

Final Report: SAGA 3

(03-DG-11244225-390)

December 24th, 2004

W. D. Potter, P. W. Brooks, S. Vikram

Abstract

This report documents the latest work in the SAGA (Spray Advisor using Genetic Algorithms) project, the goal of which is to provide an effective means for a computer to recommend an optimal set of parameters for spraying a wooded area with pesticide. Current work is in two directions. One of these is the maintenance of older SAGA programs, some of which were built on now-obsolete platforms. The other is the development of a unified framework, SAGA3, to maximize code reuse and provide a consistent and flexible interface across SAGA programs.

Brief History of the SAGA Project

The most recent work in the SAGA project builds upon what previous researchers have already accomplished. This section briefly describes the history of the SAGA project and its successes to set the stage for discussion of the issues facing SAGA and how we resolve them.

SAGA originated as a prototype generational genetic algorithm (henceforth GA) written in Fortran. To compute the fitness of an individual, it created a text file describing the individual and ran a standalone DOS version of AgDisp on it. Fortran-SAGA proved the feasibility of applying evolutionary computation methods to this domain, but it suffered from a crude interface and long running time. Its successor, VB-SAGA, was written in Visual Basic 5. It featured a friendlier graphical user interface and linked to a DLL of AgDisp instead of calling a standalone executable, which resulted in significantly improved running time. A second version of it improved usability and implemented self-adaptation of GA parameters. Yet VB-SAGA still used more or less

the same simple generational GA as Fortran-SAGA, which limited its performance. These early versions of SAGA are described by Bi (2000).

SAGA2 was the first of the second generation of SAGA programs (Wu 2002). The second generation differed from the first in two major respects: they were written in C++ for greater speed, and they featured a user interface more consistent with other Windows programs. SAGA2 in particular also replaced the generational GA of VB-SAGA with a steady-state GA, which replaces only a portion of the population each cycle. This means that a newly generated individual can begin reproducing immediately, especially if it has a relatively high fitness. Steady-state GAs tend to converge faster than their generational counterparts. In the case of aerial spraying, this resulted in better and more reliable performance.

SAGA2NN, another second generation SAGA program, randomly generated a large number of potential solutions and chose the best for its initial population using a neural network trained using AgDisp. Although not as accurate as AgDisp itself, the neural network-based fitness estimation took significantly less time to run (thus making the approach feasible). In experiments, SAGA2NN's early populations were considerably better than those of the other SAGA programs, but sometimes fell behind by the end. These results suggest that SAGA2NN would be most useful in situations where time is limited.

SAGADO, the final notable member of the second generation, implemented GADO (Genetic Algorithm for Design Optimization), a method for generating parameters for artifact design (Rasheed 1998). Of all the SAGA programs, SAGADO exhibited the best overall performance.

The Issues

SAGA is aging. The early Visual Basic versions were written in Visual Basic 5 (henceforth VB5), which is now obsolete. It has been replaced by Visual Basic 6 (henceforth VB6) and Visual Basic .NET (henceforth VB .NET). VB6 is a revision of VB5 with some new capabilities and other mostly minor differences, but VB .NET is an entirely new and incompatible language. Updating to VB6 requires only making small changes in the program, but moving to VB .NET means rewriting most of the program. We therefore chose to move SAGA to VB6.

SAGA has no common user interface. Because each program was developed separately from the others without being built around a common user interface, users of the different programs must familiarize themselves with different GUIs. Furthermore, the existing SAGA programs lack an API, an interface through which other programs can control them and interact with them. In order for another piece of software to call on SAGA to recommend a set of spray parameters, either it must walk a user through running SAGA and reporting the results, or the relevant code must be copied and edited to fit. Either way is time-consuming and inconvenient.

