CONSTRAINT SATISFACTION IN "CONSCIOUS" SOFTWARE AGENTS – A PRACTICAL APPLICATION

Arpad Kelemen[1,2,3], Stan Franklin[1], Yulan Liang[3]

[1]University of Memphis

[2]Niagara University

[3]University at Buffalo

Mailing Address: Arpad Kelemen
249 Farber Hall
3435 Main Street
Buffalo, NY 14214, U.S.A.
Phone: 1-716-829-2715
Fax: 1-716-829-2200
Email: akelemen@buffalo.edu
URL: http://www.msci.memphis.edu/~ida

Abstract

The U.S. Navy has been trying for many years to automate its personnel assignment process. Periodic assignment of personnel to new jobs is mandatory according to Navy policy. Though various software systems are used regularly, the process is still mainly done manually and sequentially by Navy personnel, called detailers. This paper is a case study in applying cognitive theories implemented via new AI techniques to produce flexible adaptive human-like software. In it we present a sketch of an Intelligent Distribution Agent that aims to largely automate the detailer's tasks. In particular, we look inside its constraint satisfaction, which is mainly responsible for satisfying the requirements of the Navy's policies, the command's needs and the sailor's preferences in order to achieve "satisfactory" job assignments according to ever evolving Navy standards.

Key words: Job assignment, distribution, constraint satisfaction, matching, scheduling, intelligent agent, consciousness.

Introduction

Currently there are more than 400,000 enlisted personnel in the U.S. Navy. Every enlisted person needs to be reassigned to a new job periodically (typically every two to five years). Each year more then 100,000 Navy personnel are assigned to new jobs at a cost of some $600 million dollars just for moving expenses. Other costs apply, too. In general, distribution is the process whereby personnel managers direct the movement of individuals to fill vacancies in field activities. The distribution of sailors is done by some 280 Navy personnel called detailers. (Distribution of highly ranked officers is done through a more complex, but similar process. Automation of officer distribution is not a current goal of the US Navy and is not going to be discussed further.) Each detailer makes assignments for a community of typically between 1,000 and 5,000 sailors. A community consists of sailors of the same rating, for example: Aviation Support Equipment Technicians (AS). A detailer's task is quite complex, including reading and understanding email messages written in natural language, handling phone calls, searching personnel databases, searching for jobs for sailors, trying to satisfy various constraints, negotiating with sailors and other agents (human and non-human) (Franklin and Graesser, 1997) producing orders, writing email messages, and so on. Our project aims to automate most - if not all - of the tasks performed by a human detailer using a "conscious" software agent (Baars, 1988, 1997; Franklin, Kelemen and McCauley, 1998; Ramamurthy, Bogner, & Franklin, 1998; Bogner, 1999; Kelemen, 2002; Kelemen et al., 2002). Our agent has modules for perception (natural language understanding), working

and associative memories, emotions, learning, cognition, metacognition, "consciousness", constraint satisfaction, action selection, deliberation, and negotiation by means conversing with sailors and others via email in natural language. We have designed and implemented a "conscious" software agent called Intelligent Distribution Agent (IDA) (Franklin, Kelemen and McCauley, 1998; Kelemen, 2002; Kelemen et al., 2002) who is intended to do the job of a human detailer efficiently while modeling human cognition.

The job assignment problem of other military branches may show certain similarities to that of the Navy, but the Navy's mandatory "Sea/Shore Rotation" policy makes it unique and perhaps, more challenging than other typical military, civilian, or industry types of job assignment problems. Unlike in most job assignments, the Navy sends its sailors to short term sea and shore duties periodically, making the problem more constrained, time demanding, and challenging. This was one of the reasons why we designed and implemented a complex, computationally expensive, human-like "conscious" software. This software is completely US Navy specific, but it can be easily modified to handle any other type of job assignment, yet as we will see later, it may not always be advisable.

After a walk through of one of IDA's typical cycles to orient the reader, we turn our interest to the module which is largely responsible for making decisions with regard to finding suitable jobs for sailors. This module is called the constraint satisfaction module. It not only applies standard operational research techniques but also some ideas from psychology and other related fields.

IDA

IDA is a "conscious" software agent, which has been designed and implemented by the Conscious Software Research Group at the University of Memphis. "Conscious" software agents (Franklin, Kelemen and McCauley, 1998; Ramamurthy, Bogner, & Franklin, 1998; Bogner, 1999; Kondadadi, Dasgupta and Franklin, 2000; Kelemen, 2002; Kelemen et al., 2002) are software agents that implement Baars' global workspace theory, a psychological theory of consciousness (Baars, 1988; 1997). According to global workspace theory the mind's architecture includes many small, specialized processes, which are normally unconscious, and a global workspace (or blackboard).

Baars used the analogy of a collection of experts, seated in an auditorium, each of whom can solve some problem or problems. It is not known who can solve a given problem. In global workspace theory somebody makes the problem available to everyone in the auditorium by putting it on the blackboard. An expert on this particular problem can identify and solve the problem. One of the main functions of consciousness is to recruit resources to deal with novel or problematic situations. When a novel situation occurs, an unconscious process tries to broadcast it to all other processes in the system by trying to put it on the blackboard, which causes it to become conscious. Only one problem can be on the blackboard at a time. Therefore consciousness is a serial system. These processes are embedded into every module other than the consciousness module. These are the places where new situations may arise and therefore consciousness may be needed to deal with them.

We implement these unconscious processes as codelets borrowing the term from Mitchell and Hofstadter (1991) and Mitchell (1993). A *codelet* is a small piece of computer code capable of performing some basic task (analogous to an expert in the auditorium). Information codelets are codelets, who carry primitive pieces of information. An attention codelet is a codelet that "pays attention" to a particular type of event. When that event happens it collects the right information codelets, and they try to make it to "consciousness" together in order to broadcast the situation to other codelets.

At any given time, different processes (attention codelets) could be competing with one another, each trying to bring its information to "consciousness". These codelets compete in a place called the playing field. Codelets may form coalitions depending on the strength of associations between them. The activation (different from association) of a codelet measures the likelihood of it becoming "conscious". The coalition with the highest average activation finds its way to "consciousness." In the event of a tie the winner is randomly chosen.

Figure 1 shows the general architecture of IDA. An arrow from module 1 to module 2 means that execution of module 1 can trigger the execution of module 2, of course via codelets. Note that two modules are connected to almost all the other modules: the behavior net and the consciousness modules. The behavior net is the agent's action selection module and will be discussed later (Maes 1990, 1992; Song and Franklin, 2000). The "consciousness" module, as we discussed above, provides novel problem solving capabilities to the agent. The modules in the second line typically (but not necessarily) get executed sequentially (from left to right). These modules are not

directly connected and therefore don't directly trigger one another. Trigger happens through consciousness and (careful) action selection.

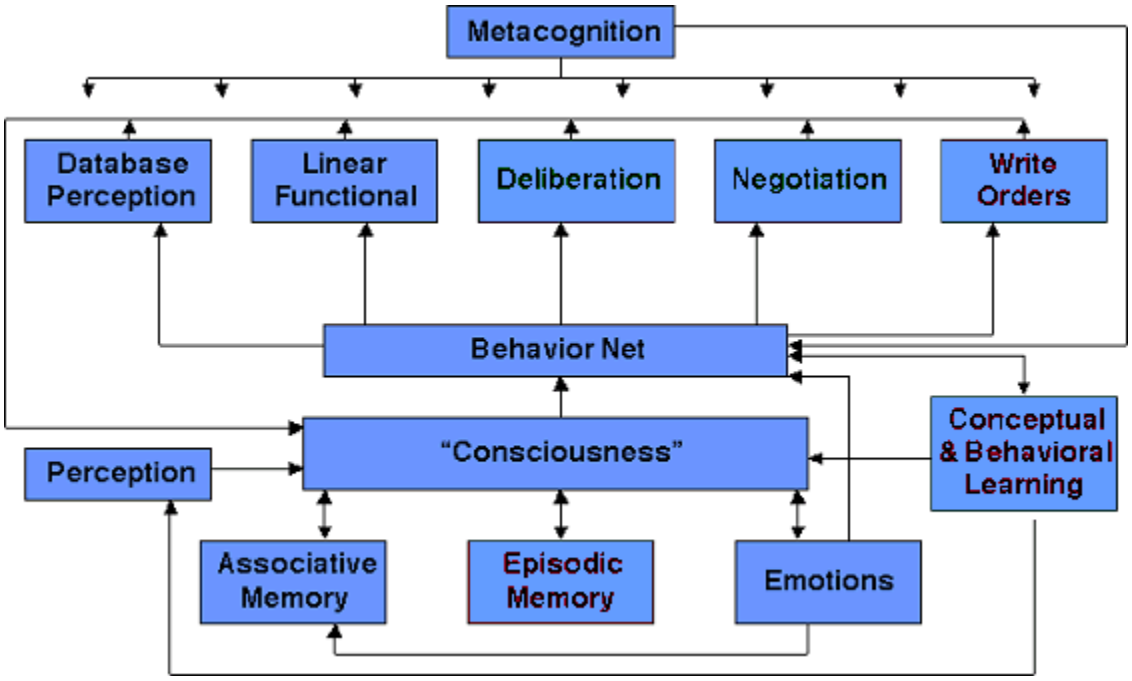

Figure 1. The IDA Architecture

Thread through IDA

To best demonstrate the actual work of IDA let's go though briefly a single episode beginning with an email message from a sailor requesting the start of an assignment process and ending with the composition of a reply message offering the sailor a choice of one, two, or occasionally three jobs. We'll refer to this process as a thread through IDA.

The following thread was implemented, tested and presented to Navy officials.

THREAD BEGIN

Suppose the following email message comes from a sailor:

“Date: Tue, 10 Jun 2003 16:53:23 +0000

From: LEGAULT JOHN DOE <doe@navy.us>

Subject: SSN: 123456789

IDA,

I am AS1 LEGAULT JOHN DOE. Please find me a job.

Thanks,

AS1 LEGAULT JOHN DOE”

The following pieces of data will be perceived from the given email message:

- message date

- sailor’s name

- sailor’s SSN (Social Security Number)

- the sailor is looking for a new job

- sailor’s contact information

IDA uses the Copycat architecture (Hofstadter & Mitchell, 1994) for natural language understanding. Copycat has been designed for analogy making. The knowledge, that IDA needs to understand different concepts in a message such as the

sailor's name, social security number (SSN), idea type, etc., are stored in the form of a slipnet (Mitchel, 1993), a knowledge representation structure similar to a semantic net. When IDA receives a message from a sailor, the perception codelets analyze the message and look for patterns, which could be similar to a name, a SSN, other sailor particulars, and an idea-type. Percepts then are stored in perception registers, a part of the focus. The focus is part of the workspace (short term memory), which is used to write to and read from the long-term memory. IDA's long term memory is implemented as a Sparse Distributed Memory (SDM) (Kanerva, 1988). SDM is an associative memory with statistical features that make it perform much like a human memory. For example, as unrehearsed data grows old in SDM, access to it may become difficult, or temporarily impossible. On the other hand, interesting (and important) new associations may be formed that is the basis of human learning as well as in our agent. A read (retrieval of associations) from SDM happens every time the focus is updated, using part of the focus chosen based on activation levels. We use a mechanism that uses probabilities and thresholds. Activation levels are set to 1 for each new item written to the focus. Activations for all the fields in the focus are decaying according to a decay mechanism. The range of the activation levels is [0, 1]. Associations coming back from SDM are written to the second layer of the focus, so they don't destroy the content that was used to read SDM.

After the associations from the long term memory (SDM) are placed in the focus (and therefore the workspace), the perception registers activate some nodes in the slipnet and finally, depending on the perceived idea type, a behavior (action selection) template is chosen and filled with the information obtained in the message. In this case, IDA

would perceive the idea “find job” and would choose an existing template, called “find job” behavior template.

Information and attention codelets sitting in the stands (auditorium) see the message in the workspace and instantiate copies of themselves that get the information from the message. At the same time an attention codelet sitting in the stands looking at the workspace recognizes the idea of the new message, gathers the correct information and attention codelets, increases its activation, and they all jump to the playing field.

The coalition manager (Bogner, 1999) forms a coalition of codelets based on associations. Some initial associations are built in, while others are learned. Hopefully the codelets from the previous paragraph will form a coalition. The spotlight controller (Bogner, 1999) selects the coalition with the maximum average activation, and shines the spotlight on it. In other words the codelets in the spotlight are the ones whose contents are in “consciousness”. The broadcast manager (Bogner, 1999) broadcasts all the information carried by the codelets in the spotlight. After their information is broadcast, all the codelets in the spotlight leave the playing field.

Some behavior priming codelets (Kelemen, 2002; Kelemen et al., 2002) sitting in the stands find the broadcast relevant, bind their variables using the information in the broadcast and jump to the sidelines, a special part of the playing field. Using the auditorium analogy, the behavior priming codelets are the experts who identify the problem and know how to solve it, but they don’t solve it themselves, instead they initiate the problem solving process.

Some coalition of codelets instantiates an appropriate goal structure (unless it's already instantiated), binds as many variables as it can, and sends activation

(environmental link) to the initial behavior of the goal structure. A goal structure is a chunk of IDA's behavior network (Maes, 1990, 1992, Song and Franklin 2000; Kelemen 2002), which consist of linked behaviors, environmental states, goals and drives (Franklin, 1995). Each goal structure is trying to satisfy a drive or a goal. In the example thread, the first goal structure will look up the sailor's personnel record.

The behavior net eventually chooses and executes the initial behavior of the goal structure, causing it to instantiate codelets to create an SQL query to access the sailor's personnel record for the relevant fields, one field at a time. Behaviors get performed by codelets, called behavior codelets. The task of one behavior is typically done by one to four behavior codelets.

After the first behavior is selected an emotion and an action are written to the focus. With the current content of the focus, taking decay into consideration, IDA writes to SDM. The result of this query, the content of a single field, is copied to the corresponding fields in the workspace and the focus (internal perception). This triggers another SDM read of associations with this data. Attention codelets provoke another broadcast containing these contents. The answering behavior priming codelets send activation eventually causing another behavior in the goal structure to execute another query, yielding the content of another field that is again written to the workspace and the focus. This process continues until all the required fields are written into the workspace.

Then a different goal structure recovers information from the requisition list (current items from the job database). Upon execution this goal structure instantiates codelets to create SQL queries to access the relevant fields with matching Navy Enlisted Classification (NEC) (job qualifications) and matching paygrade resulting in a coarse

selection of dozens of possible jobs. The same process through "consciousness" and the behavior network is used to retrieve each field. Again, the results of these queries are copied to the corresponding fields in the workspace (internal perception), with the requisite reads from and writes to long-term memory.

Yet another goal structure sends in turn each job from the coarse selection to the linear functional (part of the constraint satisfaction module) to receive its fitness value (Kelemen 2002; Kelemen et al., 2002). The process is again the same. As a result, the fitness values will be written to the workspace beside the corresponding jobs. Fitness values are assigned to sailor-job pairs reflecting degrees of potential matches. The higher the fitness value, the better the match is between sailor and job.

Deliberation

An attention codelet watching the workspace notices that all the jobs listed there have fitness values assigned to them. It picks the job with the highest fitness value, gathers information codelets for the job, jumps into the playing field, forms a coalition and, hopefully, gets its message broadcast. Behavior priming codelets in the stands find the broadcast relevant, jump to the sidelines, and instantiate a goal structure "To create a scenario for the job" (Song and Franklin, 2000; Franklin, 2000; Kondadadi and Franklin, 2001). Using our behavior net process as above, a possible detach date from the sailor's current job is calculated and written to the workspace by a behavior codelet using the Projected Rotation Date. (The Projected Rotation Date is the date a sailor has to report to his/her new job and each job comes with such date.) Another attention codelet notices

this detach date and starts the process that adds leave time to the scenario. Similarly, travel time and proceed time (if needed) are added. The gap between the date the job should be filled in and the date the sailor would report to the job is calculated, and the scenario building process is started anew if needed.

This process of scenario building continues until an appropriate number of jobs are selected for offer to the sailor, or until there are no more jobs with sufficiently high fitness about which to build a scenario.

Final selection according to Ideomotor Theory

The players in this part of the drama are attention/emotion codelets who have preferences for jobs with certain attributes. One might be concerned with keeping moving costs down, another with job priority, and still another with the size of the arrival gap. Some might be concerned with two of these attributes. Another player is the timekeeper codelet to be described below.

At any time after at least one scenario has been completed, one of these attention/emotion codelets can propose a particular job that has a completed scenario. To do so he gathers information about which job he is proposing and brings it to "consciousness" if he can. In the stands is the timekeeper codelet. If enough time passes without an objection to this job or without the proposal of another job, the timekeeper marks the proposed job as one to be offered. This process can continue until two, or even three, jobs have been so marked.

On the other hand, one of these attention/emotion codelets might choose to object or to propose a different job to be offered. This is done in the same way, with the attention/emotion codelet gathering information and coming to "consciousness". In the case of a new proposal, it simply replaces the old proposal, but with a reduction in the timekeeper's patience, and is handled in the same way. An objection, unless soon followed by a new proposal, or a reproposal of the original (with less activation to get to "consciousness"), results in another scenario initiated by the attention codelet who has been competing with proposers for "consciousness".

After a proposal is accepted, the process continues until three are accepted, or until no more scenarios are being created and the timekeeper is out of patience (James, 1890; Franklin 2000; Kondadadi & Franklin, 2001). He then calls a halt, setting a flag to this effect in the workspace.

Creating a message, offering a job (language generation module)

The appropriate attention codelet notices the end of job selection flag and begins the message generation process by gaining "consciousness" with the appropriate information codelets (sailor's name, paygrade, etc).

The same action selection process recruits a codelet who writes the first paragraph of the message, the appropriate salutation, into the workspace. Another attention codelet gains "consciousness" with the salutation and with the number of jobs to be offered. The same action selection process results in the second paragraph, the introduction, being written into the workspace. This whole process repeats until paragraphs for each job to

be offered and a closing paragraph are written into the workspace (the message). IDA signs her name, copies the appropriate content of the workspace into an email body, fills out the remaining fields of the email and sends the email message using the same action selection process.

THREAD END

This completes our description of the thread. With constraint satisfaction's place in the process laid out, we turn to a detailed description.

Constraint Satisfaction in IDA

Constraint satisfaction in IDA happens in three separate phases. The first is called coarse selection, which happens when IDA is looking for jobs for a given sailor using certain criteria. If a job doesn't match all the criteria specified in the SQL query then it doesn't pass this phase. IDA uses job qualification match (NEC match), meaning that the sailor must have at least one trained skill required by the job. IDA also uses paygrade match meaning that the sailor's paygrade can't be off by more than one from the job's preferred paygrade. IDA uses the Navy's Assignment Policy Management System's (APMS) sailor and job databases, which include up to date detailed data on sailors and available jobs. Various additional sailor community specific criteria are also in place. Depending on the community and the paygrade of the sailor a handful to hundreds of jobs

pass the coarse selection. Once the coarse selection is over the second phase of the constraint satisfaction takes place.

The second phase of the constraint satisfaction is done with a linear functional. As we will see below a functional assigns normalized fitness values for each job in the workspace for the given sailor at the given time considering several additional constraints. Once the linear functional has done its job, each job in the workspace will have a fitness value assigned to it.

Then the third phase of the constraint satisfaction takes place, what we call deliberation. In the deliberation module, as we have seen above, temporal scenarios are created for the high fitness jobs. On the basis of these scenarios jobs are proposed and some of them may be accepted for offering to the sailor. Issues like timing, costs, feasibility, and training are taken care of in the deliberation module. Emotions also play an important role in this module. For example, if a sailor makes IDA mad, the timekeeper may run out of patience earlier resulting in fewer jobs with scenarios, and therefore fewer jobs to be offered to that sailor.

The Linear Functional

Though the Navy has about 600 constraints (issues), about 500 of them apply only to small groups of jobs/sailors and are seldom considered by a detailer. The remaining, approximately, 100 constraints bear very different importance, and different detailers may consider different ones more important then others, even if handling the same

community. Changing the community may largely change the importance of constraints, and as time goes by there is a natural change in their relevance too.

We distinguish three levels of constraints in IDA: hard, soft, and semi-hard. A hard constraint always has to be satisfied and is such that a sailor's eligibility can't be earned before the sailor's new assignment. If it is not satisfied, then the sailor is not eligible for that job. A soft constraint does not have to be satisfied, though desired. A semi-hard constraint has to be satisfied, but eligibility can be earned before the sailor's new assignment. Decisions on whether individual constraints should be hard, soft, or semi-hard were made by us based on input from Navy detailers. These decisions were relatively straightforward. We will see examples for each later.

Similarly, the constraints can also be divided into three classes according to whose interest they serve: sailor, command, and Navy constraints.

Some of the sailor constraints are:

- Sailor wants to be stationed in a certain place
- Sailor wants to serve on a specific ship/command
- Sailor wants training
- Sailor doesn't want training
- Sailor wants to be at sea
- Sailor wants to be on shore
- Sailor wants promotion opportunity
- Sailor wants schooling for his child, etc.

The first three are the most typical requests from sailors and are the most respected by detailers.

Some of the command constraints are:

- Sailor has appropriate job skills
- Job priority (some jobs are more important to be filled than others at a given time)
- Sailor is in a certain age group
- Paygrade (Sailor's paygrade has to match the job's paygrade)

Some of the Navy constraints are:

- Keep down moving costs
- Geographic Location (it is better to assign a sailor in the same geographic location where he currently serves)
- It is better to assign a sailor to a job with his existing job skills without needing further training
- Manning Control Authority (The Navy sets a quota for the number of enlisted people for the Pacific and Atlantic coast and it's bad to go above or below it. If the difference between the Pacific and Atlantic manning is larger than 5% then it's better to assign a sailor to the coast with lower manning.)

To consider all the constraints simultaneously we need a common currency function, a function which compares time with money, sailor preferences with Navy policies and so on. To account for all the constraints simultaneously we apply a linear

functional (a convenient abuse of language with the hard constraints as products) with the following form:

$$F(\underset{\sim}{x}) = (\prod_{i=1}^{m} c_i(\underset{\sim}{x_i})) * \sum_{j=1}^{n} a_j f_j(\underset{\sim}{x_{m+j}}) \ ,$$

where

$\underset{\sim}{x_i}$ (i=1,…,m+n) are input vectors of variables, which are not necessarily disjoint in their variables and $\underset{\sim}{x}$ is the union of all the variables occurring in all the $\underset{\sim}{x_i}$ s

c_i (i=1,…,m) is a Boolean function for the i'th hard constraint; it yields 1 if the corresponding hard constraint is satisfied, otherwise it yields 0.

f_j (j=1,…,n) is a function for the j'th soft constraint; it yields values in [0,1] (the closed unit interval of real numbers)

a_j (j=1,…,n) is a coefficient, constant in [0,1] (the closed unit interval of real numbers) for the j'th soft constraint.

The equation

$$\sum_{j=1}^{n} a_j = 1$$

guarantees that F will yield values in [0,1]. This will also be useful later for tuning purposes (the actual setup of the a_j coefficients).

Every f_j is a function which measures how well the j'th soft constraint is satisfied if we were to offer the given job to the given sailor at the given time. These functions are monotonic, but not necessary linear, however often they are. The setup these functions was based on community specific knowledge, up to date Navy policies, and common sense. Detailers, who handle communities are the most useful source of information.

Every a_j tells how important is the j'th soft constraint in relation to the others. This gives the common currency property, where we can specify which constraint we care about most at a given time frame.

Note that if a hard constraint is satisfied then it doesn't change the value of F, but if it is not then F=0, which means that the sailor doesn't match the given job, so he/she is not eligible for it. F=1 would be the ideal job for the sailor, which practically never happens due to the long functional and the sometimes self-contradicting constraints.

After the sailor's particulars are in the workspace behavior codelets of the linear functional module calculate all the function values for each of the constraints for one job and write them to the workspace, where other modules can access it as well. Once all the values are in place a different behavior codelet calculates and writes the final fitness to the workspace. The process continues until all the jobs have fitness values assigned. Once each job is assigned a fitness value, the deliberation module may build temporal scenarios on high fitness jobs over threshold in order to find jobs for final offering to the sailor. Note that different sailors in the same community are subject to the same F functional at a given time frame. Different communities have different F functional.

Implementation of IDA

IDA was implemented in Java and was successfully tested on machines with a minimum of 333MHz Pentium-II processor and 64 MB RAM. The newer, faster machines provided better performance in terms of running time, but not in terms of problem solving skills. To access real sailor and job information IDA used the Navy's APMS job and sailor databases.

IDA was implemented for several communities. The following constraints apply to all communities IDA currently considers. They were all implemented in the linear functional module. (Constraints are listed on the left side, and the corresponding behavior codelets with the functions they implement are in parenthesis on the right side.)

Hard Constraint: **Behavior codelet:**

1. Sea/Shore Rotation Match (CalculateSeaShoreMatch.class, c_1 function)

According to Navy policy if a sailor served on shore then his next assignment must be on sea. This codelet, if executed, returns 1 if the considered job satisfies the "sea/shore rotation" policy. Otherwise it returns 0.

2. Dependents Match (CalculateDependentsMatch.class, c_2 function)

This codelet, if executed, returns 1 if the considered sailor and job satisfies the "No more than 4 dependents to overseas location" policy. Otherwise it returns 0.

Soft Constraint:

1. Job Priority Match (CalculateJobPriorityMatch.class, f_1 function)

Each job in the job database has a priority. The higher the job priority the more important it is to fill in the job. Accordingly, the function assigns f_1=1 if the job priority is 1, and is linearly decreases to 0 for job priority 101. Jobs with priority 102 or above get f_1=0 assigned.

2. Order Cost Match (CalculateOrderCostMatch.class, f_2 function)

This is the moving cost. The lower the cost the better the match is. If the moving cost is $0 then f_2=1, and it linearly decreases to f_2=0 for $100,000, and stays 0 over $100,000. Moving costs are automatically calculated by the Autocost software, which was provided by the Navy and was integrated with IDA. Autocost calculates moving costs from Area Type Codes (ATC), paygrades, and the number of dependents.

3. Location Preference Match (CalculateLocationPreferenceMatch.class, f_3 function)

The sailor's primary, secondary and tertiary preferences are kept for sea, shore and overseas locations in APMS using Area Type Codes. This function checks if the ATC of the sailor's preference matches the considered job's ATC. If the primary preference is matched then f_3=1, if the secondary then f_3=0.67, if the tertiary then f_3=0.33, otherwise f_3=0.

4. NEC Reutilization Match (CalculateNECReutilizationMatch.class, f_4 function)

Up to five Navy Enlisted Classification codes (skills acquired via training) are kept in Navy databases about enlisted people. Jobs may ask for up to two NECs. No sailor can get a job without an NEC required by the job, though having both is not preferred. This function assigns f_4=1 if the job's first NEC matches the sailors first NEC, f_4=0.9 if the jobs second NEC matches the sailor's first NEC, f_4=0.8 if the jobs first NEC matches the sailors second NEC, and so on. If only the job's second NEC matches the sailor's fifth NEC then f_4=0.1, and f_4=0 if there is no match. Note that the NEC Reutilization Match is a soft constraint therefore it can be violated. However, a different NEC consideration, called NEC Match was already taken care of in the coarse selection therefore no jobs without NEC Match would be considered by the linear functional in IDA.

5. Paygrade Match (Soft) (CalculatePaygradeMatch.class, f_5 function)

This function assigns f_5=1 if the sailor's paygrade equals the job's paygrade, f_5=0.5 if the sailor's paygrade is off by one and f_5=0 otherwise. Note that a hard version of the paygrade match was applied in the coarse selection, so jobs whose paygrade is off by more than one from the sailor's paygrade are not considered by the linear functional.

6. Geographic Location Match (CalculateGeographicLocationMatch.class, f_6 function)

This function assigns f_6=1 if the sailor's current ATC is the same as the proposed job's ATC and f_6=0 otherwise.

Various other community specific constraints have been implemented in IDA. As we have seen above IDA applies a minimum of two hard constraints in the coarse selection, two hard and six soft constraints in the linear functional. To settle which hard constraints are considered in the coarse selection and which in the linear functional we followed cognitive modeling of detailers and cost efficiency. Note that even for such a small set of constraints like the above some overlap applies. The Order Cost Match and the Geographic Location Match are highly correlated, but it's not so obvious. Although moves within an ATC tend to be cheap, some ATCs are large. Therefore the cost of a move within an ATC sometimes can be significant.

In practice the vast majority of jobs get discarded because of a failing hard constraint in both the coarse selection and the linear functional. On the other hand soft constraints by themselves practically never make a fitness value 0. To get a fitness value of 1 is practically impossible. The deliberation module will consider jobs with fitness over threshold, which is currently set to 0.35. Please note that the three phases of constraint satisfaction could be done in one, but for both cognitive modeling and task efficiency we choose not to do so. The three significantly different phases also give us a better overview and control over the agent. Finally, due to their delicate nature, semi-hard constraints are all handled in the third phase of constraint satisfaction: in the deliberation module.

Table 1. shows the data of a sailor which is used by the constraint satisfaction. The data contains the followings: SeaShore code, NEC1-5, Number of dependents, Sea Location Preference 1-3, Shore Location Preference 1-3, Overseas Location Preference 1-3, Area Type Code, Paygrade respectively.

1	7606	7609				1	GMY	GMY	FNO	GPS	GKE	GMY	BER	CUB	GUM	ELA	6

Table 1. Sailor's data

Table 2. shows seven examples for job data which are used by IDA. The data contains the followings: SeaShore code, Area Type Code, Paygrade, NEC1-2, Job Priority, value for c_1, c_2, f_1-f_6, functions, and finally F, the overall fitness value. Note that $a_1=a_2=a_3=a_4=0.2$ and $a_5=a_6=0.1$ coefficients were used for demonstration purposes.

2	FNO	6	7612		1	1	1	1	0.91	0.33	0	1	0	0.548
2	FNO	6	7617		2	1	1	0.99	0.88	0.33	0	1	0	0.54
2	FNO	6	7618		5	1	1	0.96	0.91	0.33	0	1	0	0.54
2	ING	6	7618		4	1	1	0.97	0.85	0	0	1	0	0.464
2	FNO	6	7609		8	1	1	0.93	0.95	0.33	0.8	1	0	0.702
2	FNO	6	7606		7	1	1	0.94	0.93	0.33	1	1	0	0.74
3	ICE	6	7609		10	1	1	0.91	0.74	0	0.8	1	0	0.59

Table 2. Job's data, function values, and fitness values

"Conscious" Constraint Satisfaction in IDA

IDA currently has two versions of the second phase of the constraint satisfaction module: an unconscious and a "conscious" one. In the case of the unconscious linear functional only one attention codelet goes to "consciousness", the one, which realizes that the perception module has finished its job. In the "conscious" version each needed sailor and job data item goes to "consciousness" as well as the value for each hard constraint, soft constraint and the final fitness for each job.

The linear functional module can be viewed as an external module from a cognitive agent, because it's a little bit hard (but not impossible) to find it's true parallel in a human agent who calculates functions, and applies coefficients to come up with a normalized real value to measure the "fitness" of a possible job for a given sailor at a given time. On the other hand every detailer must have his own judgment to consider various constraints in order to decide which jobs are suitable for a sailor. A detailer may make such decisions unconsciously, but usually it is a conscious decision. This is comparable to a linear functional approach of the constraint satisfaction problem in IDA.

Because of the more than 300 parallel running java threads taking up resources, the long delay cycles in some of them to conform to the "consciousness" module, the large sized emotion network, the usage of a graphical user interface for demonstration purposes, and various other reasons, running the presented IDA thread is very time consuming. A typical run on a 333MHz 64MB PC takes about 30 minutes with the unconscious linear functional and 40 minutes with the "conscious" one. On a 1GHz,

1GB dual processor computer these times were improved to about 8 minutes for the unconscious, 11 minutes for the "conscious" one.

If we just start up IDA and run through the thread once then the linear functional assigns the same values weather it is "conscious" or not, and it runs through only once. Therefore, in this case, making the linear functional "conscious" wouldn't buy us anything other than cognitive modeling. On the other hand if we run IDA over time and allow her (IDA) to handle multiple, perhaps different kinds of, emails in the same run, the two versions perform differently. Consider, for example, the following situation: after an incoming message from a given sailor asking for a job, IDA has performed the above process (the thread), found two suitable jobs and offered them to the sailor. Now a new email comes from the same sailor saying that he doesn't like either of the jobs and asks why he wasn't offered a particular third one. Now IDA needs to find out why she didn't offer the job to the sailor earlier. The answer can't simply be "the fitness of the job was too low for you"; she needs to find out exactly why. If IDA did the entire constraint satisfaction unconsciously then she needs to go through it with the same data consciously, in order to find out where the sailor missed the most points and why. Note that she can only do this if she has access to the same data. Such data might be recovered from the long term memory, because it was written there during the first run (since everything that goes to the workspace goes to the SDM too). Once the reason is found, the negotiation module can produce the proper answer with the main reason or reasons. Notice that this highly resembles what a human detailer would do; therefore for cognitive modeling it is the right thing to do. However the issue of efficiency requires more discussion. The cost of "consciousness" is high, so avoiding it for the first pass seems to be justified. For the

second pass the same unconscious module would never produce a detailed answer, only the fitness, which is not enough to answer the sailor's second question.

As opposed to the previous two-pass system, in a one pass system where the constraints satisfaction (in particular the linear functional) always happens consciously, the needed data might be written to long term memory, and could be recovered later from there, but not with absolute certainty (just like from a human memory). To do so, a "very conscious" constraint satisfaction is necessary, which is not only "conscious" of the data, and the fitness, but also of each of the c_i hard constraints and the f_i soft constraints for each job as well as the a_i coefficients. A further question about the two-pass constraint satisfaction is how to reconstruct the same data for the second pass. By the time IDA is about to do the "conscious" constraint satisfaction some of the data might be gone. The sailor's data is probably still available with no change in the database, but the job data might have lost some significant piece of information, for example the job priority or even the job itself. To recover such data might be possible from the agent's own long term memory. Whether IDA tries to recover data from a database or from long term memory the probability of recovering the data grows less with time. If IDA receives the sailor's new email after a new requisition list (job list) has arrived then the old data (in the database) is lost and she no longer has access to the same data about the job in the database. On the other hand if IDA receives the sailor's second email before a new requisition list arrives, chances are that she will have access to all the data as before and therefore be able to find an answer to the sailor's question. All this means that IDA should send job offers to sailors as soon as possible to give them plenty of time to ask

about it. If the answer comes early, IDA also has a better chance of being able to recover the context from memory.

Another issue is that as time goes by the situation is no longer the same. Considering a job for a sailor now is often very different from considering it four weeks later. For example, in the third phase of constraint satisfaction IDA considers the Projected Rotation Date (PRD) as one issue, meaning that the closer the PRD the more important it is to find a job for the sailor. Such complex problems are not solved by a single module in IDA, instead by a collaboration of various modules through consciousness, emotions, associative memory, action selection, and constraint satisfaction.

A third possible way to answer the sailor's second email would require some external storage. Though on some important issues a detailer does make some notes for himself, it is inefficient and inappropriate to keep track of every single piece of datum, and every single decision he makes in files in terms of speed and memory. For cognitive modeling it is clearly not a possibility.

Missing Data

What to do if a data item about a constraint is missing? The most frequent missing data are the sailor's preferences. If the corresponding constraint is a hard constraint then IDA should ask the corresponding stakeholder (Navy, command, sailor) to provide its preference if applicable. If some other piece of information is not available

then IDA writes an email to the right office or assume that the constraint is satisfied. In the Navy, such decisions are up to the detailer. A detailer can learn how to deal with situations like this, so does IDA. If the constraint is a soft constraint then IDA assumes that the stakeholder has no preference, and the corresponding constraint is highly satisfied. A value close to 1 (0.95) indicates that the constraint is satisfied, but not as well as if it was explicitly asked for and satisfied. Another method is discussed in the next section. The version presented below handles preferences in a more flexible way. Not specifying certain preferences may emphasize the relevance of other preferences of the same stakeholder as a side effect.

Increasing Constraint Satisfaction Efficiency

Survey preference rankings

Based on feedback from all the three stakeholders (Navy, command, sailor) the linear functional can yield more accurate and up to date results. If we hold a survey where each stakeholder ranks its preferences, the coefficients can be set accordingly. The sum of a stakeholder's coefficients is prefixed and is subject to detailer or Navy supervision, and the sum of all coefficients is 1.

For example, suppose the sailor ranks his constraints as follows:

1. Sailor wants training

2. Sailor wants to be in a certain place

3. Sailor wants to be on a specific ship/command

etc.

This means that the sailor's most important constraint (preference) is to get training, his second one is to be in a certain place, etc.

Similarly the command constraints could be ranked as follows:

1. Job priority

2. Paygrade match (soft)

etc.

and the Navy constraints:

1. Moving cost

2. NEC reutilization

3. Geographic location match

etc.

Once IDA has all the rankings, she transforms them into values according to some previously agreed strategy, such as: set the values linearly decreasing from 1 to 0, where an imaginary last+first preference gets the value 0, so even the last preference/constraint

will get some positive value. Then project the transformed values to the prefixed quota of coefficients specified by a detailer, the Navy, and eventually by IDA.

Survey preferences with values

As an addition to the rankings, IDA may even ask each stakeholder to assign a value to the constraints on a normalized scale, say on [0,1]. In this case IDA doesn't have to transform the rankings into values, only to project them to the prefixed quota of coefficients. This enables the linear functional to yield more accurate results, because of the use of more accurate data. For example, if a sailor feels that his first preference is twice as important as his second one then he should give 1.0 for the first and 0.5 for the second. Giving equal values to multiple constraints is also possible. Note that the actual numerical value of a stakeholder's specifications doesn't matter because of the projection; only their relative values matters, and it's important to let the stakeholders understand this. Giving the highest (1.0) value to all of his preferences will not help a sailor in getting a "better job", it only means that all of his/her constraints are equally important for him/her.

Grouping the stakeholders' preferences

Once all the constraints are given in the transformed (weighted) form for each stakeholder, we need to set up the general weights for the projection for each of the three stakeholders according to detailer or Navy supervision (eventually IDA). This will tell

which class of constraints is more important in relation to the other two. For example if the command constraints and the Navy constraints are equally important, and both of them are twice as important as the sailor's preferences, then coefficients for the command, Navy, and sailor's, preferences respectively are the following: 0.4, 0.4, 0.2. The projections should be done into them. These relative weights may change from time to time, or from community to community, and always needs to be set by the Navy, or by a detailer, and eventually by IDA.

All the rankings and the above values can only be applied to the soft constraints, which can be violated. Hard and semi-hard constraints, which must be satisfied are not part of the preferences.

Some hard constraints are:

- No women on certain ships
- Sea/Shore rotation (if a sailor served on shore then his next assignment must be at sea)

Incorporating the hard constraints and the soft constraints we come up with the following grouped version of the linear functional, which considers a given job for a given sailor at a given time:

$$F(\underset{\sim}{x}) = (\prod_{i=1}^{m} c_i(\underset{\sim}{x_i})) * \sum_{j=1}^{3} a_j f_j(\underset{\sim}{x_{m+j}})$$

and

$$\sum_{j=1}^{3} a_j = 1$$

All terms have the same meaning as before with the following changes: f_j is a function which considers all the soft constraints listed by the j'th stakeholder along with their given values if available (j=1, 2, 3). We use the transformed values if only rankings were allowed.

Each a_j is a coefficient, which measures the relative importance of the i'th stakeholder.

The f_j functions have the following form:

$$f_j(\underset{\sim}{x}_{m+j}) = \sum_{k=1}^{p} b_k g_k(\underset{\sim}{x}_{m+3+k}),$$

Every g_k is a function (for k=1, 2, …, p) which measures how well the k'th soft constraint is satisfied if we were to offer the given job to the given sailor. The range of all of these functions is [0, 1]. These functions are a priori defined, and are often linear but not necessarily.

Every b_k is a coefficient which tells how important is the k'th soft constraint in relation to the others for the given stakeholder. All the coefficients are in [0, 1]. These values (or at least the rankings) are provided by the appropriate stakeholder.

Note that in this version of the constraint satisfaction the only control a stakeholder has over the module lies in the b_k coefficients. The f_j function incorporates all of the j'th stakeholder's preferences. The g_k functions are a priori set, though some additional survey may provide further information in order to better design them. In many cases they are linear. F is different for each sailor in the corresponding f_j part, but the same F is used for all jobs for the same sailor.

Keeping Constraint Satisfaction Up-to-date

Once IDA (and the linear functional) is online qualified Navy personnel (possibly a detailer) may monitor IDA's decisions and provide feedback. This would enable us or IDA to initiate changes when policies change and to tune the coefficients or the functions in order to make more accurate calculations, eventually decisions. The feedback could happen in three ways.

The first way requires code updating by the monitor. Updating only the coefficients is trivial. Changing existing functions isn't much harder, but introducing or deleting constraints may require a lot more effort. New behavior, attention, and information codelets would become necessary, which is not trivial, especially in the "conscious" version of constraint satisfaction. A whole new tuning of the coefficients also becomes necessary as a side effect of this process.

The second way only requires a high level, but formal specification, implemented as XML files, of the required changes in the constraint satisfaction module. XML can be

easily learned by any human detailer. IDA can parse XML files which create or delete behaviors, goal structures, and update old ones, effectively operating on java files.

The third, ideal, way would be to learn online through both natural language feedback and metacognition. Again, IDA needs to be tooled to create and delete behaviors, and goal structures, and update old ones for constraint satisfaction. However, to see the results of such actions, assuming that IDA runs in a real environment, may take a long time even if she changes one function at a time. Note that this strategy may work for other parts of the agent, but not as obviously as in the linear functional module, where input-output pairs are fairly deterministic, and can be easily traced. Some written history of the agent's actions and some internal modules may help in tuning.

Unfortunately, even the Navy faces difficulties in measuring the performance of detailers, so tuning of IDA (in particular, the linear functional) differ from detailer to detailer. No final ideal setup seems to be possible. The change in Navy policies, the economy, manpower availability, peace and wartime situations, etc. makes tuning more difficult and unavoidable. IDA needs to be highly adaptive, especially in regard to constraint satisfaction where key decisions are made for the sake of the stakeholders.

Recent tuning of the linear functional was done with neural networks based on large data sets provided by detailers (Kelemen 2002; Kelemen et al., 2002). Via supervised learning, neural networks not only learned to make detailer like decisions, but also provided data on how to tune the linear functional for constraint satisfaction in IDA. Neural networks have not been integrated into IDA so far, instead they act as external modules, which help to tune IDA's linear functional module.

In spite of all the difficulties, keep in mind that the ultimate goal of the IDA project and of much of artificial intelligence is to create an agent that can perform well with little or no human supervision on a wide domain, real world environment.

Matching Multiple Sailors to Multiple Jobs

Matching multiple sailors to multiple jobs could have been a long-term objective of IDA. However the classic n:m matching models (Gale and Shapley, 1962) can only be applied with restrictions for the Navy domain. Situations may change in a matter of minutes. New emails may arrive, training classes may become filled by other detailers, jobs may be taken by other sailors, and so on. Doing n:m matching, including negotiations with sailors, could be time consuming, and by the end of it some of the options may not be valid any more. However, a limited version of n:m matching applied to small groups of sailors in a narrow time frame could result in better overall satisfaction of the stakeholders. Unfortunately, this could mean that IDA only offers one job instead of the desired two or three to each sailor, which would probably not improve retention.

Any of the methods discussed can be expanded to n:m matching, and further research is currently under way. Such optimization processes would look for a global fitness for a group of sailors in a given time frame. There were several models proposed, implemented and tested for selected parts of the discussed Navy domain. To mention but a few a genetic algorithm approach was discussed by Kondadadi, Dasgupta, and Franklin (2000) and operations research techniques have been developed by Liang and Thompson

(1987), Liang and Buclatin (1988) and Ali, Kennington, and Liang (1993). Our efforts now carry over to the successor of IDA, the Multi Agent Naval Resource Distributor (MANRD) project, where every sailor has an agent capable of carrying out services similar to that IDA does on behalf of sailors, commands, or the Navy through a simulated economy (Kelemen 2002).

Discussion and Conclusion

In this paper IDA, an Intelligent Distribution Agent to automate the job assignment problem of the US Navy, was described with the main concern being its constraint satisfaction. As the second phase in a "three phase constraint satisfaction" a linear functional is largely responsible for decisions made in order to "keep the sailor happy and the Navy ready". Various versions of a standard linear functional approach were discussed along with the psychological theory of consciousness which plays an important role in IDA's constraint satisfaction as well as in IDA in general. We have seen that the best way to do constraint satisfaction is through "consciousness" in spite of its time and memory expenses. We have also seen how a thorough survey from stakeholders would contribute to making more accurate assignments to maximize everyone's happiness.

High-quality decision making using optimum constraint satisfaction is an important goal of IDA, to aid the Navy to achieve the best possible sailor, command, and Navy satisfaction performance. Neural networks with statistical criteria were applied to

either improve the performance of the current way IDA handles constraint satisfaction or to come up with alternatives. IDA's constraint satisfaction, neural networks, and traditional statistical methods are complementary with one another. It has been shown by Kozma, Harter, and Achunala (2002) that constraint satisfaction plays the primary role in action selection and therefore in intelligence. Based on our full investigation of IDA we have learned that to achieve a good general model of human behavior and decision making in an artificial intelligent system constraint satisfaction, machine learning, and statistical assessment have to be assembled automatically. We have also learned that intelligent behavior and decision making in an intelligent system can be increased through adaptation to environmental changes and the right data has to be provided to be used as training data in sufficient quantity and frequency. Having learned all these about artificial intelligent systems we can build a theory that constraint satisfaction, learning, and statistical assessment have to be integrated automatically for human intelligence. Also it can be safely said that high level human intelligence requires adaptation to environmental changes and efficient training data has to be provided in sufficient quantity and frequency.

In order to make IDA human like we applied and implemented several cognitive theories borrowed from the field of psychology. In order to make IDA's decisions efficient we performed thorough surveys of Navy detailers and officers. In particular, IDA's linear functional was tuned with the help of neural networks to make decisions similar to those made by actual detailers. We discussed three methods on how to keep constraint satisfaction, and IDA, up to date. Although IDA is already capable of different

kinds of learning, automatic tuning of its linear functional is currently under investigation.

Through our successful implementation of IDA we have learned that “conscious” software agents can be used to automate human information tasks, where even constraint satisfaction needs to be “conscious” to help the agent to deal with truly novel situations. However it is also learned that routine situations with novel content don't require “consciousness” in constraint satisfaction. It has also been learned that cognitive modeling can model large areas of cognition, which may occasionally lead to emerging theories of human cognition. Moreover it has been learned that building software based on cognitive modeling often demands vast computational power from today's computers and many tasks can be performed more efficiently without cognitive modeling (“airplanes don't flap their wings”).

In this work we briefly described a working constraint satisfaction model within an intelligent agent framework, which is able to support decision making efficiently. Such efficiency was acquired through design, implementation and testing of a wide variety of machine learning approaches, careful surveys of Navy experts and study of the vast data provided by the Navy. Design and implementation details of IDA are currently being taken over to IDA's successor, the Multi Agent Naval Resource Distributor (MANRD) in the form of sailor, command, and Navy agents, where constraint satisfaction plays a key role and n:m matching is attempted within a simulated economy.

Acknowledgement

This work is supported in part by ONR grant (no. N00014-98-1-0332). The authors thank Mohammad Amini, Jozsef Balog, Robert Kozma, Lee McCauley and the "Conscious" Software Research group for their contributions to this paper.

References

Ali, A. I., J. L. Kennington, and T. T. Liang, 1993. Assignment with En Route Training of Navy personnel. Naval Research Logistics, Vol 40, pp. 581-592.

Baars, B. J. 1988. A Cognitive Theory of Consciousness. Cambridge University Press, Cambridge.

Baars, B. J. 1997. In the Theater of Consciousness. Oxford University Press, Oxford

Bogner, M. 1999. Realizing "consciousness" in software agents. Ph.D. Dissertation, The University of Memphis, Memphis, TN, USA.

Bogner, M., U. Ramamurthy, and S. Franklin. 2000. Human Cognition and Social Agent Technology, ed. K. Dautenhahn John Benjamins, p. 113. Amsterdam.

Franklin, S. 1995. Artificial Minds. Cambridge, Mass.: MIT Press.

Franklin, S. and A. Graesser. 1997. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Intelligent Agents III, Berlin: Springer Verlag, 21-35.

Franklin, S., A. Kelemen, and L. McCauley. 1998. IDA: A cognitive agent architecture. In the proceedings of IEEE International Conference on Systems, Man, and Cybernetics '98, IEEE Press, p. 2646.

Franklin, S. 2000. Deliberation and Voluntary Action in 'Conscious' Software Agents. Neural Network World 10:505-521.

Gale, D., and L. S. Shapley. 1962. College Admissions and stability of marriage. The American Mathematical monthly, Vol 60, No 1, pp 9-15.

Hofstadter, D. and M. Mitchell. 1994. The copycat project: A model of mental fluidity and analogy-making. In Holyoak, K. and Barnden, J., editors, Advances In Connectionist And Neural Computation Theory, Vol 2: Analogical Connections, Norwood, NJ. Ablex.

James, W. 1890. The Principles of Psychology. Harvard University Press, Cambridge, MA.

Kanerva, P. 1988. Sparse Distributed Memory. Cambridge, Mass.: MIT Press.

Kelemen, A. 2002. Constraint Satisfaction as a Support for Decision Making in Software Agents. Ph.D. Dissertation, The University of Memphis, Memphis, TN, USA.

Kelemen, A., Y. Liang, R. Kozma and S. Franklin. 2002. Optimizing Intelligent Agent's Constraint Satisfaction with Neural Networks. In: "Innovations in Intelligent Systems" (A. Abraham, L. Jain, J. Kacprzyk, Eds.), in the Series "Studies in Fuzziness and Soft Computing", Springer-Verlag, Heidelberg, Germany, pp. 255-272

Kondadadi, R., D. Dasgupta, and S. Franklin, 2000. An Evolutionary Approach For Job Assignment. Proceedings of International Conf. on Intelligent Systems, Louisville, KY

Kondadadi, R., and S. Franklin. 2001. A framework of deliberative decision making in "conscious" software agents. Proceedings of International Symposium on Artificial Life and Robotics, Japan

Kozma, R., D. Harter and S. Achunala. (2002). Action Selection Under Constraints: Dynamic Optimization of Behavior in Machines and Humans. Proceedings of the IEEE International Joint Conference on Neural Networks, 2002, Hawaii, pp. 2574-2580.

Liang, T. T. and T. J. Thompson. 1987. Applications and Implementation – A large-scale personnel assignment model for the Navy. The Journal For The Decisions Sciences Institute, Volume 18, No. 2 Spring.

Liang, T. T. and B. B. Buclatin. 1988. Improving the utilization of training resources through optimal personnel assignment in the U.S. Navy. European Journal of Operational Research 33 183-190 North-Holland

Maes, P. 1990. How to Do the Right Thing. Connection Science, 1:3.

Maes, P. 1992. Learning Behavior Networks from Experience. In F. J. Varela and P Bourgine (Eds.), Toward a Practice of Autonomous Systems, Proceedings of the First European Conference on Artificial Life. MIT Press/Bradford Books.

Mitchell, M. and D. R. Hofstadter. 1991. The Emergence of Understanding in a Computer Model of Concepts and Analogy-making. Physica D 42: 322-34. Reprinted in S. Forrest, ed., Emergent Computation. Cambridge, Mass.: MIT Press.

Mitchell, M. 1993. Analogy-Making as Perception. Cambridge, Mass.: MIT Press

Ramamurthy, U., M. Bogner and S. Franklin. 1998. Conscious Learning In An Adaptive Software Agent. In Proceedings of The Second Asia Pacific Conference on Simulated Evolution and Learning, pp.24-27, Canberra, Australia.

Song, H. and S. Franklin. 2000. A Behavior Instantiation Agent Architecture. Connection Science, Vol. 12, pp. 21-44.