Lindsay D. Hanna
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA. 15213

Jonathan Cagan
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA. 15213
cagan@cmu.edu

Evolutionary Multi-Agent Systems: An Adaptive and Dynamic Approach to Optimization

This paper explores the ability of a virtual team of specialized strategic software agents to cooperate and evolve to adaptively search an optimization design space. Our goal is to demonstrate and understand how such dynamically evolving teams may search more effectively than any single agent or a priori set strategy. We present a core framework and methodology which has potential applications in layout, scheduling, manufacturing, and other engineering design areas. The communal agent team organizational structure employed allows cooperation of agents through the products of their work and creates an ever changing set of individual solutions for the agents to work on. In addition, the organizational structure allows the framework to be adaptive to changes in the design space that may occur during the optimization process. An evolutionary approach is used, but evolution occurs at the strategic, rather than solution level, where the strategies of agents in the team are the decisions for when and how to choose and alter a solution, and the agents evolve over time. As an application of this approach in a static domain, individual solutions are tours in the familiar combinatorial optimization problem of the traveling salesman. With a constantly changing set of these tours, the team, with each agent employing a different solution strategy, must evolve to apply the solution strategies which are most useful given the solution set at any point in the process. We discuss the extensions to our preliminary work that will make our framework useful to the design and optimization community.

1 Introduction

True engineering design is a cooperative process in which individuals with different areas of expertise coordinate their efforts to develop the best solution to a given problem [1]. Such problems, especially in the early stages of design, are often open-ended, multi-modal, and even dynamic [2]. In industry, individuals on a design team may be considered specialists whose approaches for moving designs towards optimality are the product of both their own and their predecessors' experiences. The effective application of each individual's approach depends not only on what the particular approach is, but also on how and when it is employed based on the team's progress. For example, computational fluid dynamics or finite element analysis is not as valuable early in the design process as back of the envelope calculations and heuristic estimations. Together, the approach to design improvement and the timing of its application can be considered a *strategy*. We seek to understand how strategies evolve in a cooperative virtual team so that the team as a whole yields better designs than any single individual's strategy alone or *a priori* set team strategy.

This paper introduces EMAS – an **E**volutionary **M**ulti-**A**gent **S**ystem for adaptive optimization which employs the evolution of design strategies within a cooperative virtual team. As is the case in current engineering design processes, solution strategies in EMAS *evolve* as conditions change and as new solution states are discovered during the optimization process, so that the best strategies are employed at the correct time. Evolution is not a new concept in optimization and problem solving [3], but the use of evolutionary processes is typically applied at the solution level, rather than at the strategic level. In our framework, the strategies for generating solutions are recombined, altered, and removed by applying genetic operators that in typical genetic and evolutionary algorithms are performed on individual solutions. We also add a cooperative dimension to the evolutionary process, which distinguishes our work as well from genetic and evolutionary programming techniques. Cooperation between design team individuals is modeled in our framework by embodying each strategy in an autonomous agent and allowing the population of agents to communicate. Communication between agents involves the posting of working and completed solutions to the design problem in communal or shared databases. The team of agents navigates the design space cooperatively as individuals apply genetically encoded approaches for altering solutions contained in these databases according to genetically encoded rules for when to do so. At the same time, selection, mutation, and reproduction of agents in the team allow strategies to evolve and adapt to the specific problem and design space.

The evolution of cooperating solution strategies is presented in this work to combat the problem of not knowing which strategy to use in an unknown, but static design space. In our example,

we use the familiar combinatorial optimization problem of the traveling salesman. This problem has many extensions to design in the areas of manufacturing process planning, routing, and scheduling. It is a difficult problem to solve but is also so well studied that many strategies have been developed and several sample problems with known solutions are available. However, the EMAS framework is also designed to be robust and flexible in dynamic design spaces – those in which constraints, objectives, even dimension and size of the problem are changing. Changes in the design space are reflected in the shared databases and are thus passed on to all agents indirectly without disrupting the design process. Through both the evolution of the agents, again embodying the strategies for generating solutions, and their cooperation as a team, the best set of strategies for generating solutions to a particular problem at each point in the optimization process is created.

2 Background

2.1 Agent-Based Methods

We have chosen to represent strategies in this work as autonomous software agents. Though the definition of "agent" is a topic of some debate in the theoretical artificial intelligence community (see Franklin and Graesser [4]), we define agents here as entities which perceive their environment, in this case the shared memories and solutions, and autonomously act upon that environment through effectors. This definition is very similar to those of Russell and Norvig [5] and De Souza [6].

EMAS follows in the footsteps of several other agent-based approaches which have been very effective in optimization and engineering design. Most notably, the structure of EMAS, with shared memories or databases as the primary mode of cooperation between algorithmic or strategic agents, is closest to the concept of Asynchronous Teams (A-Teams) [6,7,8]. This concept was also extended in the research of Campbell, Cagan, and Kotovsky [1] for engineering design environments where user preferences in multi-objective conceptual design problems were changing. In the work of Olson and Cagan [2], the effect of cooperation and coordination of design decisions in these systems in the area of manufacturing process planning was studied as well. Each of these approaches have proven that agents with different abilities, cooperating on a common problem can produce more and better solutions and designs than any individual working independently. Our extension to these earlier approaches is in the incorporation of the evolutionary process for determining the best set or team of abilities for solving that common problem. Instead of specifying the agents' abilities and strategies *a priori*, we allow these characteristics to emerge naturally by subjecting the team to the processes of evolution. Again, our goal is to study how these strategies emerge in a dynamic virtual design team of cooperating but specialized individuals.

2.2 Algorithm Portfolios

The population of agents in EMAS represents a set of strategies for creating solutions. This population evolves as a result of selective pressure on the strategies, or agents, to perform better given the current problem definition and the set of solutions in the memory. Similar motivation is employed in algorithm selection [9], algorithm portfolios [10,11], meta-learning [12] and other similar paradigms [13] which seek to understand and exploit the (often unknown) properties of algorithms that make them successful. Unlike many of these approaches, in EMAS, each agent cooperates by working off of the solutions created by other agents. In this way our work is similar to that of Moral, Sahoo, and Dulikravich [14], in which a single population of candidate solutions is transferred automatically between different evolutionary algorithms, with the criteria for switching between the algorithms specified *a priori*. EMAS extends work in this area by making the agents autonomous, capable of deciding *on their own* when to affect which solution or solutions. Thus rather than automatic selection of the best algorithm or set of algorithms at an appropriate time by an external "manager", the choice is distributed among the agents (or algorithms). This means that instead of having to tune global parameters for switching between algorithms or agents, such as a function for online time allotment to each algorithm [15] or amount of improvement necessary to continue running an algorithm, these decisions can be made and updated locally. Not only that, the decision making criteria can change over time to adapt to new environmental conditions as the agent population evolves.

2.3 Evolutionary Computing

The fact that strategies, embodied in agents, evolve over time makes EMAS an extension of evolutionary computation [16,17,18] as well. The individuals in EMAS are agents, which again correspond to solution strategies - methods of choosing and transforming solutions in the shared memories. EMAS has several advantages associated with the evolutionary computation areas of genetic and evolutionary programming – namely that a fixed string or complex representation of the solution is not necessary [19]. The extension of EMAS in the area of evolutionary and genetic programming comes from the incorporation of cooperation between individuals in the population. Though cooperation has been considered between multiple related populations in cooperative co-evolution [20], few have considered or exploited cooperation within the same population. One exception is the work of Cristea, Arsene, and Nitulescu [21], which explores the concept of an evolving population of sensorimotor agents navigating in a changing 2D world. EMAS abstracts concepts from this research as well for the more complex interaction of agents needed for engineering design with EMAS. A summary of the placement of EMAS in terms of the related literature can be seen in Table 1.

Table 1 Placement of EMAS in relation to relevant evolutionary computing concepts and literature

	Individual	Multiple Populations?	Cooperation
Genetic/Evolutionary Algorithms	Solution	No	None
Genetic/Evolutionary Programming	Solution strategy	No	None
Cooperative Co-evolution	Partial solution	Yes	Between populations
Evolutionary Multi-Agent Systems	Solution strategy (Agent)	No	Between individuals

3 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) was chosen as an application for the proposed framework because it is such a well known and straightforwardly defined problem with extensions to the engineering areas of planning and scheduling. However, the goal of this work is not to present an algorithm which solves the TSP better than any other algorithm thus far. Instead, we wish to illustrate through a well studied example problem that our framework is capable of increasing the effectiveness of individual solution strategies by evolving and coordinating them in a decentralized manner. The objective of the TSP is, given a set of cities and a cost function associated with traveling between each pair of cities, to find the round trip tour with the lowest cost that visits each city once and only once. For the problems we will explore in this paper, the cost, or distance, function between cities is a 'pseudo-euclidean' function described by Padberg and Rinaldi [22]. However, extension of the TSP to engineering design applications such as scheduling and manufacturing planning can be accomplished by utilizing different cost or distance functions.

Simple, heuristic algorithms for solving TSP were intentionally used in place of more powerful ones like the Concorde TSP solver [23,24] so that the inner workings of the EMAS team and its ability to create solutions superior to those created by individuals could be studied more easily. There is another reason for this choice as well: as design problems become more and more complex, the algorithms and tools for solving them become similarly complex and the conditions under which those tools are successful are often unknown. Though the simultaneous search of both strategy and solution space may seem unnecessary, it avoids *a posteriori* fine tuning of parameters and assumptions of the design space characteristics that can plague other more centralized optimization approaches. By showing that the bi-level approach is effective at producing good solutions even when the algorithms and methods used are not the 'best', we are also able to provide designers with an alternative to trial and error experimentation with multiple algorithms as more complex problems are encountered.

The algorithms chosen fall into three categories: *construction*, *improvement*, and *reduction*. Construction algorithms are so named because they take, as input, an incomplete or partial tour and return either a complete tour or a longer partial tour after adding cities in a predefined, heuristic manner. For this study three simple heuristic construction algorithms, nearest insertion, farthest insertion, and arbitrary insertion, were used (for descriptions of these algorithms, the reader is referred to Golden and Stewart [25]). Improvement algorithms, as their name would suggest, improve an existing partial or complete tour by rearranging the order of the cities in the tour based on different rules. There were three heuristic-based improvement algorithms used in this study, 2-Opt [26], 3-Opt [27], and simple mutation (city swapping). Reduction algorithms break down complete tours into partial tours. In this work, two very basic heuristic reduction algorithms are employed. The first of these is random reduction, which involves randomly removing a random number of cities in the tour. Best partial reduction, the second reduction algorithm, returns the best partial tour (the tour with the shortest average leg length) containing half of the total number of cities in the same consecutive order as the original tour.

4 Methodology

4.1 Evolutionary Agents

To perform genetic operations such as crossover and mutation, we have chosen to represent EMAS agents, again embodying strategies, as binary strings. Other representations abound in the evolutionary computation literature, see Rothlauf [28] and Deb [29], but binary is generally regarded as the simplest representation for individuals. The binary string, or chromosome, defining an individual agent in the evolving team of the proposed framework represents the decisions the agent will make in its lifetime, in other words, its behavior. We argue that decisions should be the primary element of an agent's genetic makeup because autonomy, the ability of an agent to make decisions on its own without being told what to do, is essential to the definition of an agent [7,30]. For the particular application of the TSP, agent decisions were defined as follows:

1. From what memory will a tour be chosen,
2. Which tour from that memory will be worked on,
3. How will the chosen tour be worked on (i.e. which algorithm will be run), and
4. Where (which memory) will the new tour be put once work is completed on it.

Thus, the chromosome of each agent consists of four binary genes identifying these properties (see Fig. 1). It is important to reiterate that unlike typical genetic algorithms, these gene strings represent agents' strategies for generating solutions, not the solutions themselves. The choice method defines the tour characteristics which will make an agent more or less likely to choose that tour, i.e. if the agent will choose the best tour, the worst tour, be biased towards better tours, or be biased towards worse tours. These choice methods may be made more complex as well by increasing the gene length or representation to include reference to specific characteristics of a solution which make it more or less attractive to an agent. Note also that incorporating more complex or powerful algorithms into EMAS is simply a matter of changing the encoding paradigm for the algorithm gene, for instance by increasing the number of bits to describe

the algorithm. In addition, the gene could be lengthened to describe additional parameters of algorithms which we would prefer not to specify *a priori*, such as how many iterations to run, convergence criteria, etc. This could be particularly useful in design problems where multiple algorithms or heuristics may be available but where the parameter specification greatly affects algorithm performance on particular problem classes. The significance of the memories will be discussed in the next section.

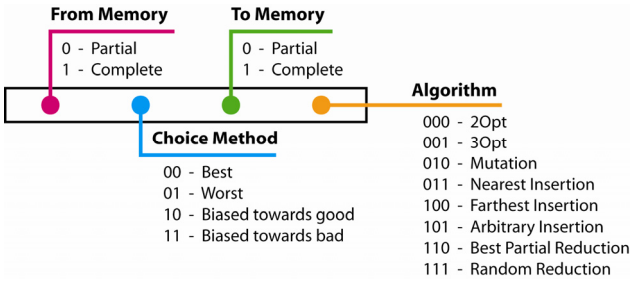

Fig. 1 Structure of proposed EMAS agent chromosome

4.2 Agent Organization: Creating an Evolutionary Multi-Agent Team

The agent system architecture developed is similar to the asynchronous team architecture developed by [8] in that it incorporates the idea of shared memories, which allow agents to cooperate indirectly by providing a place for agents to present their work so that it is visible and available to others. However, in those systems the characteristics of each agent and the rules for their relationships to the memories are specified *a priori* [6,7]. In the proposed system the agent-memory cycles are evolved by including input space and output space decisions in the agents' chromosomal representations (toMemory and fromMemory). For the specific application of the Traveling Salesman Problem, only two memories were used: one for partial tours, or tours that do not contain all of the cities, and one for complete tours. The tours in these memories can also be said to evolve over time – but through the genetically determined actions of the agents, rather than through recombination and mutation within the population of solutions as would occur in a typical genetic algorithm [31,32]. To prevent the overflow of memories, a simple quality-based destruction protocol has been used in preliminary studies that involves simply removing the worst solution in memory when a better solution is added. This simple destruction protocol can easily be enhanced as well to incorporate additional parameters for destruction of solutions, such as those used in A-Teams [6,7].

Memories in EMAS may be defined by the *problem independent* characteristics of the solutions that are acceptable within each memory. For example a "partial" solution in a layout problem would be one in which only some of the components have been placed, or one in which all components have been placed but constraints are violated. The use of shared memories with such extendible definitions is one of the keys to utilizing EMAS in dynamic environments. Consider a change in the problem formulation that involves the addition of a new variable – for instance, the addition of a new city in TSP which must be assigned a location in the tour. This event would cause many approaches, such as genetic algorithms using fixed length chromosomes and cooperative co-evolution with a single population for each variable, to require a restart with the new parameters. In EMAS, however, such a change results merely in a transfer of solutions from one memory to another – leaving the team of agents intact. Thus the knowledge and strength of the currently evolved team is not lost when a change occurs. Changing the number of constraints in the problem results in a similar transfer between memories.

5 Algorithm Description

The EMAS algorithm simulates temporal asynchrony by dividing the overall process into discrete iterations. Each iteration, all agents undergo *activation*, at which point they make decisions and perform actions based on their genetic sequence. The result of activation is the generation of a new solution by the agent. Solutions are thus a *byproduct* of the evolution of the agents, and do not, as in a GA, undergo reproduction, mutation, and selection themselves. The solutions do, however, provide a basis for increasing or decreasing an agents' fitness. In our preliminary work, evaluation of the agents is fairly simple – an increase in fitness is directly proportional to an increase in the average solution quality in memory brought about by the agent when it adds a new solution. We are currently investigating more complex agent evaluation protocols.

After all agents have been activated, *reproduction* occurs, in which parents are selected based on their fitness and new agents are created. Reproduction and activation both involve a simple operator for *mutation*. Finally, the agent community undergoes *selection*, where the weakest individuals (those with a low fitness value) are removed from the population. In this section, each of these important functions is discussed in detail.

5.1 Activation

Each iteration, all agents are activated. Activation results in the agents creating new solutions based on their genetic encoding for how to choose and manipulate a solution. Because an agent's fitness is directly influenced by the quality of the solution it creates at this time, activation is critical in driving the evolutionary process of the agents (strategies). The cost associated with activation is an increase in the agent's age, or number of activations. As will be discussed in the Selection section, age influences the likelihood of an agent being destroyed or selected for reproduction. Activation of an agent consists of verification that it is able to work and simulation and testing to determine if it will make a positive difference. The verification step is performed because some memory-algorithm combinations are incompatible, which means there may be genetic 'misfits'. For instance, a construction algorithm cannot be run on a complete tour, so if an agent's strategy specifies that the agent both choose from the complete memory and utilize a construction algorithm, the agent cannot apply its strategy and is mutated. Simulation, the process outlined in Fig. 2 by the dashed line, is performed so that agents will not place a solution that will decrease the average solution quality into their destination memory (toMemory). This keeps the quality of solutions in the memory high because worse solutions are never added, though perhaps at the cost of diversity of solutions. If an agent is unable to improve the average solution quality after n tries, the agent undergoes mutation, as described in the Mutation section, and its fitness is updated (decreased) to reflect its inability to contribute. For our implementation, $n = 3$. If an agent *is* successful in

coming up with a solution that increases the average solution quality, it then inserts the new tour into its destination memory and its fitness is updated based on its contribution. A flowchart of agent activation is shown in Fig. 2.

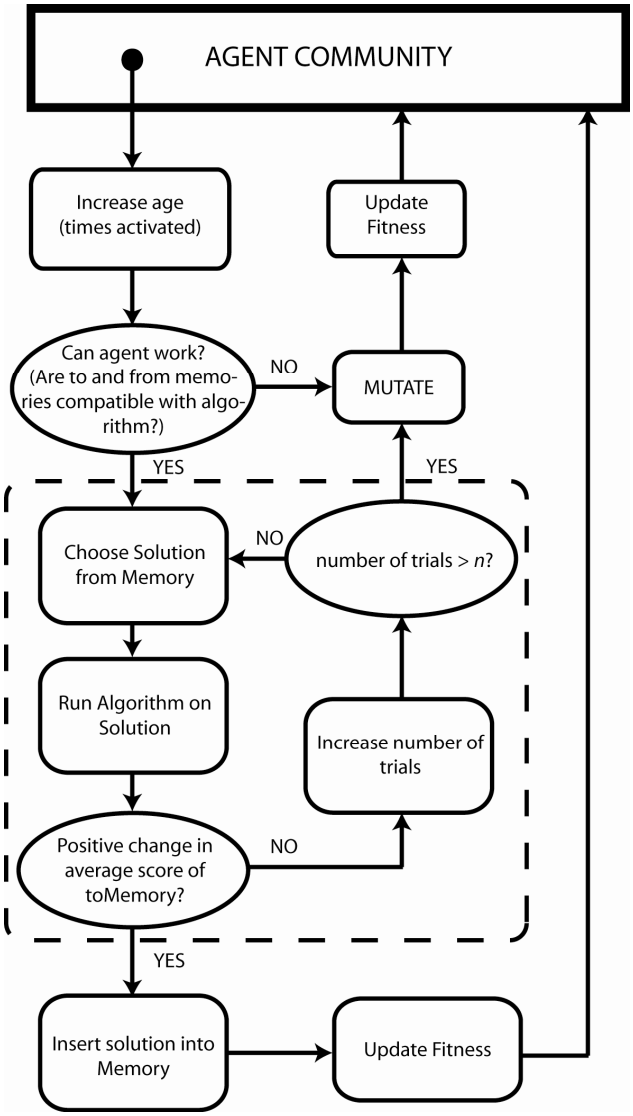

Fig. 2 Flowchart of agent activation

5.2 Mutation

In the proposed framework, mutation is used for two purposes. The first purpose, common to most evolutionary and genetic algorithms, is to make the system more stochastic – mutation allows a more thorough exploration of the design space for individuals by introducing randomness into their creation. Here, the "design space for individuals" is the strategy space – the collection of all possible strategies available to agents for creating new solutions – not to be confused with the design space of the problem. As mentioned in the previous section, individuals may also mutate during activation if they are not being successful, which could be considered a simple form of Lamarckian learning in evolution. This secondary mutation is a way of allowing individual agents to adapt to the problem environment by trying new decision methods, achieving diversity by variation. Both types of mutation are random, meaning that a single randomly chosen bit is altered in the binary gene.

5.3 Reproduction

After activating each agent in an iteration, agents with a score above zero are randomly paired up as parents and allowed to reproduce. Each agent may only reproduce once in an iteration, and during reproduction is subjected to crossover with a randomly assigned partner at a single random crossover point. Recombination using single point crossover was the only method of reproduction considered in this preliminary study, though incorporation of others (see [33] for descriptions of other evolutionary operators) is possible as well. This means that following reproduction, two children (agents embodying solution strategies) are created which contain reciprocal parts of their parents' genetic string. The resulting two children each have a 50% chance of being mutated. The initial fitness of the children is determined by activating them immediately after their creation.

5.4 Selection

When new agents are added through reproduction, the worst agents are selected from the population to be eliminated, keeping the population size constant. Selection begins by sorting the agents by fitness with lowest on the bottom and highest on the top. Fitness, as mentioned earlier, is based on the ability of the agent to make positive changes in its destination memory. The amount of increase in an agent's score after a single iteration is thus directly proportional to the percentage decrease in distance from optimal of the solution they have chosen. Agents with the same fitness are further sorted by age, the oldest on the bottom and the youngest on the top. Once sorting is complete, agents are removed from the bottom of the list (the oldest and worst performing) until the population is back to its original size.

6 Results

6.1 Comparison to Hybrid Algorithms

The goal of this work, as stated in the introduction, is to understand how individuals' strategies may evolve within a cooperative virtual team so that the team as a whole yields better solutions than any single individual's strategy alone or *a priori* set team strategies. We thus compared the resulting tours generated by EMAS to those generated by both individual algorithms on their own as well as *a priori* determined hybrid algorithms. Each hybrid algorithm, which may be considered a more advanced strategy or a combination of strategies since no agent in EMAS runs more than a single algorithm, consisted of a construction algorithm followed by an improvement algorithm. Intuitively, and based on the No Free Lunch Theorem [34], we would expect EMAS to perform at least as well as the base algorithms, which it did. In addition, though, the solution quality reached by EMAS was also consistently better than that reached by the hybrid algorithms. Recall that less complex, heuristic algorithms were intentionally used to allow us to study the behavior of EMAS more closely and that producing the optimal solution to TSP was not the ultimate objective.

For the 48-city problem, EMAS was run 48 times and compared to the solutions resulting from running construction algorithms from each starting city and then running improvement algorithms on the resultant tours. Each trial of EMAS consisted of 100 iterations with an agent team or population size of 10. Selection, reproduction, and mutation were performed at the end of each iteration and pre-processing indicated that solution quality generally converged before 100 iterations. The same starting tour or city will always lead to the same final optimization solution after running any of the base construction or improvement algorithms presented (with the exception of arbitrary insertion). By running all hybrid algorithms on every starting city we are attempting to provide a good approximation of the effectiveness of these hybrid strategies. Table 2 indicates that, on average, the solution quality produced by EMAS is much better than that of the hybrid strategies. In the table, the mean is taken over all trials and the value given for each trial of EMAS is the average solution quality in the complete memory at the end of that trial. Similar results were found for the 532-city problem. The same procedure (running the hybrid algorithms on every starting city) was used for the 532-city problem, though EMAS was only run 50 times for 50 iterations each time because of its long computation time. Even so, the average solution quality of EMAS was still much better than any of the hybrid strategies for this more complex problem.

Table 1 Mean distance (%) from optimum solution and standard deviation of hybrid algorithms compared to EMAS algorithm for 48- and 532-city problems

Algorithm	ATT48	ATT532
3-Opt + NI	3.2 ± 2.7	13.8 ± 3.6
3-Opt + FI	3.5 ± 1.5	10.5 ± 1.6
3-Opt + AI	3.1 ± 1.3	6.1 ± 1.3
2-Opt + NI	9.6 ± 2.3	23.4 ± 2.0
2-Opt + FI	6.7 ± 1.1	14.1 ± .6
2-Opt + AI	6.1 ± 2.1	8.2 ± 1.2
EMAS	***.7 ± .6***	***3.1 ± 1.6***

The consequence of this increased quality of solutions was computation time. Because EMAS involves running several of the base algorithms each iteration (and because they are currently run synchronously rather than asynchronously), it is expected that the amount of time required to reach the solutions generated is much higher. A single run of the hybrid nearest insertion followed by 3-Opt on any individual starting tour for ATT48 would take less than a second, whereas a single trial of EMAS run on ATT48 for 100 iterations takes an average of around 8 seconds. However, no matter how many times a hybrid algorithm is run on the same starting city, it will always produce the same final tour (with the exception of arbitrary insertion), which as we have just shown for the hybrid algorithms is almost always worse than the result of the EMAS algorithm. Also, EMAS will continue to improve the quality of solutions in the memory if the number of iterations is increased or made continuous (albeit at an ever decreasing rate), whereas once the hybrid algorithms exhaust their heuristic rules, they can no longer improve the solution even if improvement is possible.

The evolutionary component of EMAS is the most likely culprit of the model's long runtime. However, just as is the case in the actual design process, we have shown that this element is crucial to the success of a cooperative team. Without the benefit of the previous generations' successes and failures, the team at each iteration is essentially a random collection of strategies. So even when the same team size is used for the same number of iterations with the same cooperation scheme as EMAS, the lack of intelligence behind each iteration's team strategy selection in the randomly assigned teams scenario will limit the scenario's performance. In Table 3, the results of such a random assignment of individual strategies are compared to EMAS. As in the last row of Table 2, the first column in Table 3 is the mean taken over all trials where the value given for each trial corresponds to the average solution quality of the complete memory at the end of that trial. Computation time for EMAS is still inferior to the randomly assigned team, though to a much lesser extent than the comparison between EMAS and the hybrid algorithms. The difference in computation time is undoubtedly a result of the selection, reproduction, and mutation processes which are performed at each iteration of EMAS. However, the average solution quality of EMAS is more than 30% better than that of the randomly assigned teams, which can also be attributed to the use of evolutionary processes.

Table 3 Comparison of EMAS to randomly generated team for ATT532 (averages of 50 trials)

	Mean distance (%) from optimal solution of average tour in complete memory	St. Dev. (%)	Avg. Time (hrs)
Randomly Assigned Teams	12.6	0.8	4.3
EMAS	8.6	1.1	14.0

6.2 Utilizing Emergent Patterns

We have just shown that randomly assigned teams do not yield the same caliber results as those obtained by EMAS. Clearly, the evolutionary nature of strategy selection in EMAS is essential to the formation of higher-quality solutions in the cooperative multi-agent team. In analyzing the behavior of the agent population over the course of several runs of EMAS, however,

we discovered patterns in the number of different types of agents flourishing in the agent community at various iterations. These patterns emerged solely through the evolutionary processes of the agent population and were not the result of any external manipulation of the system. Even averaging the team behavior of 1000 trials of EMAS on ATT48 and 25 trials of EMAS on ATT532 was not enough to conceal a defined *peak of constructors in early iterations followed by a smaller peak of improvers in later iterations*. For example, in the 48-city problem (behavior shown in Fig. 3 (a) and (b)), at about the fifth iteration, the team is dominated by the construction algorithms nearest, farthest, and arbitrary insertion, with an average of 5.5 out of 10 agents running one of these algorithms. In the 532-city problem (behavior shown in Fig. 3 (c) and (d)) a similar peak in construction agents occurs around the 20th iteration, with more than 7 out of 10 agents on the team running either nearest, farthest, or arbitrary insertion. Following this initial constructor-dominated team, the number of improvers peaks as well, around the 10th iteration for the 48-city problem and around the 50th iteration in the 532-city problem. At this point, around 5 out of 10 agents on the team in the 48-city problem and 6 out of 10 agents on the team in the 532-city problem are running an improvement algorithm, mostly 3-Opt. Eventually, after about 40 iterations for ATT48 and almost 100 iterations for ATT532, averaging many trials leads to an approximately equal number of each type of agent in the team. There is never a defined peak in the number of reducers in these trials, though their influence does seem to increase following the constructor peak. This may suggest that their activity following the initial creation of solutions by the constructors is crucial to generating the fodder in the partial memories for further construction and improvement later.

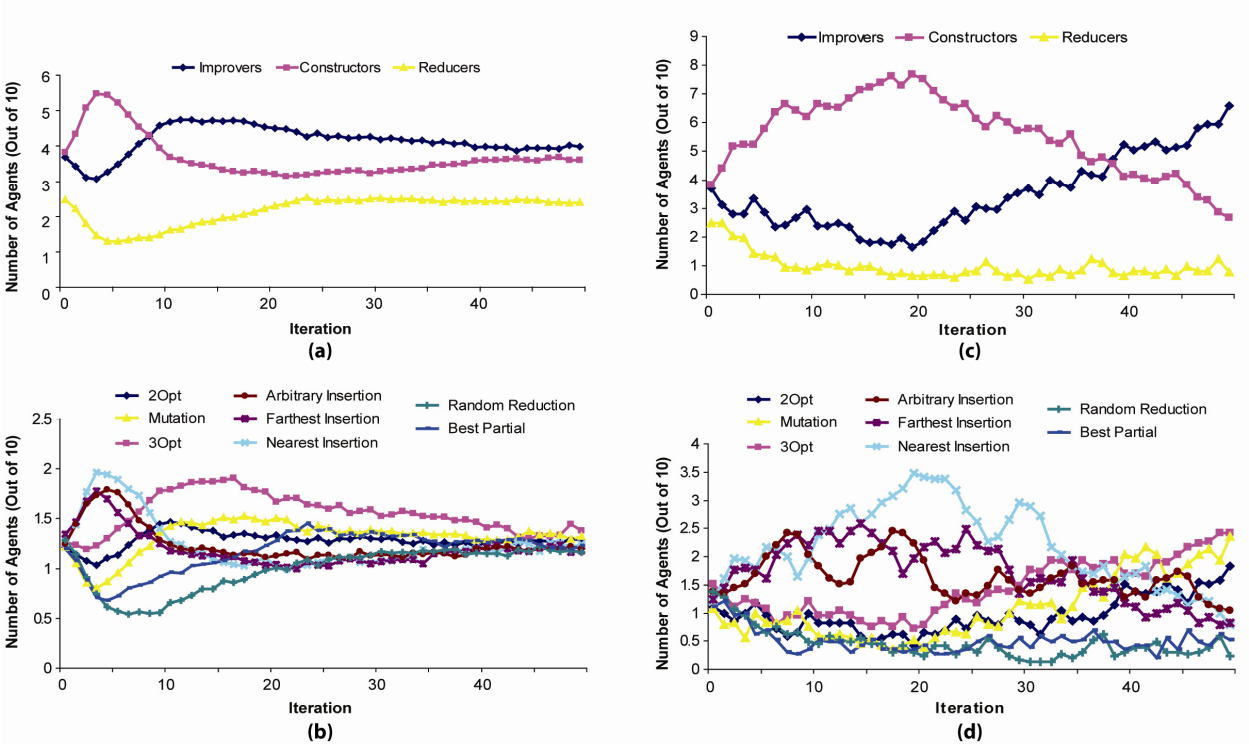

Fig. 3 Category and individual algorithm behavior for ATT48 (a and b) and for ATT532 (c and d)

Though the exact reason for the patterns is unclear, it is easy to confirm whether or not those patterns have an effect on the quality of the solutions generated by the EMAS algorithm. To do this, five individual trials of EMAS were run on ATT48 and the resulting distribution of agents was compared against the averages shown in Fig. 3. To measure correlation between each trial and the average distribution, we used the following formula:

$$c_{tot} = c_{const.} + c_{imp.} + c_{red.}, \qquad \textbf{Equation 1}$$

Where $c_{const.}$, $c_{imp.}$, and $c_{red.}$ are correlation coefficients between the number of constructors, improvers, or reducers over the course of the sample trial and the average number of each in 1000 trials over the same number of iterations. In other words,

$$c = \frac{\sum_{iter}\left(x_{iter} - \bar{x}\right)\left(avg_{iter} - \overline{avg}\right)}{\sqrt{\sum_{iter}\left(x_{iter} - \bar{x}\right)^2 \sum_{iter}\left(avg_{iter} - \overline{avg}\right)^2}} \qquad \textbf{Equation 2}$$

Where

- x_{iter} is the number of constructors, improvers, or reducers at a specific iteration in the sample trial,
- $\overline{x}$ is the average number of constructors, improvers, or reducers over the course of the sample trial (100 iterations),
- avg_{iter} is the average number of constructors, improvers, or reducers found at a specific iteration in 1000 trials of ATT48. For example, according to Fig. 3 (a), avg_8 for constructors and improvers is approximately 4.2, since both constructors and improvers represent 4.2 out of 10 agents in the team at the 8th iteration in the average of 1000 trials.
- $\overline{avg}$ is the average over all iterations of the number of constructors, improvers, or reducers found in the average of 1000 trials.

Our results for relating total correlation coefficient to solution quality are shown in Fig. 4. It was found that trials which correlated more strongly with the average patterns had better final values (in terms of distance from the optimal value of the average solution quality in the complete memory after 100 iterations) than those that did not.

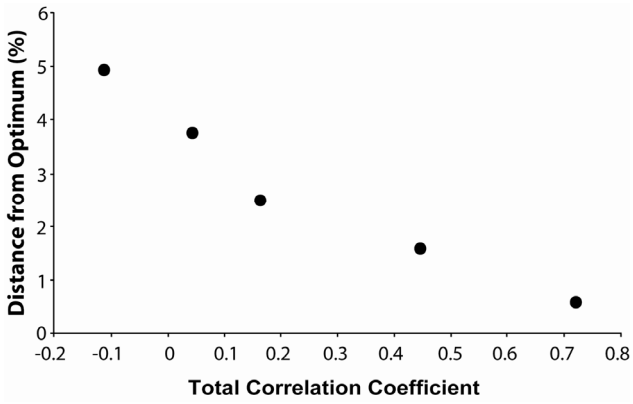

Fig. 4 Total correlation coefficient of agent behavior to average of previous trials vs. solution quality for 5 new trials of EMAS on 48-city problem

These results suggest that the EMAS algorithm may have potential as a predictive or learning tool. The patterns discovered in many trials of EMAS may be exploited to achieve similar results without actually employing evolution, just as experimentation in [35] identified heuristic rules for cooling schedules in simulated annealing rather than costly analysis for temperature reduction. In the current implementation of EMAS, computational runtime is expensive, especially as problem complexity increases. Forcing conditions on a complex problem that for a less complex instance correlate to good solution quality requires less of a computational commitment than running the entire evolution on the more complex instance. Thus it is possible that we may further decrease computation time while maintaining solution quality if EMAS is employed on a similar, less complex instance of a problem and its results extrapolated to a more complex instance. To test this, the averaging results of the 48-city problem, given in Fig. 3 (a) and (b) were applied to a new cooperative but un-evolving agent team attempting the more complex 532-city problem. In other words, rather than being influenced by the previous generation of agents, each iteration the probability of each agent having a particular strategy (algorithm, choice method, etc.) was guided by the results of the 1000 trials of ATT48. For example, at the 10th iteration in the new un-evolving team scenario, each agent had a 55% chance of employing a construction algorithm since averaging 1000 trials showed 5.5 agents out of 10 running construction algorithms at that iteration. As in the randomly assigned teams scenario, the forced conditions scenario was run with the same team size and for the same number of iterations as EMAS, with the cooperation protocol maintained as well. The results of this experiment are given in Fig. 5 and compared to our previous results for both EMAS and the randomly ordered algorithms for the 532-city problem. The y-axis represents, as in the earlier tables and Fig. 4, the average of all trials, where the value given for each trial is the average solution quality in the complete memory at the end of the trial.

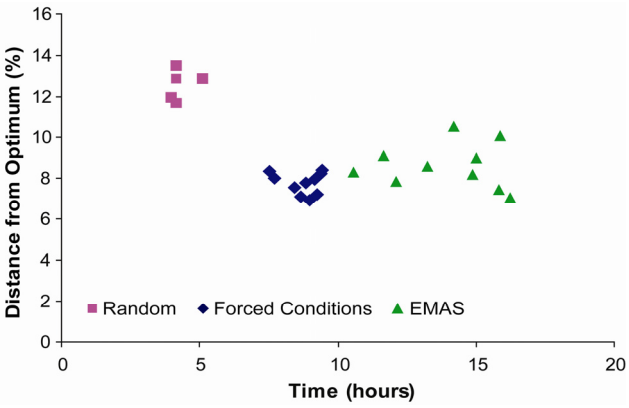

Fig. 5 Comparison of computation time and solution quality of forced conditions scenario to randomly assigned teams scenario and EMAS

As expected, forcing 48-city problem EMAS conditions on the agent population for a 532-city problem decreased computation time significantly from that of simply running EMAS on the 532-city problem. More surprising is that solution quality was maintained when forcing these conditions as well. On average, the solution quality generated by forcing previously determined EMAS behavior conditions on the agent population was more than 50% better than the random assignment of strategies to agents each iteration.

In addition to the 532 city problem, which from a landscape point of view is very similar to the 48 city problem, these patterns were also applied to a team solving a Euclidean 237 city problem modeled on a VLSI layout. This 237 city problem is essentially a manufacturing planning problem where the "cities" represent locations for holes to be drilled and the objective is to minimize the distance traveled by the drill head. In spite of the topological and cost function differences between this problem and the 48- and 532-city problems, results similar to those shown for the 532-city problem in Fig. 5 were found. Forcing the conditions of the 48 city problem on the 237 city problem yielded solutions that were almost 70% better than those produced by randomly assigned teams and equivalent to 50

iterations of EMAS – but with a 30% reduction in the computation time of EMAS. The fact that the TSP is so well defined and the constraints and objectives are so similar between the less complex 48-city problem and the more complex 237- and 532-city problems may be partially responsible for the effectiveness of the extension of these patterns. For some design problems, in which the topology, dimension, and size of the design space changes more drastically and unpredictably with increasing complexity, such extension may not be as fruitful. Additional research is needed to establish for certain the potential learning or predictive capability of EMAS for such problems.

7 Potential Applications

We have so far presented the core framework and methodology for utilizing an evolving virtual team of agents for adaptive optimization. The framework is designed to be flexible enough to be effective even in dynamic environments, where the design space size, characteristics, and topography may be changing. Such environments are common in many types of applications for which we expect the EMAS framework would be advantageous. Three potential contributions of EMAS, applicable to many areas of engineering design, are 1) its ability to produce good solutions without the user having to fine-tune optimization parameters or even choose which algorithm to apply, 2) its potential as a prediction and learning tool for guiding the optimization process, and 3) its flexibility in the face of a constantly changing design space. Optimization problems where several algorithms are available and the parameters need to be set and adjusted due to the complexity of the problem are particularly appropriate for EMAS, especially if the environment is changing. One such application is product layout, in which the number, size, and shape of components may change as the overall problem evolves, moving, for instance, from generally defined cubes to more complex geometries as more information becomes available. Though several effective layout algorithms have been developed in recent years [36,37], important parameters for producing good solutions (such as step size, annealing and resolution schedule, etc.) are usually unknown initially and need to be fine tuned by the user based on the specific problem. Another such application is the optimization of manufacturing processes for efficiency, cost, and energy usage, which has close ties to TSP and as mentioned in the previous section has been explored in this work.

The EMAS virtual team of solution strategies cooperates and evolves over time, producing as a byproduct solutions of increasing quality, potentially even in the face of changing problem design space. In addition, the knowledge gleaned from EMAS, as we showed in the previous section, may be used to predict or learn which parameters or algorithms should be used in the future on larger and more complex problems to decrease computation time.

8 Summary and Future Work

The results present a convincing argument for the evolution of agents in a cooperative virtual team at the population level. The cooperative teams of individual strategies evolved to generate better solutions than both individual strategies alone and *a priori* set hybrid strategies. It has been shown that the strength of the EMAS algorithm lies in its ability to evolve the best team of agents dynamically. Removing the evolutionary element by randomly assigning strategies within a team has been shown to be inferior to the EMAS model. We thus argue that the use of cooperative evolutionary agents to determine the best solution strategies dynamically is a strong approach to adaptive optimization. Decentralizing the strategy selection process and allowing strategies, rather than solutions, to progress in an evolutionary manner means that a much broader range of design and optimization applications, including layout, scheduling, and manufacturing planning, can be confronted in the face of design space uncertainty and change. Another strength of the EMAS algorithm is as a predictive or learning guide for which set of algorithms or strategies should or should not be employed and when. Utilizing EMAS in this way, at least as seen on a static case of the TSP, has been shown to lower computation time while maintaining or even improving solution quality.

In the future, our approach to adaptive optimization utilizing evolutionary multi-agent teams will be applied to engineering design optimization problems, and several extensions will be made. A primary focus of additional future research will be in specifically addressing dynamic design environments – those in which the actual structure of the solution, or the number of variables in the design problem, may change over the course of optimization. This extension to our preliminary work will provide an even more concrete example of the relevance of our research to engineering design applications in which flexibility to dynamic design environments is desired.

Acknowledgements

This research was sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA95500710225. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR or the U.S. Government.

References

[1] Campbell, M.I., J. Cagan, K. Kotovsky, 1999. "A-Design: An Agent-Based Approach to Conceptual Design in a Dynamic Environment," *Research in Engineering Design* **11**(3): 172-192.

[2] Olson, J. T., and J. Cagan, 2004. "Inter-Agent Ties in Computational Configuration Design", *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, *(Special Issue on Agent-Based Design)*, **18**(2):135-152.

[3] Branke, J. 2002. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Norwell, MA.

[4] Franklin, S. and A. Graesser, 1997. "Is it an agent, or just a program?: A taxonomy for autonomous agents," *Lecture Notes in Computer Science* 1193: 21-35.

[5] Russell, S.J. and P. Norvig, 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

[6] De Souza, P.S. 1993. "Asynchronous Organizations for Multi-Algorithm Problems" Ph.D. Dissertation, Carnegie Mellon University, Department of Electrical and Computer Engineering.

[7] Sachdev, S. 1998. "Explorations in Asynchronous Teams" Ph.D. Dissertation, Carnegie Mellon University, Department of Electrical and Computer Engineering.

[8] Talukdar, S., L. Baerentzen, A. Gove, P. De Souza. 1998. "Asynchronous Teams: Cooperation Schemes for Autonomous Agents". *Journal of Heuristics*, **4**: 295-321.
[9] Rice, J.R. 1976. "The Algorithm Selection Problem," in M. Rubinoff and M.C. Yovitz, eds. *Advances in Computers* **15**: 65-118.
[10] Gomes, C.P., B. Selman. 2001. "Algorithm Portfolios" *Artificial Intelligence*. **126**: 43-62.
[11] Nudelman, E., G. Andrew, J. Mcfadden, K. Leyton-Brown, Y. Shoham, 2003. "A portfolio approach to algorithm selection," *Proc. IJCAI-03, 18th Int'l. Joint Conf. on Artificial Intelligence*.
[12] Giraud-Carrier, C., R. Vilalta, P. Brazdil, 2004. "Introduction to the special issue on meta-learning," *Machine Learning*, **54**(3): 187-193.
[13] Schmidhuber, J. 2004. "Optimal Ordered Problem Solving," *Machine Learning* **54**(3): 211-254.
[14] Moral, R.J., D. Sahoo, G.S. Dulikravich, 2006. "Multi-Objective Hybrid Evolutionary Optimization with Automatic Switching," *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*.
[15] Gagliolo, M. and J. Schmidhuber. 2007. "Learning Dynamic Algorithm Portfolios," *Technical Report No. IDSIA-02-07*.
[16] Back, T., D.B. Fogel, Z. Michalewicz, eds. 2000. *Evolutionary Computation 1*. Institute of Physics Publishing.
[17] T. Back, D.B. Fogel, Z. Michalewicz, eds. 2000. *Evolutionary Computation 2*. Institute of Physics Publishing.
[18] Fogel, D.B. 2000. "What is Evolutionary Computation?" *IEEE Spectrum*.
[19] Koza, J.R. 1992. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press.
[20] Potter, M.A. and K.A. De Jong, 2000. "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents," *Evolutionary Computation* **8**(1): 1-29.
[21] Cristea, P., A. Arsene, B. Nitulescu, 2000. "Evolutionary Intelligent Agents," *Proc. 2000 Congress on Evolutionary Computation* **2**: 1320-1328.
[22] Padberg, M. and G. Rinaldi. 1987. "Optimization of a 532-city symmetric traveling salesman problem by branch and cut." *Operations Research Letters* **6**: 1-7.
[23] Applegate, D.L., R.E. Bixby, V. Chvatal, and W.J. Cook. 2006. *The Traveling Salesman Problem*. Princeton University Press, New Jersey, USA.
[24] Concorde TSP solver webpage: www.tsp.gatech.edu/concorde.html
[25] Golden, B.L. and W.R. Stewart, 1985. "Empirical Analysis of Heuristics." In *The Traveling Salemsan Problem*, E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan and D.B. Shmoys, eds. John Wiley.
[26] Bentley, J.L. 1990. "Experiments on Traveling Salesman Heuristics." *Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, PA. pp. 91-99.
[27] Syslo, M.M., N. Deo, and J.S. Kowalik. 1983. *Discrete Optimization Algorithms with Pascal Programs*. Prentice Hall, Englewood Cliffs, NJ.
[28] Rothlauf, F. 2002. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag.
[29] Deb, K. "Encoding and Decoding Functions," in [30] Wooldridge, M. and N.R. Jennings. 1995. "Intelligent Agents: Theory and Practice" *The Knowledge Engineering Review*. **10**(2): 115-152.
[31] Grefenstette, J.J., R. Gopal, B.J. Rosmaita, and D. Van Gucht. 1985. "Genetic Algorithms for the Traveling Salesman Problem". *Proc. 1st Int'l Conf. on Genetic Algorithms*. pp. 160-168.
[32] Potvin, J. 1996. "Genetic Algorithms for the Traveling Salesman Problem". *Annals of Operations Research*, **63**(3): 337-370.
[33] Eiben, A.E. and J.E. Smith, 2003. *Introduction to Evolutionary Computing*. Springer-Verlag.
[34] Wolpert, D.H. and W.G. Macready. 1997. "No Free Lunch Theorems For Search," Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, USA.
[35] Swartz W., and C. Sechen (1990), "New Algorithms for the Placement and Routing of Macro Cells," in *Proceedings of the IEEE Conference on Computer-Aided Design*, Santa Clara, CA, November 11-15, IEEE proceedings: Cat No. 90CH2924-9, pp. 336-339.
[36] Cagan, J., K. Shimada, and S. Yin. 2002. "A Survey of Computational Approaches to Three-dimensional Layout Problems", *Computer Aided Design*, **34**(8): 597-611.
[37] Tiwari, S., G. Fadel, P. Fenyes, 2008. "A Fast and Efficient Compact Packing Algorithm for Free-Form Objects," to appear in *Proc.ASME IDETC/CIE 2008.*

List of Figures

Fig. 1 Structure of proposed EMAS agent chromosome
Fig. 2 Flowchart of agent activation
Fig. 3 Category and individual algorithm behavior for ATT48 (a and b) and for ATT532 (c and d)
Fig. 4 Total correlation coefficient of agent behavior to average of previous trials vs. solution quality for 5 new trials of EMAS on 48-city problem
Fig. 5 Comparison of computation time and solution quality of forced conditions scenario

List of Tables

Table 1 Placement of EMAS in relation to relevant evolutionary computing concepts and literature
Table 2 Mean distance (%) from optimum solution and standard deviation of hybrid algorithms compared to EMAS algorithm for 48- and 532-city problems
Table 3 Comparison of EMAS to randomly generated team for ATT532 (averages of 50 trials)