,

```
Procedure: HASTS-EA()
While (termination-criterion-not-satisfied)
        While (Max_Ant_Not_Reached)
              Ants_generation_and_activity
              Pheromone_Evaporation
              Reset_Tabu_List
              Daemon_actions
        end Schedule_activities
   end While
```

```
Procedure: Ants_generation_and_activity ()
   While (available_resources)
         Schedule_creation_of_new_ant
         New_Solution = New_active_ant
      update_Tabu_List (New_Solution)
   end While
```

```
Procedure: New_active_ant ()
   Initialize_ant;
   M = read_Pheromone Trail
   T = read_Tabu_List
   While (current_state != target_state)
         A = read_local_ant_routing_table
         P = compute_transitional_probabilities (A, M)
         For each Next_state do
         Next_state = apply_ant_decision_policy(P)
         While (check_Tabu_List (Next_state) == non-Tabued)
         Move_to_next_state (next_state)
         If (online_step-by-step_pheromone_update)
               Deposit pheromone
               Update M
         end If
   end While
   If (online_delayed_pheromone_update)
         For each visited_arc   do
               Deposit pheromone
               Update M
         end
   end If
```

Figure 7.4   Pseudo codes of HASTS-EA

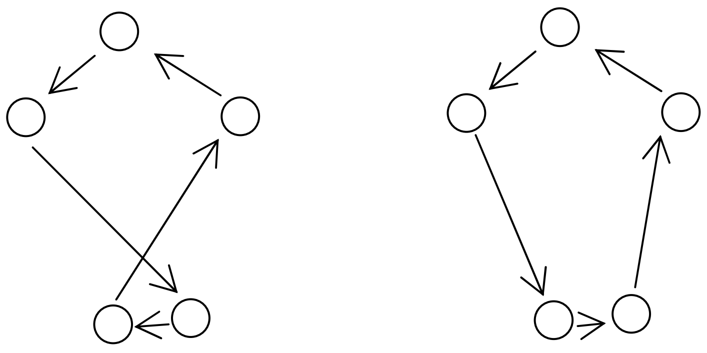


Figure 7.5 Use TS to help AS reduce crossing

due to the presence of “crossings” in the tour (see Figure 7.5). With the help of TS, such crossings can be eliminated easily by swap moves such as 2-opt.

7.2.4 HASTS-ED (Enhanced Diversification)

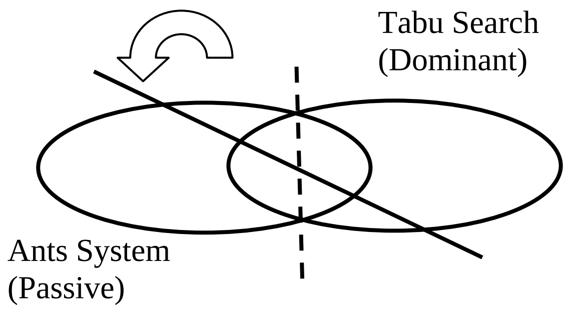


Figure 7.6 HASTS- ED(Enhanced Diversification)

In this model, AS acts as a diversifier for TS. As TS suffers from local optimality, a diversification strategy is to apply another meta-heuristic as a diversifier (e.g. [Li and Lim 2001]). HASTS-ED uses an AS diversifier with following rationale. First,

the probabilistic nature of the AS gives a higher chance of successfully diversifying from the local optimum. Second, the diversifier should make a radical move from the current solution so as to explore new regions. Although a random restart is a good strategy, the new starting solution is often poor. AS provides a remedy to this by reconstructing quality solutions.

7.2.5 HASTS-CC (Collaborative Coalition)

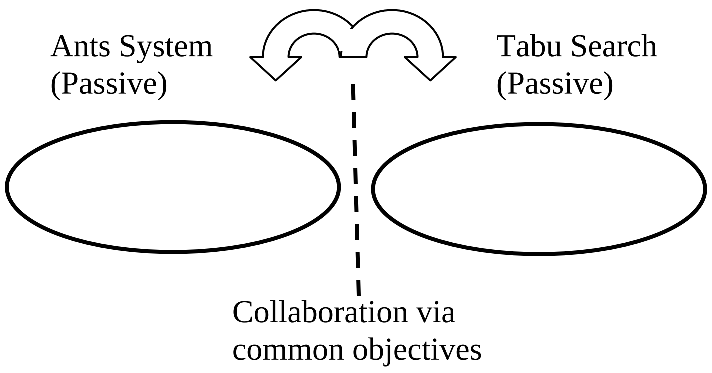


Figure 7.7 HASTS- CC(Collaborative Coalition)

This final model proposes a collaborative coalition between the AS and TS. This model offers the least coupling between the two meta-heuristics but allows great flexibility in the solution approach. One configuration of HASTS-CC is to espouse the two-phase approach as advocated by [Schulze and Fahle 1997]. This approach consists of a construction phase follow by a local improvement phase. The AS works extremely well for the construction phase as it could be used independently to obtain quality solutions. Being an optimization heuristic, TS fit naturally into the second phase of the approach. Such collaboration exploits the natural heritage of each meta-heuristic.

HASTS is a good choice of objective agent for solving MOMCP. First, it has four derived models and is capable to adapt itself to different objective of sub problems.

Second, it is coding economic. Programmer can use only the code of Ants System and Tabu Search to realize the effectiveness which can not be achieved by pure Ants system or Tabu Search. All of these can be demonstrated by our experiment on Inventory Routing Problem with Time Window.

7.3 Conquer technique for each objective agent

OA1—HASTS-EA

Sub-problem 1: We can reformulate this objective to its dual model and writing it as maximizing the customers served in given a set of vehicles, and reduce the required vehicles each time we find a solution that serves all the customers. *OA_1* employ the ***HASTS-EA*** derived model. Initially *m* vehicles are obtained by applying a greedy heuristic to serve all customers. The algorithm then reduces the value of *m* by 1 and seeks to construct a feasible solution that services all the customers. Once a feasible solution is found, the number of vehicles is reduced to the best-found number of vehicles and the process is repeated for a new feasible solution. This sub-problem requires search so as to find a configuration where the customers can fit into the pre-set vehicles. HASTS-EA performs well since the tabu list assists each ant in an iteration to construct a radically different solution. Although other derived models can also be used, they lack the intensified exploration that HASTS-EA provides.

OA2—HASTS-ED

As we have already optimized on the number vehicles, this sub-problem will have a tighter solution space. HASTS-EA is not very effective in such situation because of the difficulties involved in constructing different feasible solutions on an allowed number of vehicles, due to the nature of the AS. Hence, ***HASTS-ED*** is employed by OA_2, to minimize the total distance on a fixed set of vehicles. HASTS-ED uses TS as the core heuristic with AS acting as the diversifier. TS is effective in solving this sub-problem as it optimizes the route distance rather than reconstructs the solutions. When TS meets a local optimum, it randomly selects some of the routes to be reconstructed by AS. AS then assists TS by radically re-configuring the selected partial routes. The output is a route plan which is the input to OA_3.

OA3—HASTS-IE

HASTS-IE is adopted by OA_3 to minimize the inventory and backlog costs. In order to reduce inventory or backlog, more frequent deliveries have to be made, hence increasing the transportation cost. Hence, the goal here is to minimize the number of customers served each day *without* increasing the total cost. Our goal is to delete customers from routes in a manner that does not incur additional cost. HASTS-IE uses the AS to construct different solutions. It then uses TS to improve its exploitation to reduce missing elite solutions. The TS uses the standard "add", "delete" and "swap" moves that attempts to improve the solution quality found by the AS.

After we fix the set of customers and the route plan, we are able to apply the *Minimum Cost Flow* (*MCF*) model to work out a distribution plan. For a given set of customers and their route plan, the *MCF* can work out the optimal distribution plan. ([Lau et al. 2002])

The output is a distribution plan that induces the set of customers to be served for OA_1. The solution can be either passed on for further optimization or be output as the final solution.

Chapter 8

Experiment Results and Analysis

Extensive experiments have been conducted to demonstrate the efficiency of FGAS and CGAS. The experiment results are presented in this chapter. Some observations and analysis will also be provided.

8.1 Experiment Setup

Following the strategy of [Lau et al. 2002], our test cases are generated based on Solomon test cases for VRPTW. A total of 56 test cases in Solomon test cases cover different scenarios. In C series test cases (C101-C109 and C201-208), the locations of customers are clustered. These test cases are best solved by assigning vehicles to service the same or nearby clusters in the problem. In R series test cases (R101-R112 and R201-211), the locations of customers are randomly decided. Solving them is more problem specific. In RC series test cases (RC101-108 and RC201-208), the locations of customers are both random and clustered. Moreover, the C201-208, R201-211 and RC201-208 are also called extended Solomon test cases, in which there are 200 customers.

In the generated IRPTW test cases, the vehicle capacity, locations and time-windows of the customers and depot are those specified in the Solomon instances. The planning period is 10 days. The demand d_{it} of customer i for day t (t=1,…, 10) is equal to the demand d_i of the Solomon instance, by partitioning the value 10*d_i into 10 parts, i.e. d_{i1}, d_{i2},…,d_{i10} randomly such that d_{it} is within the range [0.5*d_i, 1.5*d_j]. The capacities of consumers and warehouse are the vehicle capacity and infinity respectively. As for cost coefficients, the inventory cost and backlog cost for each customer are 1 and 2 respectively. The transportation cost of each route is 10 times of its total distance.

8.2 Analysis of Results

The experiments were conducted on a Pentium 1.13GHz machine with 128M memory. We compare the solution quality, run time performance, and solution convergence speed of each approach.

1. Solution quality

In Table 8.1, we compare the solutions of FGAS and CGAS with the previous work. The columns *ILS+VRP* and *TS+VRP* denote the results obtained from [Lau et al.[1], 2002], where *ILS+VRP* is the results obtained using Iterated Local Search [Gu 1992; Johnson 1990] and *TS+VRP* employs a Tabu Search technique. The cases from R210 to RC208 are not experimented in previous work, therefore, in Table 8.1 we left blank in the corresponding cells. The column *FGAS* and *CGAS* refer to the results obtained using FGAS and CGAS respectively. The Imv_F is improvement of FGAS than the average of *ILS+VRP* and *TS+VRP*. Similarly, Imv_C is that for CGAS. We can see from Table 8.1 both FGAS and CGAS achieve much better results than the previous work. On average, CGAS improve the previous solution by 56% and FGAS improves previous solutions by 53%.

The gap between FGAS and CGAS are given in column GAP_{CF}. Note the CGAS gives a better result than FGAS, the average gap is 6.1% (with min 1.8% for C205 and max 9.9% for R206). The reason is CGAS works in a more centralized manner. Each improvement of solution is precisely calculated and the best one is chosen (before reach the local optimal). In contrast, FGAS enables subagents, which represent subsets of variables, to evolve towards their own objective. Although subagents are generally cooperative, they have their own opinion of the quality of a move. Unfortunately, those opinions are not always right, but sometimes subagents are too stubborn to accept the result of negotiation and selfishly stick to their wrong opinion, which lead to the decrease of solution quality.

Test Cases	**ILS+VRP**	**TS+VRP**	**FGAS**	**Imv_F**	**CGAS**	**Imv_c**	**GAP_{CF}**
C201	113263	112821	53654	52.54%	52104	53.91%	3.0%
C202	117483	124312	55756	53.88%	53404	55.83%	4.4%
C203	131920	122055	56901	55.19%	53620	57.78%	6.1%
C204	136384	142300	57401	58.81%	54778	60.69%	4.8%
C205	116147	109248	52827	53.12%	51907	53.94%	1.8%
C206	123978	127876	53685	57.37%	50507	59.89%	6.3%
C207	122204	117735	53935	55.04%	51453	57.11%	4.8%
C208	124110	125667	54052	56.72%	52501	57.96%	3.0%
R201	111330	116893	63538	44.32%	62034	45.64%	2.4%
R202	116982	114717	60593	47.70%	56071	51.60%	8.1%
R203	110215	115070	56550	49.80%	53000	52.95%	6.7%
R204	114118	114118	53139	53.44%	49708	56.44%	6.9%
R205	122333	123009	57088	53.46%	53877	56.08%	6.0%
R206	120928	123251	57958	52.53%	52747	56.80%	9.9%
R207	115438	115438	55271	52.12%	51867	55.07%	6.6%
R208	120011	117255	51764	56.37%	49541	58.24%	4.5%
R209	116840	120725	57455	51.63%	52453	55.84%	9.5%
R210	-	-	56630	-	52478	-	7.9%
R211	-	-	54781	-	50521	-	8.4%
RC201	-	-	72902	-	67765	-	7.6%
RC202	-	-	71149	-	67534	-	5.4%
RC203	-	-	66275	-	61722	-	7.4%
RC204	-	-	62255	-	60542	-	2.8%
RC205	-	-	70920	-	68507	-	3.5%
RC206	-	-	70242	-	64750	-	8.5%
RC207	-	-	66333	-	61677	-	7.5%
RC208	-	-	61994	-	58330	-	6.3%
AVG				53.18%		55.63%	6.1%

Table 8.1. Comparison of Results for IRPTW test cases

2. Run Time Performance

Table 8.2 compares the time performance of CGAS and FGAS. T_C and T_F columns denote the running time for CGAS and FGAS to achieve their best solution respectively. The column $T_{c=f}$ refers to the time for CGAS to reach the solution with same quality as FGAS. On average, CGAS spends 8 times longer than FGAS to achieve the same-quality solution. For some cases, such as RC201, FGAS runs as much as 44 times faster than CGAS.

Calculating each step precisely in CGAS does bring us high solution quality but at the expense of computational time. In FGAS, concurrent computing will further increase

Test Cases	**T_C (s)**	**$T_{c=f}$(s)**	**T_F (s)**	**$T_{c=f}/T_F$**
C201	2203.20	766.93	173.16	4.43
C202	1741.33	677.69	154.72	4.40
C203	2423.05	554.14	134.75	4.14
C204	1399.94	602.68	209.20	2.88
C205	2763.52	638.76	224.32	2.85
C206	2497.03	1077.41	277.94	3.89
C207	2539.15	685.26	287.33	2.39
C208	2278.86	790.36	199.67	3.97
R201	828.35	754.32	22.45	33.60
R202	2089.18	734.89	34.66	21.20
R203	2027.01	649.47	30.72	21.14
R204	2046.14	716.17	74.77	9.58
R205	1789.15	924.79	36.49	25.34
R206	1862.95	689.81	49.36	13.98
R207	2454.79	922.07	55.57	16.59
R208	1468.98	1000.01	114.28	8.77
R209	1918.24	602.47	39.42	15.28
R210	1812.79	828.31	48.05	17.24
R211	2821.55	652.79	72.43	9.01
RC201	2052.38	1108.78	24.65	44.98
RC202	1817.48	951.42	33.93	28.04
RC203	2581.78	1057.68	47.27	22.50
RC204	2656.73	1210.69	87.83	13.78
RC205	1221.23	852.59	26.42	32.27
RC206	2782.00	729.54	30.58	23.86
RC207	2058.50	1008.37	34.28	29.42
RC208	2848.84	758.07	66.82	11.35
Avg	2110.52	812.7952	95.81	8.48

Table 8.2 Time performance of CGAS and FGAS

the computational speed. The subagent’s ability to intelligently detect potentially good solution and realize it through its selfishness also contributes to its good run time performance.

3. Solution Convergence

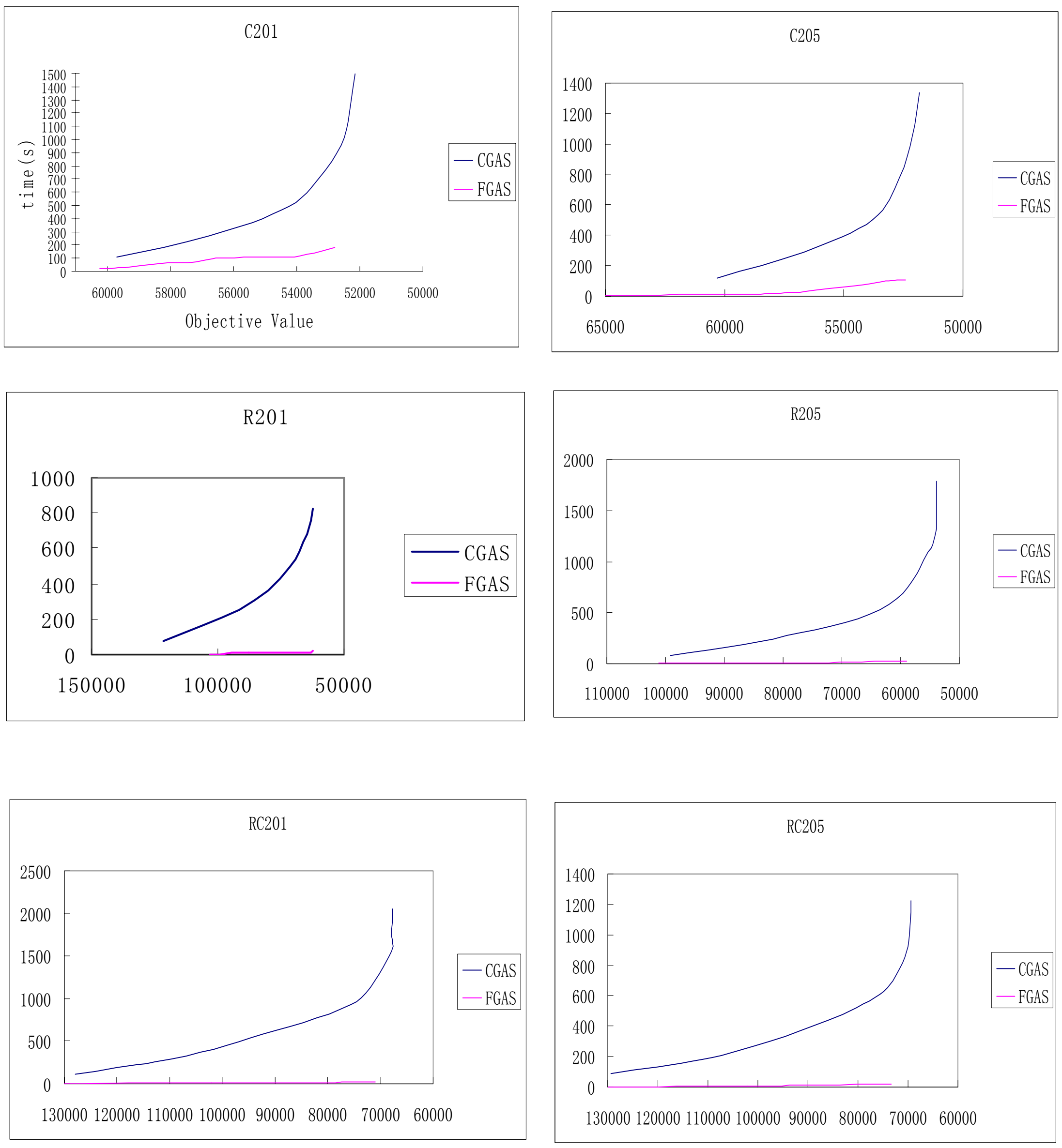


Figure 8.1 solution quality vs. running time

Figure 8.3 shows the solution convergence curves for FGAS and CGAS, using C201, C205, R201, R205, RC201 and RC 201 as examples. We can see clearly FGAS converges quickly to its best solution, while CGAS is relatively slow, especially when it reaches its near best solution region. However, FGAS fails to reach the same quality as CGAS, which can be seen from the x-coordinate of the curve's end point.

4. *Selfishness of subagents*

Finally, we like to demonstrate the effectiveness of the subagents' intelligence realized through subagents' selfishness. We conduct another set of experiments using selfishness-excluded FGAS (Table 8.3). We list the results of selfishness-included FGAS as the comparison. The columns "W/O Selfishness" and "With Selfishness" give the average solution achieved by subagents without and with selfishness respectively. The GAP column gives the gap between the solution qualities. Each test cases run for 40000 iterations. The effectiveness of the subagents' selfishness can be clearly seen through the gap (Also see Figure 8.2).

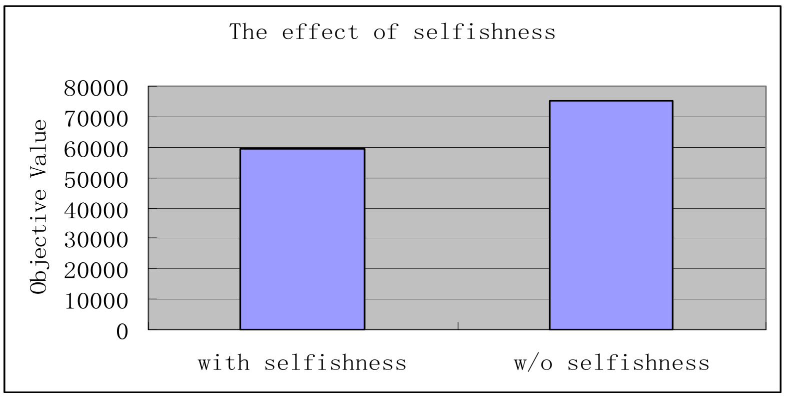


Figure 8.2 Gap between solution quality given by selfishness-included and selfishness-excluded FGAS

Test Cases	With Selfishness	W/O Selfishness	GAP
C201	53654	69237	0.29
C202	55756	77647	0.39
C203	56901	79508	0.40
C204	57401	81150	0.41
C205	52827	56793	0.08
C206	53685	80637	0.50
C207	53935	79953	0.48
C208	54052	71114	0.32
R201	63538	72333	0.14
R202	60593	71159	0.17
R203	56550	72387	0.28
R204	53139	67882	0.28
R205	57088	70232	0.23
R206	57958	71229	0.23
R207	55271	71334	0.29
R208	51764	69700	0.35
R209	57455	68052	0.18
R210	56630	66501	0.17
R211	54781	68236	0.25
RC201	72902	84361	0.16
RC202	71149	83744	0.18
RC203	66275	82398	0.24
RC204	62255	86501	0.39
RC205	70920	81217	0.15
RC206	70242	82991	0.18
RC207	66333	82392	0.24
RC208	61994	78066	0.26
Avg	59446.22	75064.96	0.27

Table 8.3 Comparison between FGAS with selfishness and without selfishness.

Chapter 9

Conclusion

In this thesis, we proposed two agent-based approaches for solving multi-objective multi-constraint problems.

The rationality of agents and the strength of local-search are combined in FGAS to successfully solve the large-scale optimization problem. The objectives are reflected through the desire of a set of subagents and realized through their cooperation and competition. By competition, the objectives of subagents are able to improve fast, which indirectly lead to a fast solution convergence speed. By cooperation through cooperativeness-based negotiation, FGAS is also able to produce a quality solution.

CGAS deals with the objectives in a centralized manner by assigning an objective agent to each objective. Each objective agent works on the entire solution. In this way, CGAS avoids the trouble of coordinating partial solutions handled by subagents. The objective agents also take responsibility of satisfying constraints, but not the full set of constraints. In this way, CGAS stands a chance to find a potentially good solution. In particular, a hybrid meta-heuristic technique—hybrid Ants System and Tabu Search—is introduced as optimization technique for objective agents. By adjusting the relative importance of the two algorithms, the sub-models cater to problems with different nature.

We apply our two systems to Inventory Routing Problem with Time Window. Solutions were compared with the existing work, and we showed that our solution quality was much better than the given benchmark results. In addition, CGAS gives a better performance in solution quality and FGAS gives a better performance in running time.

Further work could be conducted on the following directions:

1) One motivation of HASTS is code economy. Adjusting the relative importance of two components and getting four derived models with different strength is much easier than writing four different pieces of code. More works could be done on proposing solving models following that idea. In MOMCP, different natured sub problems need different optimization technique. It is highly code economic if the derived models of a solving model cater for different objectives in MOMCP respectively.
2) More work could be done on introducing more effective negotiating strategies and further enhance the performance of FGAS.

Reference

[Armstrong and Durfee 1997] Armstrong, A. and E. Durfee. *Dynamic Prioritization of Complex Agents in Distributed Constraint Satisfaction Problems.* P620-625.In Proceddings of the Fifteenth International Joint Conference on Artificial Intelligence,1997.

[Ben-Tal, 1979] A. Ben-Tal, *Characterization of pareto and lexicographic optimal solutions*, in Multiple Criteria Decision Making Theory and Application, Fandel and Gal, Eds., pp. 1--11. SpringerVerlag, 1979

[Chainbi et al. 2001] W. Chainbi, A. Ben-Hamadou, and M. Jmaiel. *A belief-goal-role theory for multiagent systems.* International Journal of Pattern Recognition and Artificial Intelligence, 15(3):435–450, 2001.

[Deb et al. 2000] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II.* In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, Parallel Problem Solving from Nature -- PPSN VI, pages 849-858. Springer Verlag, 2000.

[Fonseca and Fleming 1993] Carlos M. Fonseca and Peter J. Fleming. *Genetic algorithms for multiobjective optimization: Formulation*, discussion and generalization. In Stephanie Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, pages 416-423, San Mateo, California.Morgan Kaufmann, 1993.

[Franklin and Graesser, 1996] Franklin, S. and Graesser, A., *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, SpringerVerlag, 1996.

[Gambardella et al. 1999] L. M. Gambardella, E. Taillard, and G. Agazzi. *MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows.* In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization, pp. 63-76. McGraw Hill, 1999.

[Goldberg 1989] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[Hajela and Lin 1992] P. Hajela and C.-Y. Lin. *Genetic search strategies in multicriterion optimal design. Structural Optimization*, 4, pp. 99-107, 1992

[Hans, 1988] Hans, A.E.(1988) *Multicriteria Optimization for highly accurate systems*. Multicriteria Optimization in Engineering and Sciences, W. Stadler(Ed.), Mathematical concepts and methods in science and engineering, 19, 309-352. New York: Plenum press.

[Jung et al. 2001] H. Jung, M. Tambe, and S. Kulkarni. *Argumentation as distributed constraint satisfaction: Applications and results*. In Proceedings of the International Conference on Autonomous Agents, 2001.

[Jung and Tambe 2003] Hyuckchul Jung, Milind Tambe. *Performance Models for Large Scale Multiagent Systems: Using Distributed POMDP Building Blocks*, In Proc of Autonomous Agents and Multiagent Systems, 2003.

[Knowles and Corne 1999] J. D. Knowles and D. W. Corne. *The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimization.* In Congress on Evolutionary Computation (CEC99), volume 1, pages 98-105, Piscataway, NJ. IEEE Press, 1999.

[Kursawe 1991] Frank Kursawe. *A variant of evolution strategies for vector optimization.* In H.-P. Schwefel and R. M anner, editors, Parallel Problem Solving from Nature, pages 193-197, Berlin. Springer, 1991.

[Lao 2002] Lao Yizhi, *A Multiagent based approach to the inventory Routing Problem*, National University of Singapore, School of computing, 2002

[Lau et al. 2000] H. C. Lau, A. Lim, and Q. Z. Liu. *Solving a Supply Chain Optimization Problem Collaboratively*. Proc. 17th National Conf. on Artificial Intelligence (AAAI), 780-785, 2000

[Lau et al. 2002] H. C. Lau, H. Ono, and Q. Z. Liu. *Integrating Local Search and Network Flow to Solve the Inventory Routing Problem.* Proc. 19th National Conf. on Artificial Intelligence (AAAI), 9-14, 2002.

[Lerman and Shehory 2000] K. Lerman and O. Shehory. *Coalition formation for largescale electronic markets.* Proceedings of the International Conference on Multi-Aent Systems, 2000

[Li and Lim 2001] H. Li and A. Lim. *A Metaheuristic for the Pickup and Delivery Problem with Time Windows.* 13th IEEE Int'l Conf on Tools with Artificial Intelligence (ICTAI), 2001

[Liu and Tang 2002] J. Liu, J. Han, and Y. Y. Tang. *Multiagent oriented constraint satisfaction. Artificial Intelligence,* 136(1):101–144, 2002.

[Modi et al. 2003] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. *An asynchronous complete method for distributed constraint optimization.* In Proc of Autonomous Agents and Multiagent Systems, 2003.

[Rao, S.S. 1991] *Optimization theory and application*. New Delhi: Wiley Eastern Limited, 1991.

[Rao and Georgeff 1995] A. S. Rao and M. P. Georgeff. *BDI-agents: from theory to practice*. In Proceedings of the First Intl. Conference on Multiagent Systems, pages 312–317, San Francisco, 1995. AAAI Press.

[Rudolph and Agapie 2000] G. Rudolph and A. Agapie. *Convergence properties of some multi-objective evolutionary algorithms.* In Congress on Evolutionary Computation (CEC 2000), volume 2, pages 1010-1016, Piscataway, NJ. IEEE Press, 2000.

[Schaffer 1985] J. David Schaffer. *Multiple objective optimization with vector evaluated genetic algorithms*. Proceedings of an International Conference on Genetic Algorithms and Their Applications, pages 93-100, Pittsburgh, PA. 1985.

[Schulze and Fahle 1997] J. Schulze, and T. Fahle, *A Parallel Algorithm for the Vehicle Routing Problem with Time Windows Constraints*, 1997.

[Silverman 1986] B. W. Silverman. *Density estimation for statistics and data analysis.* Chapman and Hall, London, 1986.

[Shehory et al. 1999] O. Shehory, S. Kraus, and O. Yadgar. *Emergent cooperative goal satisfaction in large-scale automated-agent systems.* Artificial Intelligence, 1999.

[Srinivas and Ddb 1994] N. Srinivas and K. Deb. *Multiobjective optimization using nondominated sorting in genetic algorithms.* Evolutionary Computation, 2(3), 221-248, 1994.

[Stone and Veloso 1997] P. Stone and M. Veloso, *Multiagent Systems: A Survey from a Machine Learning Perspective*, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA CMU-CS-97-193, 1997.

[Stutzle and Dorigo 1999] T. Stutzle and M. Dorigo, AS Algorithms for the Traveling Salesman Problem. In Evolutionary Algorithms in Engineering and Computer Science, pp. 163-183, Wiley, 1999.

[Yokoo et al. 1992] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara. *Distributed constraint satisfaction for formalizing distributed problem solving*. In Proceedings DCS, pages 614 621, 1992.

[Yokoo et al. 1998] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. *Distributed constraint satisfaction problem: Formalization and algorithms.* IEEE Transaction on Data and Knowledge Engineering, 10:673--685, 1998

[Yokoo and Hirayama 1998] M. Yokoo and K. Hirayama. *Distributed constraint satisfaction algorithm for complex local problems.* In Proceedings of the Third International Conference on Multiagent Systems (ICMAS-98), pages 372-379, Paris, France, 1998.

[Zitzler and Thiele 1999] E. Zitzler and L. Thiele. *Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach.* IEEE Transactions on Evolutionary Computation, 3(4), 257-271, 1999.

[Zitzler et al. 2001] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm.* Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.