Evolutionary computation is domain-neutral. It does not rely on special knowledge about the domains in which it is used, hence it is applicable to many different domains. So far, SAGA has focused exclusively on forest pest control, but the methods it uses are applicable to other domains such as rangeland herbicide or even ones outside of forestry. In spite of this, one would have to rewrite and debug each program separately in order to apply SAGA's methods to a new domain.

Finally, SAGA is difficult to maintain and debug. Each SAGA program is completely self-contained, with its own user interface, fitness function, and so on. Much

of the code contained in each program is redundant, reproduced (sometimes in different ways) in other programs. As a result, to fix a bug in these common portions requires each one to be modified separately. As noted before, to switch the domain or modify the user interface also requires different changes to each program. Reducing redundancy means creating a central repository of code that also reduces the amount of programming required to implement new optimization algorithms or problem domains for experimentation.

The SAGA3 Framework

One way to solve the problems described in the previous section is to design a foundation for developing and using evolutionary computation, and rebuild the SAGA project around it. Such a foundation must provide a common user interface and API for all optimization methods, a central domain description which can be changed to allow work in other domains, and a library of code for performing common tasks. We developed SAGA3 along these lines. This sort of approach was not used in previous SAGA efforts because of the changing nature and timing of the project. Now that the effectiveness of evolutionary computation for optimizing spray parameters is established, the next step is to focus on making SAGA easier to use and further develop.

SAGA3 is an object-oriented framework for evolutionary computation. Objects are organizational units within a program that act like intelligent data. They contain both declarative information and procedures to act upon that information. An individual object is an instance (or token) of a class (or type), like a particular car might be an instance of the type Volvo. Classes are organized hierarchically, with some classes inheriting features from others. The inheriting class is called the subclass or derived

class, and the class inherited from is called the superclass or base class. Continuing the car analogy, Volvo is a subclass of automobile.

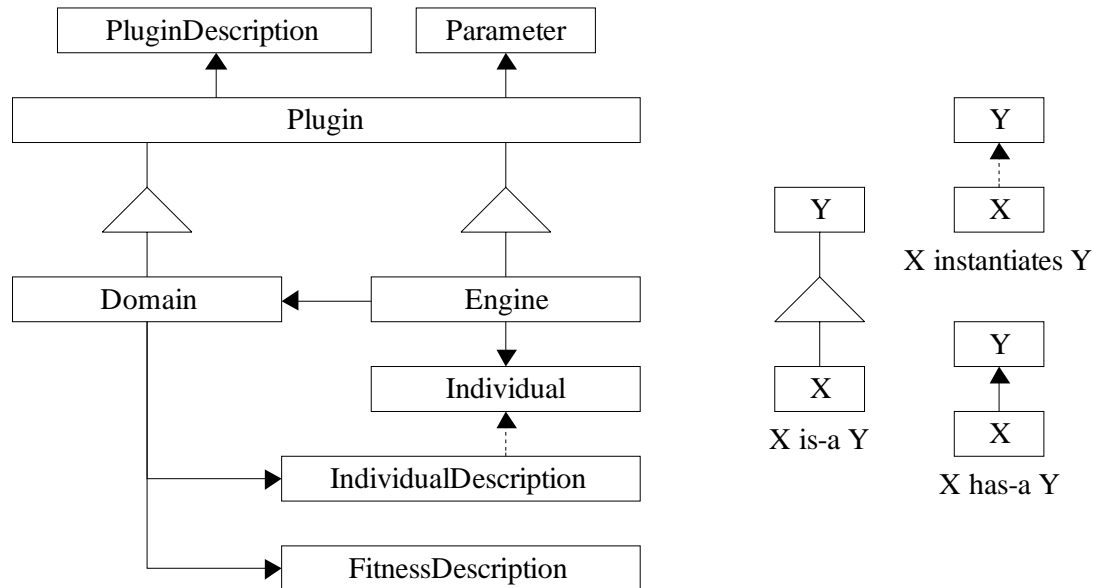


Figure 1. SAGA3 Architecture

The object-oriented programming paradigm is useful because it provides a natural way to divide programs into smaller units, and it encourages code reuse through inheritance. In the case of SAGA3, we define two types of objects as plugins, which a program (for example, SprayAdvisor) can load at runtime. Domain plugins represent a problem domain, such as forest pest control, providing a representation of possible solutions and a fitness function for evaluating them. Engine plugins represent domain-neutral optimization engines such as GADO and the simple generational GA used by VB-SAGA. To use SAGA3, a program dynamically links to one of each kind of plugin and performs the configuration. In the case of Engine, the configuration process involves specifying the Domain plugin to use. The program can then let Engine run and request its results when needed.

These two plugin classes all inherit from the base class `Plugin`, as illustrated in Figure 1. `Plugin` provides such basic services as configuration and identification. It contains zero or more `Parameter` objects, each of which defines a configuration parameter. A `Parameter` object manages the name, description, valid values, and default values of the configuration parameter it represents, and can be given a new value by the program linking to the plugin. `Plugin` also contains a `PluginDescription` object, which describes the plugin by providing its name, version, type (`Engine` or `Domain`), and a short description. An application linking to the plugin uses its `PluginDescription` object to get information about it.

A `Domain` object contains two more objects, `IndividualDescription` and `FitnessDescription`. The former defines how individuals within the domain behave. It includes such information as the number of genes (the parameters for the fitness function), and the type and valid ranges for each one (the latter of which the user can change at runtime). SAGA3 currently works with three different kinds of genes. Real genes approximate real numbers. Within the domain of forest pest control, swath width is such a gene. Cardinal genes have integer values that are not ordered, so that 2 is no more similar to 3 than it is to 7. An example of a Cardinal gene is the aircraft number. Ordinal genes have integer values such that order does matter, so 2 is closer to 3 than it is to 7. An example of an Ordinal gene is the number of nozzles. The types of genes are differentiated so that the optimization method can perform appropriate crossover and mutation operations on them. Several previous versions of SAGA treated all genes as real numbers.

Previous versions of SAGA were also limited in that to change the ranges of a gene (necessary to run John Ghent's scenarios, below), one had to modify the source

code and recompile. At runtime, one could only decide to let the gene roam freely over the entire range allowed by `AgDisp` or set it to one value from which it could not vary. SAGA3's `IndividualDescription` class allows the user to change the range of a gene at runtime.

`IndividualDescription` acts as a factory for `Individual` objects, meaning it can create (instantiate) new instances of the class for use by the engine. An `Individual` object comprises a sequence of genes (patterned after the `IndividualDescription`) and knows how to compute its own fitness. It serves as the interface to the domain's fitness function.

`FitnessDescription` describes the fitness function(s) in a domain. The forest pest control domain has 3 fitness functions: drift fraction, COV, and VMD. Every version of SAGA to date has combined these into a single fitness function by taking a weighted sum of these values, but there are also multi-objective methods (not yet tried by SAGA) that take each fitness function into account separately. The SAGA3 framework supports either approach. It allows a `Domain` plugin to recommend weights for generating a single fitness value, but still provides access to the individual functions for multi-objective engines or allows the possibility of adding other functions such as efficacy.

Engine objects are optimization engines. Given a fitness function (which they access through `Individual` objects), they attempt to find the parameters that return the most desirable results. In order to do this, they need an object from the `Domain` class, provided by the user. This describes the fitness landscape and provides a source of `Individuals` for manipulation.

To use SAGA3, one chooses a problem domain (represented by a `Domain` object) and an optimization engine (`Engine` object). These may need to be configured, by changing parameters unique to the components (such as the `VMDCenter`), customizing the weights of various fitness functions, or setting limits on the search space (by changing the range of values a given gene can take on). Once this is done, the `Engine` object is given an instruction to run. It then does its job, periodically updating the user on its progress. It can run until completion or the user can choose to halt it prematurely. In either case, its findings are reported to the user..

This process of configuring and using the plugins is performed by a UI (user interface) application that is aware of the SAGA3 Framework. The same UI can be used with any SAGA3 plugin, resolving the issue of multiple, inconsistent user interfaces. It need not be a GUI. For example, the experiments to test SAGA3 (described later) were performed using a simple UI that read in a list of instructions, each describing a separate run, which the computer executed one by one without human intervention. This allowed us to do the experiments and tally the results easily. It is also possible for a larger program, such as a suite of forestry management tools, to link to the SAGA3 Framework, allowing it to seamlessly call SAGA3 optimization engines with minimal effort on the part of the developers. As noted earlier, previous versions of SAGA had to be partially rewritten to be integrated into larger applications.

In addition, a graphical user interface to SAGA3 is nearing completion. It is written in Microsoft Visual C++ .NET, and takes advantage of Windows Forms to provide a familiar Windows interface. The user describes a batch of tasks for the program to carry out. Each task can use a different engine and domain plugin and can be configured separately. The user can also choose to repeat each task any number of times

to find the best results over multiple runs. Screenshots of the .NET UI are shown in Figures 2 – 4.

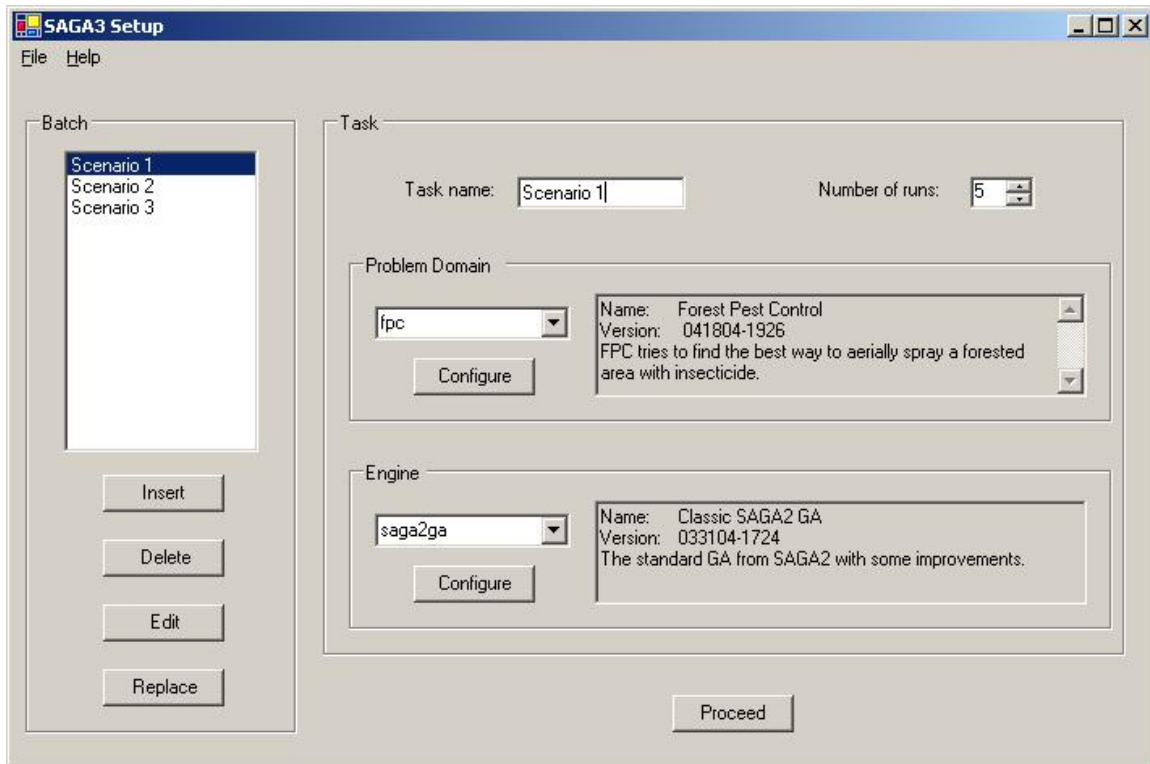


Figure 2. .NET GUI Task Configuration

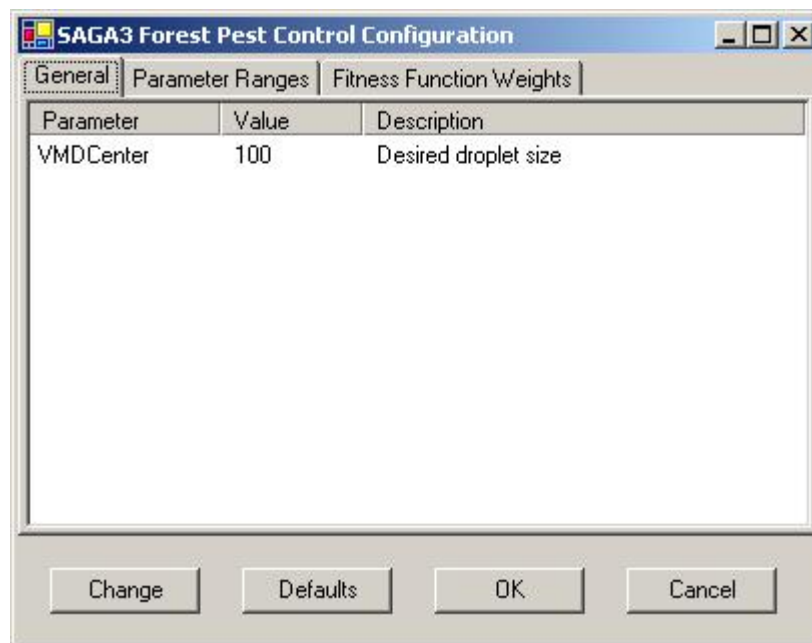


Figure 3. .NET GUI Domain Plugin Configuration

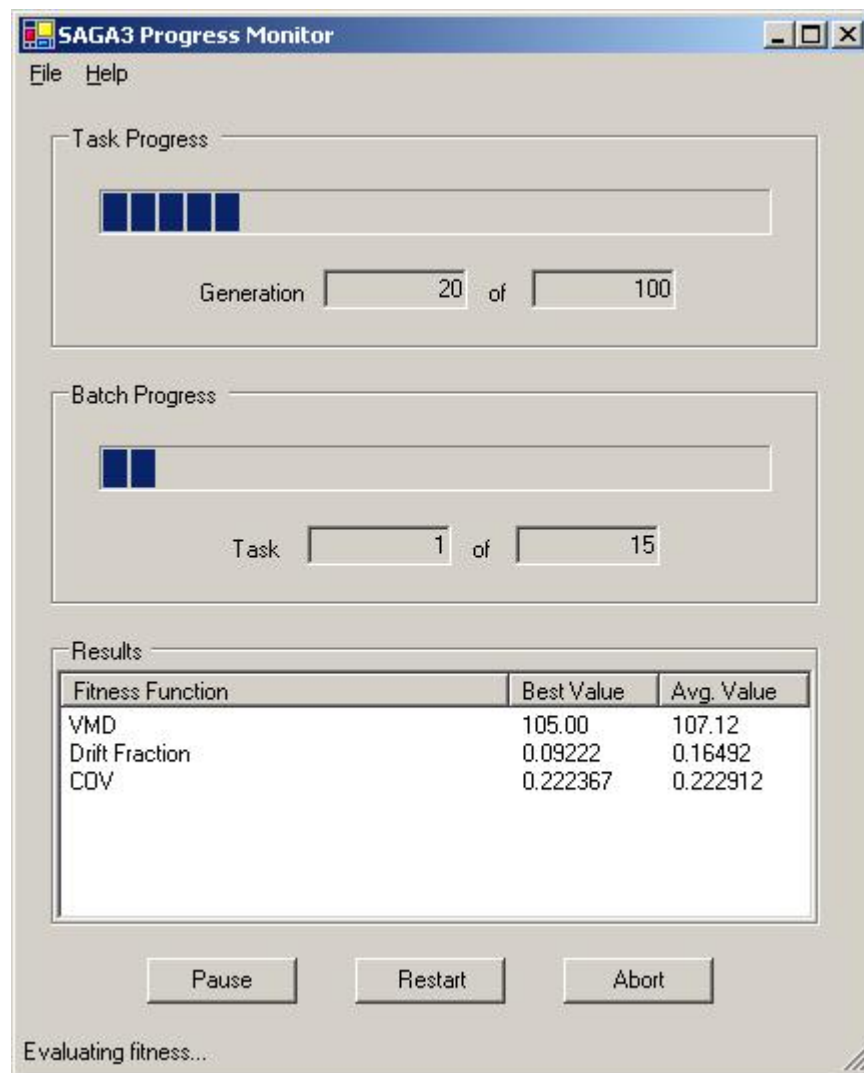


Figure 4. .NET GUI Progress Monitor

Another advantage of the SAGA3 Framework is its portability. SAGA3 (both the Framework and its currently-existing components, aside from the .NET GUI) is written in ANSI-standard C++. In principle, it should be able to work on any platform with an ANSI-standard C++ compiler and dynamic linking. This includes all current PC platforms (Windows and Macintosh since at least 1995) and also some more exotic

platforms such as Solaris and Linux. In practice, small changes may have to be made depending on the particular platform's handling of dynamic linking.

SAGA3 Plugins

At the present time, there is only one `Domain` plugin, FPC (Forest Pest Control). FPC's individuals are sets of parameters for the AgDisp simulator, and its fitness values are AgDisp's results: VMD, COV, and drift fraction. This is the same domain that previous versions of SAGA have used, except that the infrastructure is now in place for multi-objective optimization methods. It uses the fitness function described by Wu (2002).

The first `Engine` component designed for SAGA3 is SAGA2GA. It is an adaptation of the "classic" SAGA2 genetic algorithm to the SAGA3 Framework. It is not exactly equivalent in that parts of it have been rewritten for efficiency and ease of maintenance and use. As the experiments will show, it is roughly equivalent to a second-generation SAGA program in performance.

The second `Engine` plugin is OEGADO. It is based on the OEGADO algorithm described by Chafekar (2003). Unlike the other methods used in the SAGA project to date, OEGADO is multi-objective. Instead of computing a weighted sum of multiple fitness functions to simulate a single fitness function, OEGADO optimizes each individual fitness function in turn. After the initial population is generated, OEGADO makes copies of it and evolves each copy to maximize a different fitness function.

Theoretically, a multi-objective GA should find a diverse set of solutions on the Pareto front, the limit in the fitness landscape where the value of one fitness function cannot increase without decreasing the value of another. The user then has a variety of

solutions from which to choose, some of which might be better in some ways at the expense of others. In the tests run, OEGADO did not fully explore the Pareto front but instead concentrated its populations at extremes, finding individuals that, for example, had very low drift fraction but high covariance and a droplet size far afield of what was desired. As a result, the individuals had low overall fitnesses and OEGADO did not compare well with other recent SAGA methods.

The last SAGA3 Engine component is SAGA3ES, which implements an Evolutionary Strategy (henceforth ES) (Beyer 2001). ESs exhibit self-adaptation, meaning they evolve some of their own settings at the same time that they evolve individual parameters. The second version of VB-SAGA, for example, used an advanced self-adaptation method based on fuzzy sets to evolve the frequency of different mutation and recombination operators. On the other hand, SAGA3ES uses a simple method to evolve the greediness of its sole mutation operator, