Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2002), *16*, 331–342. Printed in the USA.
Copyright © 2002 Cambridge University Press 0890-0604/02 $12.50
DOI: 10.1017/S0890060402165012

Agent-based support within an interactive evolutionary design system

DRAGAN CVETKOVIĆ[1] AND IAN PARMEE[2]
[1]Soliton Associates Incorporated, Toronto, ON M5C 1Y2, Canada
[2]University of the West of England, Frenchay Campus, Bristol BS16 1QY, United Kingdom

(Received August 3, 2000; Accepted August 2, 2002)

Abstract

This paper describes the use of software agents within an interactive evolutionary conceptual design system. Several different agent classes are introduced (search agents, interface agents, and information agents) and their function within the system is explained. A preference modification agent is developed and an example is given illustrating the use of agents in preference modeling.

Keywords: Conceptual Design; Multiobjective Optimization; Preferences; Software Agents

1. INTRODUCTION

Although application of evolutionary and adaptive computing technologies for design optimization is now well established, there is little recognition of their potential for design exploration through appropriate integration with conceptual design processes. Such integration supports search across predefined design spaces while also allowing exploration outside initial constraint and variable parameter bounds. Close designer interaction allows exploration involving off-line processing of initial results, which leads to a redefinition of the design space. Further designer/evolutionary search of redefined space can lead to the discovery of innovative or even creative solutions (Parmee, 1998, 1999).

Entirely machine-based conceptual design is not suggested here, nor is it currently considered viable. Best utility can be achieved from systems that enhance the designer's inherent capabilities. Appropriate integration can result in the development of prototype evolutionary design tools that provide powerful extensions to design team activity by supporting rapid, extensive exploration and stimulating innovative reasoning. This paper therefore discusses the use of agent-based methods for both search/exploration and the support of the designer in the design process. It represents a synopsis of the second part of PhD thesis research presented in Cvetković (2000), dealing with application of agents.

The idea of using preferences and agents in conceptual engineering design is not new. Some aspects of the research are presented by D'Ambrosio and Birmingham (1995), Wellman et al. (Wellman & Doyle, 1991; Wellman, 1995; Wellman & Walsh, 2000), and many other researchers.

The paper is organized in the following manner: Section 2 briefly describes the interactive evolutionary conceptual design system (IEDS), Section 3 introduces preferences, and Section 4 introduces agents. In Section 5 the use of agents within the system is discussed and some classes of agents are introduced. Section 6 provides an example of agent use. Finally, Section 7 provides conclusions, discussion, and pointers to future work. More details, in a wider context, are given in Cvetković (2000). The IEDS is described in Parmee et al. (2000, 2001) and the preferences are described in more detail in Cvetković (2000) and Cvetković and Parmee (2002).

2. THE IEDS

Conceptual design represents the initial phase of a design process (Pahl & Beitz, 1996). The research presented here is based on whole system airframe design in cooperation with British Aerospace (BAE) Systems Ltd. Some design issues have been presented elsewhere (Parmee & Purchase, 1997; Cvetković et al., 1998). A major characteristic of conceptual design relates to innovation and creativity, which

Reprint requests to: Dragan Cvetković, Soliton Associates, Inc., 44 Victoria Street 2100, Toronto, ON M5C 1Y2, Canada. E-mail: dragan@soliton.com

is very well encapsulated by the following quote (Goel, 1997):

. . . problem formulation and reformulation are integral parts of creative design. Designers' understanding of a problem typically evolves during creative design processing. This evolution of problem understanding may lead to (possible radical) changes in the problem and solution representations.

Therefore, an IEDS (Parmee et al., 2000, 2001) was developed to assist the designer during conceptual design. The core of the IEDS is described in Figure 1. It consists of the following modules:

- *information gathering processes* (e.g., cluster oriented genetic algorithms or COGAs; Parmee, 1996; Bonham & Parmee, 1998; Parmee & Bonham, 2000): a module that constantly extracts relevant information from the search processes and presents it to the design team via machine-based agents. COGAs support the rapid decomposition of complex, multivariate design space into regions of high performance and the extraction of relevant design information from such regions through good solution cover.
- *preference module:* a module for specifying the relative importance of objectives and constraints (Cvetković & Parmee, 1999*b*, 2002). Its task is to help the designer in this process by introducing several categories of importance of objectives (much less important, less important, equally important, more important, and much more important) linguistically and to transfer these values, using the concepts of leaving score and induced ordering (Fodor & Roubens, 1994), into weights used throughout the optimization process.
- *distributed coevolutionary GA:* a module for multi-objective optimization (Parmee & Watson, 1999; Parmee et al., 2000), supporting the identification of high performance regions of a multidimensional Pareto frontier. Each objective is assigned a separate optimization process with the task of optimizing (minimizing or maximizing) that objective only. At the beginning all optimization processes are independent of each other; but as the run progresses, a penalty, relating to maximal allowable Euclidean distance between the variables of each evolutionary process, is used to lead obtained solutions toward a common region. If a variable is outside a range defined by a range constraint map, the associated solution fitness is adjusted by a penalty function. The communication between processes is implemented using the Parallel Virtual Machine (PVM) software package (Geist et al., 1994).
- *problem decomposition module:* a part of the distributed coevolutionary module that uses Taguchi methods (Peace, 1993) for identifying the sensitivity of several differing objectives to individual variable parameters.
- *database module:* a module for storing interesting and promising solutions and training data.

The two components under consideration within the paper are the distributed coevolutionary GA and preference module. Preferences are used in a coevolutionary context to change the value of penalties depending on importance factors: more important objectives are penalized less, and less important objectives are penalized more. In that way, processes compete for the best solution, as the penalty function discourages the solutions that are far apart.

Examples of different algorithms and different design aspects are illustrated through the joint research project with BAE Systems. The details of the project are described in more detail elsewhere (Cvetković et al., 1998; Cvetković, 2000; Parmee et al., 2000, 2001) and in Section 5.

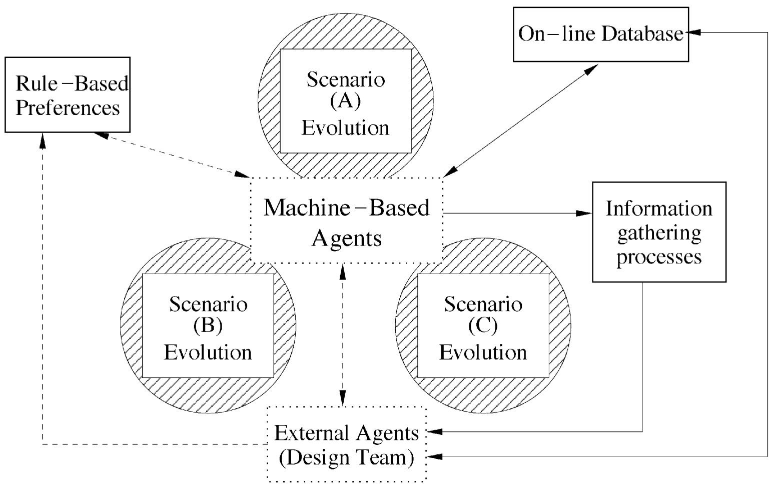
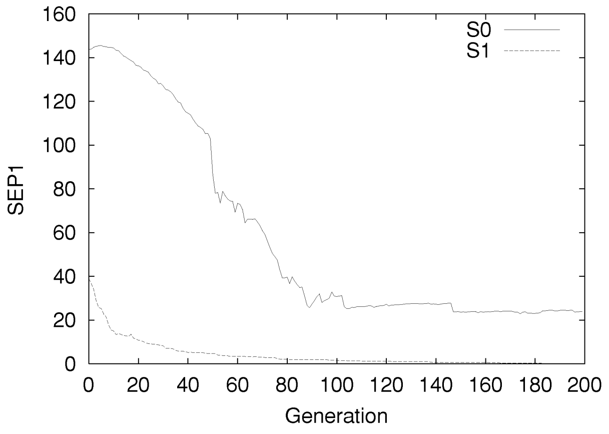
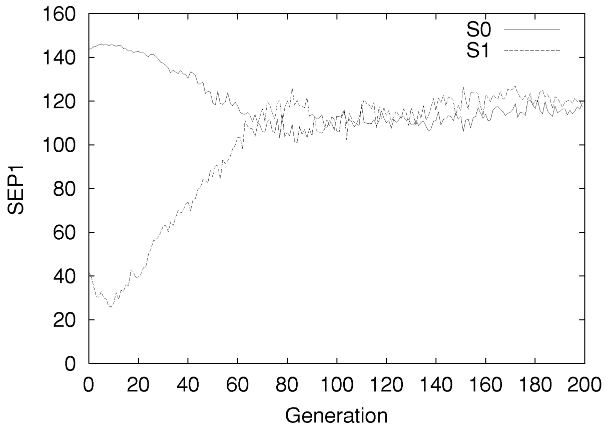


Fig. 1. The schema of IEDS.

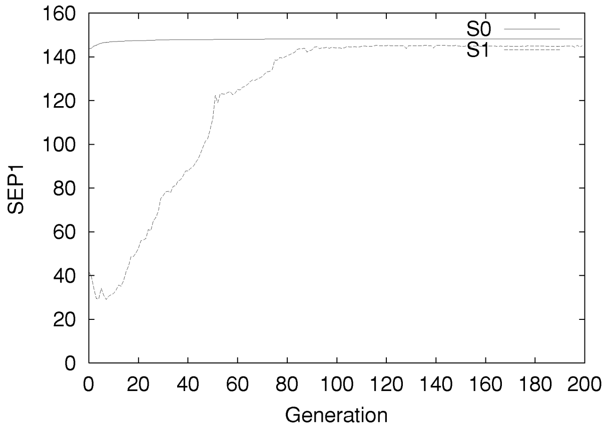
An example of the integration of these two modules is presented in Figure 2, showing the influence of preference settings on the coevolutionary optimization processes. It shows the optimization (maximization) of two objectives: both processes S_0 and S_1 work on objectives y_3 (specific excess power, SEP1) and y_9 (ferry range, FR), but the process S_0 tries to maximize objective y_3, whereas process S_1 tries to maximize objective y_9 (the objectives conflict). The plots show the values of objective y_3. Figure 2(a) shows the optimization results using preference $y_3 \ll y_9$ (i.e., objective y_3 is much less important than y_9), Figure 2(b) shows equal preferences ($y_3 \approx y_9$) and Figure 2(c) shows the optimization results using preference $y_3 \gg y_9$ (i.e., objective y_3 is much more important than y_9). These preferences direct the search toward different regions of y_3 versus y_9 values: if $y_3 \ll y_9$, the search processes will converge toward smaller values for y_3 (and larger values for y_9), as illustrated in Figure 2(a); if they are considered equally important, they will converge toward compromise regions where both objectives are "average" [Fig. 2(b)]. Similar results are obtained by plotting objective y_9 instead of y_3.



(a) $y_3 \ll y_9$



(b) $y_3 \approx y_9$



(c) $y_3 \gg y_9$

Fig. 2. Coevolution with different preferences: the influence on optimizing y_3 (SEP1, process S_0) and y_9 (FR, process S_1). The results are for SEP1 with an average of 20 runs.

Figure 2 demonstrates how preferences control the search process, driving the compromise region toward the one with better values for the more important objective. It can also be noted in Figure 2(a) that the results of the two optimization processes do not converge to the same extent as they do with equal preferences [Fig. 2(b)]. This behavior (i.e., the result difference) could be explained by noting that the more important objective is penalized less: the solution that is usually penalized if the Euclidean distance between variables is more than, for instance, 10% will now be penalized if the distance is more than, for instance, 20%.

The penalty-factor corrected value of objective y_i, $f_i(\boldsymbol{x}, t)$ was calculated using the following formula:

$$f_i(\boldsymbol{x}, t) = f_i^0(\boldsymbol{x}) \cdot \prod_{j=1}^{k} \chi_d(x_j, x_i, t), \quad \text{where}$$

$$\chi_d(x, y, t) = \begin{cases} \varphi_p \cdot \theta_p(x, y), & |x - y| \geq d_p(t); \\ 1, & \text{otherwise.} \end{cases} \tag{1}$$

for

$$\theta_p(x, y) = \min\{1, w_x / w_y\}, \tag{2}$$

where w_s is the weight of objective s, $f_i^0(\boldsymbol{x})$ is the original value of objective y_i for a given set of inputs $\boldsymbol{x}$, t is the generation number, $d_p(t)$ is the monotonically decreasing function specifying minimal nonpenalized distance, and φ_p is the original penalty factor (usually 0.5).

During the design process, the designer is able to change the preferences and objectives to optimize and to dynamically add, modify, and delete constraints (scenarios) with an almost immediate feedback from the system, showing the influence of the changes on the solutions generated.

During the later phases of design few objectives tend to be in evidence, whereas during conceptual design the designer requires a global picture and therefore routinely deals with many possible objectives. The vast number of parameters involved can confuse the designer. In order to reduce cognitive overload and to help the designer in mundane and less creative tasks, a set of agents has been developed that are an integral part of the IEDS. The components relevant to the research described in this article are marked by dotted lines in Figure 1. They are described in Section 5 after an introduction to preferences and agents.

3. PREFERENCES

The notion of preferences is not new, and different authors propose different preference systems in engineering design (Wellman & Doyle, 1991; D'Ambrosio & Birmingham, 1995; Greenwood et al., 1996). The present authors have developed a preference systems for specifying the relative importance of objectives. The following predicates have been introduced (Cvetković & Parmee, 1999*b*; Cvetković, 2000; Parmee et al., 2000; Cvetković & Parmee, 2002):

Relation	Intended Meaning
$\approx$	Is equally important
$\prec$	Is less important
$\ll$	Is much less important
$\neg$	Is not important
!	Is important

the set of axioms specifying the properties of these relations follows:

- $\approx$ is an equivalence relation;
- $\prec$ and $\ll$ are strict orders;
- $\approx$ is *congruent* with $\ll$ and $\prec$;
- $\ll$ is a subrelation of $\prec$; and
- miscellaneous properties:

$$!x \vee \neg x, \tag{3}$$

$$!y \wedge \neg x \Rightarrow x \ll y, \tag{4}$$

$$\neg x \wedge \neg y \Rightarrow x \approx y, \tag{5}$$

$$x \prec y \wedge y \ll z \Rightarrow x \ll z. \tag{6}$$

The corresponding, "more important" ($\succ$) and "much more important" ($\gg$) relations are defined as

$$x \gg y \overset{\text{def}}{\Leftrightarrow} y \ll x, \tag{7}$$

$$x \succ y \overset{\text{def}}{\Leftrightarrow} y \prec x. \tag{8}$$

The above cited work describes the use of preferences within multiobjective optimization (weighted sums optimization, weighted Pareto optimization, etc). The current article tries to merge preferences with agents, that is, to use agents to automate preference estimation.

4. AGENTS

Defining the notion of an agent is a very difficult task, as the following quote demonstrates. According to Watt (1996, p. 89):

> . . . "Agent" is a difficult word for a difficult concept; covering a rag-bag of concepts that span a whole gamut of different kinds of behaviour, including, for example, autonomy, learning and social interaction; but there is a common ground. An agent will set out to do something, and do it; therefore it has competences for intending to act, for action in an environment, and for monitoring and achieving its goals. Of course, the adequate performance of these, other competencies, such as learning, negotiation, and planning, may be helpful or even necessary.

An overview of theoretical aspects of agents (including topics such as belief, intention, default reasoning, possible world semantics, etc.) is given in Wooldridge and Jennings (1995).

In the most general sense, agents can be classified into the following categories (Stenmark, 1999): interface, system, advisory, filtering, retrieval, navigation, monitoring, recommender, and profiling. However, for the purposes of our application, conceptual design, the following classes of agents appear to offer utility:

interface agents: agents that help the designer deal with a system and that (if the designer wishes it) hide some low-level noninteresting details from the designer;

search agents: agents that cover the process of optimization, cooperation, population monitoring, jumping out of regions, constraint questioning, and so on; and

information agents: agents that deal with information obtained, look for interesting solutions and filter uninteresting ones, make decisions with regard to what and where to explore, resolve conflicts, and so forth.

Figure 3 classifies the agents used throughout the project. A similar classification is used by Sycara et al. (1996).

Because the role of agents necessitates collaboration (negotiations) and interaction, these two concepts are important and will be described in the following two subsections.

4.1. Negotiations

According to Sycara (1991), there are four conflict situations where negotiation is used in design. These conflicts follow (Berker, 1995):

- different agents make conflict recommendations for a parameter value;
- a value proposed by one agent makes it impossible for another agent to offer consistent values for other attributes;
- a decision of one agent adversely affects the optimality of other agents; and
- alternate approaches achieve similar functional results.

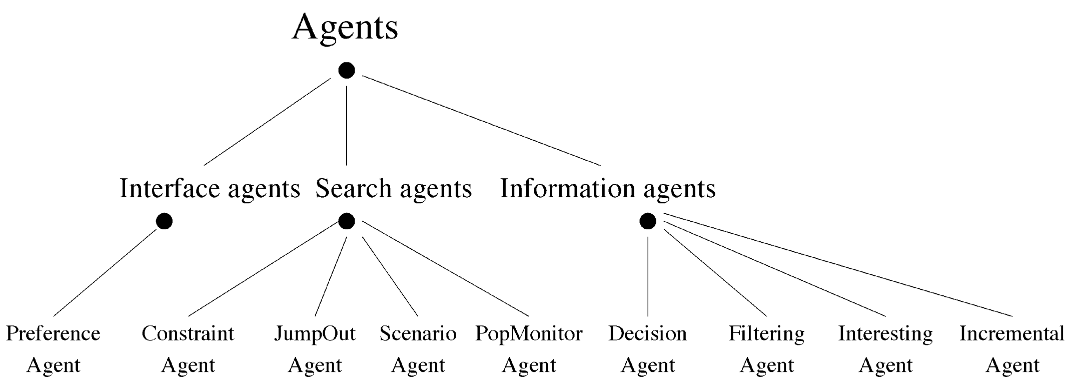


Fig. 3. Agent classification and examples of agents within classes.

The negotiation process proceeds as follows:

1. generation of proposal;
2. generation of counterproposal based on feedback from dissenting agents; and
3. communication of justifications and supporting evidence.

The paper by Nwana et al. (1996) gives an overview of different coordination techniques.

4.2. Agent communication

Agents need a common language in order to be able to communicate. The first developed methods used a blackboard architecture (Hayes–Roth, 1985; Brenner et al., 1998), as presented in Figure 4(a), where all agents are able to read from and write to a shared memory area. The other method utilizes directed message passing from agent to agent, as shown in Figure 4(b) using message transport methods (e.g., PVM, MPI, etc.). More modern agent communication languages are described in Labrou et al. (1999).

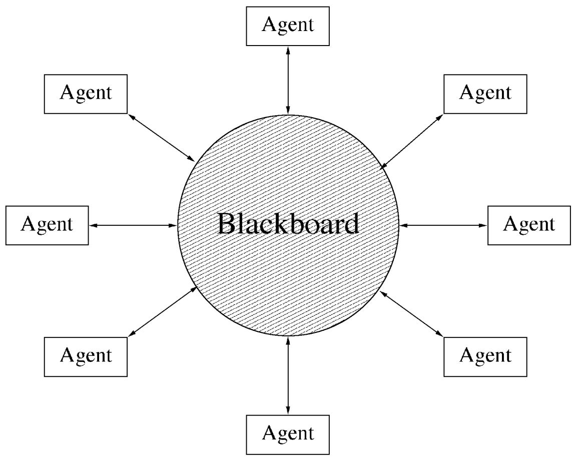
5. USE OF AGENTS IN DESIGN PROCESSES

A very successful agent-based application is described in Ygge and Akkermans (1999): it describes a climate control system for large buildings with many offices using a "market-based agent approach." Agents buy and sell cooling power resources (Huberman & Clearwater, 1995). A very comprehensive review of computer supported cooperative environments for engineering design is given in Shen and Norrie (1999). Wellman describes "market-oriented programming" (1995, 1996). Some issues are also discussed in Lander (1997). An agent-based system for conceptual design is described in two studies (Campbell et al., 1999; Campbell, 2000).

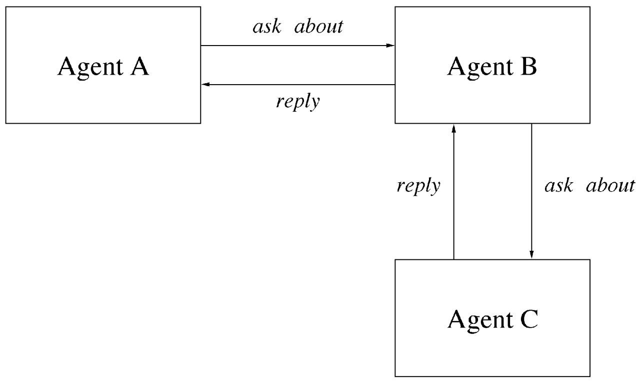
In our development of agents, we decided to follow the philosophy of simple agents: an agent performs only one function, similar to single function agents (SIFAs; Berker, 1995; Brown et al., 1995). SIFAs are designed to perform one function only, and they have the following parameters:

function: what kind of work it performs;

target: on what parameter or object the agent has an immediate effect; and



(a) Blackboard system



(b) Direct communication

Fig. 4. Agent communication methods.

point of view: the perspective that the agent takes in performing its function on its target. The point of view can be cost, strength, and so forth.

Brown et al. (1995) argue that in this way it is much easier to construct new agents and (equally important) it is much easier to debug agents.

Following the classification of agents given in Section 4, the agents developed for the BAE Systems conceptual engineering design system are schematically presented in Figure 3. They mostly follow the above SIFA philosophy: a single function per agent.

The conceptual airframe design project (or miniCAPS project, explained below) has been developed in cooperation with BAE Systems. The details of the project are not relevant for this paper, and it is not possible to describe the model in detail. However, a brief summary follows.

The miniCAPS model (Webb, 1997) is a version of Computer Aided Project Studies (CAPS), which is BAE Systems software used by designers during the earliest investigation stages of a new aircraft. MiniCAPS reproduces the general characteristics of CAPS but without the computational complexity. MiniCAPS models a variety of disciplines and consists of three modules: *aerodynamics* (lift and drag coefficients, flight envelope, etc.); *performance* (ferry range, sustained turn rate, take-off distance, cruise height, etc.) and *configuration* (wing position, wing shape, canard position, number of engines, mass estimation, etc.). A high degree of interaction is incorporated between these disciplines and many of the objectives are thus highly conflicting.

One of the goals of the project was a development of an (semi-)intelligent and (semi-)autonomous system that will utilize preferences and agents in order to reduce designer's burden through the performance of more mundane tasks, enabling designer to concentrate on more creative aspects of the design.

At present miniCAPS utilizes nine variable parameters producing a total of 13 outputs, each of which may be considered an objective.

5.1. Interface agents

Interface agents are used to reduce the complexity of increasingly sophisticated and overloaded conceptual design systems. They build a user friendly interface between the designer and the computer. The designer can specify the quality threshold of solutions, situation trigger actions, and other parameters.

Their role is to help the designer in a (boring) question and answer preference estimation procedure (e.g., "Statement: *A* is the most important to me and *B* the least important of all the objectives") that the agent transforms into a series of questions and answers suitable for the preference module.

5.1.1. *Application within the system*

We implemented an agent in the system that helps the designer in the preference estimation procedure. It allows the designer to specify the complete preference order on the command line or in a file, for example,

$$y_9 \approx y_{10} \gg y_1 \approx y_2 \approx y_6 \succ y_3 \approx y_4 \succ y_5 \gg y_7 \approx y_{11} \succ y_8,$$

instead of answering the following standard sequence of questions:

$$y_9 \approx y_{10};$$

$$y_1 \approx y_2 \approx y_6;$$

$$y_3 \approx y_4;$$

$$y_7 \approx y_{11};$$

$$y_1 \succ y_2, \quad y_1 \succ y_5, \quad y_1 \gg y_7, \quad y_1 \gg y_8, \quad y_1 \ll y_9;$$

$$y_3 \succ y_5, \quad y_3 \gg y_7, \quad y_3 \gg y_8;$$

$$y_5 \gg y_7;$$

$$y_5 \gg y_8;$$

$$y_7 \succ y_8.$$

In the initial stage, the designer will probably prefer the second method (pairwise comparisons); but as the process goes on, specifying the complete order is easier, especially if the changes are incremental (as in the case of the incremental agent described in Section 5.4.1). Because the preference method transforms preferences into a total order, these two methods are equivalent, providing that the initial complete order specified is not circular (also checked by the agent).

5.2. Search agents

The role of search agents is to look for "interesting" solutions. Here the notion of interesting is defined by the designer, for instance, a good solution with a large Euclidean distance from the majority of the population. It looks for "novelty" solutions, which might be overlooked and ignored otherwise.

Among the search agents, the following classes of agents have been investigated and developed:

jumpout agent: an agent that searches exclusively outside of variable boundaries;

quality monitoring agent: an agent that monitors the quality of solutions;

constraint agent: an agent that tries to find out what solutions can be obtained by breaking one of the constraints;

scenario agent: an agent that solves the original problem minus one of the scenarios (dynamically, on-line specified constraints in a relatively rich mathematical language with logical operators; Cvetković & Parmee, 1999a); and

population monitoring agent: an agent that monitors the convergence of the population.

These agents are described in subsequent sections. The following need to be considered when applying agents:

- where to search;
- variation of each variable parameter range; and
- constraint versus objective space (i.e., shall we use penalty functions to transform constraints into objectives, etc.).

5.2.1. *Jumpout agent*

This agent searches exclusively out of boundaries. It can initiate a new GA that works in parallel with the main one, or it can initiate a quick hill climber that starts from one of the solutions, changes a randomly chosen variable to be out of defined range, and then performs hill climbing. Alternatively, it could start modifying good solutions, not just any random population member. Parameters of the agents limit how far outside the domain the agent can go and for how many generations. How many individuals to create could also be specified. The method used could be hill climbing, simulated annealing (Laarhoven & Aarts, 1987), scatter search (Laguna, 1999), differential evolution (Storn & Price, 1995), and many others. If desired, this can be combined with tabu lists (Glover & Laguna, 1997) to remember already explored regions.

This area has also been investigated within the Plymouth Engineering Design Centre through the work of Beck and Parmee (1999).

Jumpout agents are useful for questioning the initial constraints and domain of the problem and for attempting to expand the problem domain. During conceptual design, some limits are set rather ad hoc. This agent assesses whether it is possible to obtain better solutions by ignoring some of the initial limitations.

5.2.2. *Quality monitoring agent*

If the solution fitness is at least 90% of the best solution, this agent notifies the designer about it. The quality threshold (the percentage of the best solution, 90% in this example) is a configurable parameter. Also, the user can specify when to be notified:

- immediately, so that the search can be led in that direction, or
- afterward, for off-line analysis.

5.2.3. *Constraint agent*

This agent tries to break some of the constraints; and if a good solution is obtained, it notifies the designer. Constraints can have different levels of "softness" assigned (from 0 to 1, or from "absolutely unchangeable" to "you can do whatever you want with it"). For each solution the information regarding the number of constraints broken (if using penalty functions for resolving constraints) is stored.

Another constraint agent monitors the best solutions and tries to further optimize the solution for each of them, ignoring (i.e., breaking) one of the constraints. If the obtained solution is significantly better, it will present it to the designer and let him or her decide if that constraint is necessary. The designer can mark some of the constraints as nonquestionable (as before).

5.2.4. *Scenario agent*

The scenario agent is similar to a constraint agent, except that it deals with scenarios, which are a set of constraints connected with the logical operators AND, OR, and NOT (Cvetković & Parmee, 1999a; Cvetković, 2000). Each agent solves the original problem minus one of the scenarios. For m scenarios, that means $m + 1$ parallel GAs (one with all scenarios). This could be very costly because parallel search processes are required. However, the increased use of parallel and more and more powerful computers makes such distributed strategies increasingly feasible. One example of scenarios is given in Section 6.

5.2.5. *Population monitoring agent*

If the GA search is too concentrated (i.e., too converged in variable space) in one part of the search space, this agent "jumps" far away from the converged point in space (but still within the feasible domain $\mathcal{D}$) and starts a new search there. Bookkeeping about already explored regions is needed in order to avoid visiting the same region many times.

Three levels of spontaneous behavior are available:

- machine-based agent automatically decides to try jumping out of regions, breaking constraints, and so forth;
- the designer only decides on the action taken; and
- interactively, the agent suggests and the designer declines or accepts.

5.3. **Information agents**

This class of agents is more intelligent then the previously described classes and should be able to make autonomous decision concerning (but not limited to):

- "spawning" an agent to search in a given direction;
- "killing" an agent that is not very successful;
- negotiation between agents (unless they need to consult the designer);
- recognition of the novelty of a solution (eventually consulting the database of existing solutions) and turning the designer's attention toward it; and
- when to consult the designer.

One example of an information agent is the incremental agent described in Section 5.4.1.

5.4. Closing design loop

In a conceptual design context the agents can be applied in the manner presented in Figure 5.

Common to all optimization processes, there is a standard search path: preferences $\longrightarrow$ search engine $\longrightarrow$ output. The new component here is a process that picks up solutions from the search engine and presents them to a consortium of agents that look at it in terms of different interests or points of view (say agent *A* monitors objective 1, agent *B* monitors objective 2, . . ., agent *D* monitors variable 1, which should be ideally between 0 and 5 or 15 and 20, etc.). This information is presented to the designer together with some suggestions (e.g., "can we change this preference?" or "this solution path is no good for some constraints") and preferences and some mathematical model details are changed (with the designer's approval). This connects agents nicely with the scenario concept. The agents are employed to monitor constraints and scenarios and to analyze those that are never completely fulfilled. The unfulfilled constraints and scenarios usually give an indication about the changes needed to improve the design. Ideally, these changes should be suggested by the agents.

The next section describes an agent that tries to fulfil scenarios through changing preferences and variable ranges. Every time it finds an unfulfilled scenario, it suggests the changes to the designer and, if approved, continues the search in the modified setting.

5.4.1. Incremental agent

The incremental agent developed in this section will close the design loop, as presented in Figure 6. This is a more detailed view of the general IEDS described in Section 2 and presented in Figure 1. For the sake of simplicity, in Figure 6 it is assumed that agent and scenario values are incorporated into the fitness value as some form of penalty.

The incremental agent works in the following way:

1. use the original designer's preferences (both for objectives and for scenarios) and run the optimization process;
2. if some of the scenarios are not fulfilled, suggest increasing the importance of those scenarios that are not fulfilled and repeat the search process;
3. if some scenarios are still not fulfilled, although they are classified as the most important, suggest changing variable ranges (of those variables mentioned in scenarios) and repeat the search with this new setting; and
4. if some scenarios are still not fulfilled, give up and report the results to the designer.

6. EXAMPLE OF AGENT USE

The following example illustrates the use of agents and the automated preference adjustment through the use of incre-

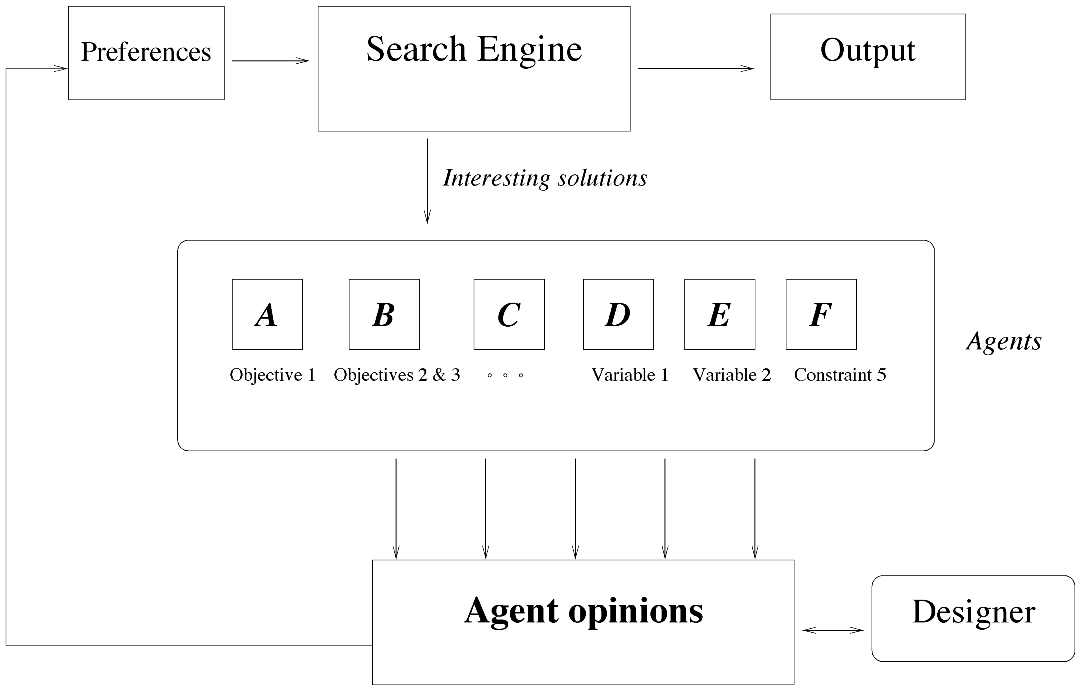


Fig. 5. Agents in an optimization context.

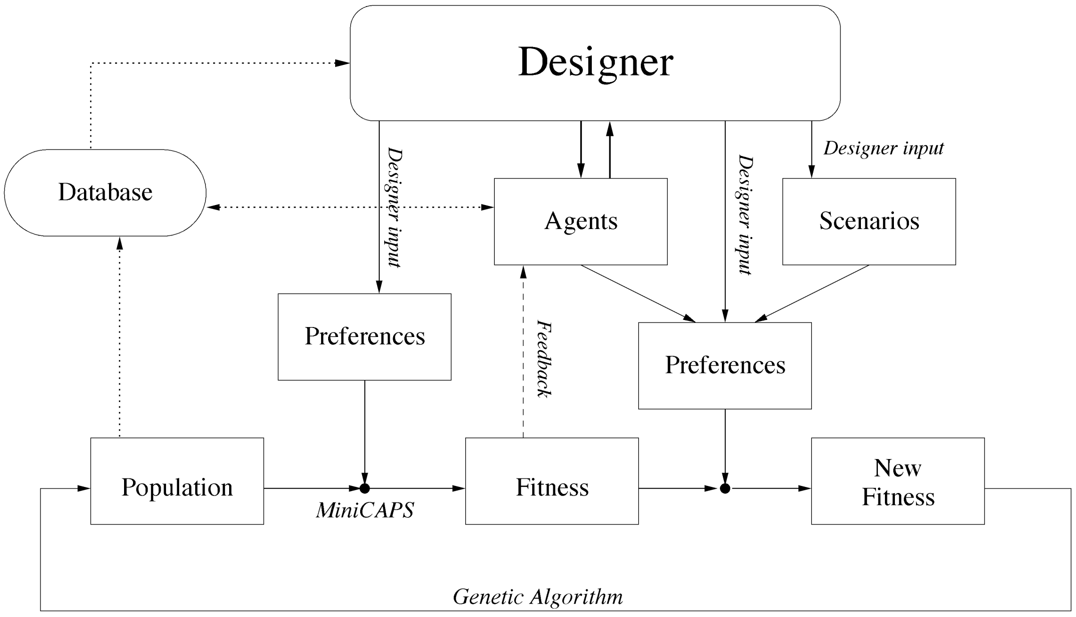


Fig. 6. Closing the design loop using agents. See text for details.

mental agents described in Section 5.4.1. The example also illustrates the interactiveness of conceptual design process using the IEDS.

EXAMPLE 1. Suppose that for BAE System's airframe design problem, the following set of scenarios is given:

```
    # constraints for F-111 Aardvark Bomber
S1: y11 >= 9.74 & y11 <= 19.20   # Wing span
S2: x4 <= 61.07 & x4 >= 48.77    # Wing plan area
S3: y9 >= 4707                   # Ferry range
S4: y10 >= 45360                 # Take off mass
S5: x3 >= 0.75                   # Max cruise
                                   speed 919 km/h
S6: y1 <= 951                    # Take-off run
```

The process goes as follows:

1. The original set of objective preferences, that is,

$$y_1 \approx y_9 \approx y_{10} \approx y_{11},$$

and the original set of scenario preferences, that is,

$$S_1 \approx S_2 \approx S_3 \approx S_4 \approx S_5 \approx S_6,$$

give the solution where $y_{10} = 30{,}936$, so scenario S_4 (which requires that $y_{10} \geq 45{,}360$) is not fulfilled and is noted by the agent.

2. At that stage the agent suggests increasing the importance of scenario S_4,

$$S_4 \succ S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6,$$

but the obtained solution still does not satisfy scenario S_4.

3. At that stage the agent suggests another increase of the importance of the S_4 scenario:

$$S_4 \gg S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6.$$

This again gives a solution where scenario S_4 is not fulfilled, and this is noted by the agent.

4. Because the importance of scenario S_4 cannot be further increased, the agent suggests modifying the variable bounds. It suggests increasing the range of all variables by 10% and starting again.

5. With all equal scenario preferences there is no difference, so an increase in S_4 importance is suggested again.

6. For a preference setting,

$$S_4 \succ S_1 \approx S_2 \approx S_3 \approx S_5 \approx S_6,$$

the solution finally satisfies S_4 but violates scenarios S_1, S_2, and S_6.

7. The agent suggests

$$S_4 \succ S_1 \approx S_2 \approx S_6 \succ S_3 \approx S_5,$$

which again gives a solution where S_1, S_2, and S_6 are satisfied but S_4 is not, so the next suggestion is

$$S_4 \gg S_1 \approx S_2 \approx S_6 \succ S_3 \approx S_5.$$

This gives a solution where scenarios S_1, S_2, and S_3 are not fulfilled: $x_3 = 0.92$, $x_4 = 86$, $y_1 = 759$, $y_9 = 290$, $y_{10} = 46{,}982$, and $y_{11} = 23.6$. At this point the agent might call the designer for further assistance.

8. From the analysis so far, the designer can immediately see that he or she can (using the given airframe model) either have S_4 fulfilled or S_1 and S_2 (wing span and wing plan area are connected to take-off mass). At this point the designer decides to use the following preferences (the last set of preferences above),

$$S_4 \gg S_1 \approx S_2 \approx S_6 > S_3 \approx S_5,$$

and to just increase the range of variable x_4 from [20, 80] to [20, 120], keeping all other variables to their original ranges. This gives a solution $x_3 = 0.87$, $x_4 = 120$, $y_1 = 951$, $y_9 = 7846$, $y_{10} = 49{,}924$, and $y_{11} = 26.8$, which violates scenarios S_1 and S_2 but is probably the best compromise in these circumstances.
9. If the designer further decides to minimize take-off mass instead of maximizing, the result is $x_3 = 0.86$, $x_4 = 120$, $y_1 = 878$, $y_9 = 9829$, $y_{10} = 45{,}360$, and $y_{11} = 26.8$.
10. If the designer, out of curiosity, further increases the range of x_4 to [20, 140], the solution is $x_3 = 0.87$, $x_4 = 140$, $y_1 = 951$, $y_9 = 8517$, $y_{10} = 49{,}452$, and $y_{11} = 14.5$, where only scenario S_2 is not fulfilled and the obtained airplane has a very large wing plan area but small wing span (probably some delta-shaped wings).
11. And so forth.

As can be seen from this example, IEDS provides a powerful system for interactive analysis and change of parameters in the run and gives almost immediate feedback about the objectives, constraints, and preferences.

Note that in this particular example the agent did not find an acceptable solution, but by trying different methods it enabled the designer to find the right one (as much as the concept of "rightness" can be unambiguous in a multiobjective framework).

6.1. Design agent cooperation

Consider a system with several agents, each with the task to optimize a single objective. The question is how to make them collaborate. Each agent is aware of the quality of its own solution. If the quality of one agent's solution is inferior to the quality of the solution of some other agents and their solutions conflict, that agent compromises and accepts a worse solution from its point of view, for the benefit of other agents. In a case where they cannot decide (e.g., both agents think that they have quality solutions), the designer is asked to decide. Once the designer resolves a conflict, the agents need to remember the decision and try to learn from it so that the next time a similar situation happens, they can resolve the conflict among themselves without the designer's intervention.

Some form of voting system, where the importance of each agent also plays a certain role in resolving conflicts, can be useful. However, that can lead to problems, because according to Arrow's impossibility theorem (Arrow, 1951), it is not possible to construct a group preference relation satisfying the following basic principles: complete domain, positive association of social and individual ordering, independence of irrelevant alternatives, individual's sovereignty, and nondictatorship.

If an agent is successful, it is made more important than the others (so that good solutions usually count as more important in the negotiation process). For each "best solution," we increase the value of the solution. If an agent is less successful, we reduce the agent's importance or the quality of its solutions. Limits to both the maximal and minimal possible importance of an agent are needed in order to keep diversity of solutions generated. More details are given in Cvetković (2000). A similar learning method is used by Campbell (2000).

7. CONCLUSION AND DISCUSSION

A frequently asked question is "do we need agents?" When writing about agents, Jennings and Wooldridge (1995) say the following:

> Although agent based technology clearly has an important role to play in the development of leading edge compound applications, it should not be regarded as a panacea. The majority of applications which currently use agents could be solved using non-agent techniques (in most cases not as well, but in some cases better!). . . . As with all system designs, the ultimate choice depends upon a large number of technical and non-technical factors . . .
>
> Whilst this new system paradigm offers many exciting opportunities, it has a down side which invariably places a limit on the types of application to which agents can be applied. The first major problem is that the overall system is unpredictable and non-deterministic: which agents will interact with which other in which way to achieve what cannot be predicted in advance. Even worse, there is no guarantee that dependencies between the agents can be managed effectively, since the agents are autonomous and free to make their own decisions . . . The second main disadvantage is that the behaviour and properties of the overall system cannot be fixed at design time. While a specification of the behaviour of an individual agent can be given, a corresponding specification of the system in its entirety cannot, since global behaviour necessarily emerges at run time.

This paper describes some techniques and methods used in the interactive design system for conceptual design of products. The use of agents is delineated and some examples of their use are presented.

As this research illustrates, the agents do not need complex architectures: simple agents such as the "jumpout agent" are very useful in the optimization process. Their use enables the designer to concentrate on the higher level aspects of the design process and frees him or her from having to worry about the behavior of the optimization module. Filtering and information agents are useful for turning the designer's attention to some interesting and/or unusual (but promising) solutions. They are also useful for questioning all (variable) limits and constraints.

We do not claim that these agents are very intelligent or sophisticated. However, they should be considered in terms of the environment in which they are used, that is, in an interactive design system. In this context, however, their usefulness has been illustrated to some extent. In the previous example the agent did not really solve the problem, but during its work it collected enough information that the designer was able to immediately see the way to an acceptable solution. The designer is able to solve the problem without using agents but not without extensive and tedious "trial and error" runs. The agents described in this article are designed for a conceptual design environment where design goals and constraints are still rather vague, and in that environment any help to the designer is important.

Because of the complex problem domain, it is very hard to judge the role of the agents in the system, that is, if they are limiting factor or a factor of improvement in design. The primary role in using agents is pragmatic: they should make the design task easier for the designer.

For the same reason it is very hard to compare the developed system with agent systems described in the literature.

Future work will include the development of new classes of agents and their tighter integration in design processes.

REFERENCES

Arrow, K.J. (1951). *Social Choice and Individual Values*. New York: Wiley.

Beck, M.A., & Parmee, I.C. (1999). Design exploration: Extending the bounds of the search space. *Proc. 1999 Congress on Evolutionary Computation—CEC99*, pp. 519–526. Washington, DC: IEEE.

Berker, I. (1995). *Conflicts and negotiations in single function agent based design systems*. Master's Thesis. Worcester, MA: Worcester Polytechnic Institute. Available on-line at http://www.cs.wpi.edu/Research/aidg/SiFA/ilan.html.

Bonham, C.R., & Parmee, I.C. (1998). Cluster oriented genetic algorithms (COGAs) for the decomposition of multi-dimensional engineering design spaces. In *Advances in Computational Structures Technology* (Topping, B.H.V., Ed.), pp. 87–95. Stirling, Scotland: Civil-Comp Press.

Brenner, W., Zarnekow, R., & Wittig, H. (1998). *Intelligent Software Agents: Foundation and Applications*. Berlin: Springer–Verlag.

Brown, D.C., Dunskus, B.V., Grecu, D.L., & Berker, I. (1995). SINE: Support for single function agents. *Applications of AI in Engineering AIENG'95*, Udine, Italy.

Campbell, M.I. (2000). *A-design: An electro-mechanical design strategy utilizing adaptive complex systems and multi-objective optimization*. PhD Thesis. Pittsburgh, PA: Carnegie Mellon, Mechanical Engineering Department.

Campbell, M.I., Cagan, J., & Kotovsky, K. (1999). A-design: An agent-based approach to conceptual design in a dynamic environment. *Research in Engineering Design 11(3)*, 172–192.

Cvetković, D. (2000). *Evolutionary multi-objective decision support systems for conceptual design*. PhD Thesis. Plymouth, UK: University of Plymouth, School of Computing.

Cvetković, D., & Parmee, I.C. (1999*a*). Genetic algorithm-based multi-objective optimisation and conceptual engineering design, *Proc. 1999 Congress on Evolutionary Computation—CEC99*, pp. 29–36. Washington, DC: IEEE.

Cvetković, D., & Parmee, I.C. (1999*b*). Use of preferences for GA-based multi-objective optimisation. In *GECCO-99: Proc. Genetic and Evolutionary Computation Conf.*, pp. 1504–1509. San Francisco, CA: Morgan Kaufmann.

Cvetković, D., & Parmee, I.C. (2002). Preferences and their application in evolutionary multiobjective optimisation. *IEEE Transactions on Evolutionary Computation 6(1)*, 42–57.

Cvetković, D., Parmee, I.C., & Webb, E. (1998). Multi-objective optimisation and preliminary airframe design. In *Adaptive Computing in Design and Manufacture: The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation* (Parmee, I.C., Ed.). pp. 255–267. New York: Springer–Verlag.

D'Ambrosio, J.G., & Birmingham, W.P. (1995). Preference–directed design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 9*, 219–230.

Fodor, J., & Roubens, M. (1994). *Fuzzy Preference Modelling and Multicriteria Decision Support*. Dordrecht: Kluwer.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., & Sunderam, V. (1994). *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. Cambridge, MA: MIT Press.

Glover, F., & Laguna, M. (1997). *Tabu Search*. Boston: Kluwer.

Goel, A.K. (1997). Design, analogy and creativity. *IEEE Expert 12(3)*, 62–70.

Greenwood, G.W., Hu, X.S., & D'Ambrosio, J.G. (1996). Fitness functions for multiple objective optimization problems: Combining preferences with Pareto ranking. In *Foundations of Genetic Algorithms 4 (FOGA'96)* (Belew, R.K., & Vose, M.D., Eds.), pp. 437–455. San Francisco, CA: Morgan Kaufmann.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence 26*, 251–321.

Huberman, B.A., & Clearwater, S. (1995). A multi-agent system for controlling building environments. In *Proc. First Int. Conf. Multi-Agent Systems, ICMAS'95*, pp. 171–176. Boston: AAAI Press/MIT Press,

Jennings, N.R., & Wooldridge, M.J. (1995). Applying agent technology. *Journal of Applied Artificial Intelligence 9(4)*, 357–369.

Laarhoven, P.J.M.v., & Aarts, E.H.L. (1987). *Simulated Annealing: Theory and Applications*. Dordrecht: Kluwer.

Labrou, Y., Finin, T., & Peng, Y. (1999). Agent communication languages: The current landscape. *IEEE Intelligent Systems 14(2)*, 45–52.

Laguna, M. (2002). Scatter search. In *Handbook of Applied Optimization* (Pardalos, P.M., & Resende, M.G.C., Eds.), pp. 183–193. New York: Oxford Academic Press.

Lander, S.E. (1997). Issues in multiagent design systems. *IEEE Expert 12(2)*, 18–26.

Nwana, H.S., Lee, L., & Jennings, N.R. (1996). Co-ordination in software agent systems. *BT Technical Journal 14(4)*, 79–89.

Pahl, G., & Beitz, W. (1996). *Engineering Design: A Systematic Approach*, 2 ed., London: Springer–Verlag.

Parmee, I.C. (1996). The maintenance of search diversity for effective design space decomposition using cluster oriented genetic algorithms (COGAs) and multi–agent strategies (GAANT). In *Proc. Adaptive Computing in Engineering Design and Control*, pp. 128–138. Plymouth, UK: University of Plymouth, PEDC.

Parmee, I.C. (1998). Exploring the design potential of evolutionary/adaptive search and other computational intelligence technologies. In Adaptive computing in design and manufacture: The integration of evolutionary and adaptive computing technologies with product/system design and realisation. *Proc. 3rd Conf. Adaptive Computing in Design and Manufacture (ACDM'98)*, pp. 27–42. New York: Springer–Verlag.

Parmee, I.C. (1999). Exploring the design potential of evolutionary search, exploration and optimisation. In *Evolutionary Design by Computers* (Bentley, P.J., Ed.), pp. 119–143. San Francisco, CA: Morgan Kaufmann.

Parmee, I.C., & Bonham, C.R. (2000). Towards the support of innovative conceptual design through interactive designer/evolutionary computing strategies. *Artificial Intelligence in Engineering, Design, Analysis and Manufacturing 14(1)*, 3–16.

Parmee, I.C., Cvetković, D., Bonham, C.R., & Packham, I.S. (2001). Introducing prototype interactive evolutionary systems for ill-defined multi-objective design environments. *Advances in Engineering Software 32(6)*, 429–441.

Parmee, I.C., Cvetković, D., Watson, A.H., & Bonham, C.R. (2000). Multiobjective satisfaction within an interactive evolutionary design environment. *Evolutionary Computation 8(2)*, 197–222.

Parmee, I.C., & Purchase, G. (1997). Integrating computational intelligence technologies with design and manufacturing team practice. *Proc. Intelligent Design in Engineering Applications Symposium (IDEA'97)*, pp. 95–99, Aachen, Germany.

Parmee, I.C., & Watson, A.H. (1999). Preliminary airframe design using co-evolutionary multiobjective genetic algorithms. In *GECCO–99: Proc. Genetic and Evolutionary Computation Conf.*, pp. 1657–1665. San Francisco, CA: Morgan Kaufmann.

Peace, G.S. (1993). *Taguchi Methods: A Hands-On Approach*. Reading, MA: Addison–Wesley.

Shen, W., & Norrie, D.H. (1999). Agent-based systems for intelligent manufacturing: A state-of-the-art survey. *Knowledge and Information Systems. An International Journal 1(2)*, 129–156. Available on-line at http://sern.cpsc.ucalgary.ca/CAG/publications/abm.htm.

Stenmark, D. (1999). Evaluation of intelligent software agents. Available on-line at http://w3.informatik.gu.se/~dixi/agent/agent.htm.

Storn, R., & Price, K. (1995). *Differential Evolution–A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces*. Technical Report TR-95-012, Berkeley, CA: University of Berkeley, ICSI.

Sycara, K. (1991). Cooperative negotiation in concurrent engineering design. In *Computer-Aided Cooperative Product Development* (Sriram, D., Logcher, R., & Fukuda, S., Eds.). New York: Springer–Verlag.

Sycara, K., Decker, K., Pannu, A., Williamson, M., & Zeng, D. (1996). Distributed intelligent agent. *IEEE Expert 11(6)*, 36–46.

Watt, S.N.K. (1996). Artificial societies and psychological agents. *BT Technical Journal 14(4)*, 89–97.

Webb, E. (1997). MINICAPS—A Simplified Version of CAPS for Use as a Research Tool. Unclassified Report BAe-WOA-RP-GEN-11313. Warton, UK: British Aerospace.

Wellman, M.P. (1995). The economic approach to artificial intelligence. *ACM Computing Surveys 27(3)*, 360–362.

Wellman, M.P. (1996). Market-oriented programming: Some early lessons. In *Market-Based Control: A Paradigm for Distributed Resource Allocation* (Clearwater, S., Ed.). London: World Scientific.

Wellman, M.P., & Doyle, J. (1991). Preferential semantics for goals. *Proc. 9th National Conf. Artificial Intelligence*, Vol. 2, pp. 698–703. Boston: AAAI Press/MIT Press.

Wellman, M.P., & Walsh, W.E. (2000). Distributed quiescence detection in multiagent negotiation. *Fourth Int. Conf. Multiagent Systems (ICMAS'2000)*, pp. 317–324. Boston.

Wooldridge, M.J., & Jennings, N.R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review 10(2)*, 115–152.

Ygge, F., & Akkermans, H. (1999). Decentralized markets versus central control: A comparative study. *Journal of Artificial Intelligence Research 11*, 301–333.

Dragan Cvetković received his BSc and MSc in Mathematics from the University of Belgrade, Yugoslavia. Recently he received his PhD in Computer Science from the University of Plymouth, UK. In the late 1980s he started teaching in the Faculty of Mathematics, University of Belgrade, Yugoslavia. Genetic algorithms became his interest in early 1990s when he worked at the Max Planck Institute for Computer Science, Saarbrücken, Germany, and later for GMD—National Research Center for Information Technology, St. Augustin, Germany. In the period from 1997 to 2000 Dragan completed his PhD research at the Plymouth Engineering Design Centre, University of Plymouth, UK. His research interests include genetic algorithms, multiobjective optimization, the logic of preferences, and agent-based methodologies. Dr. Cvetković is currently working for Soliton Inc., Toronto, Canada.

Ian Parmee has several years of experience in both the contracting and consultancy sectors of the civil engineering industry. He returned to an academic career in 1991 and played a major role in the development of Plymouth University's EPSRC Engineering Design Centre, investigating the integration of evolutionary computing technologies with engineering design. He has now joined the University of the West of England, Bristol, where he is currently establishing research in the area of evolutionary design and decision making. Dr. Parmee's research has resulted in over 100 publications in journals, conference proceedings, and books.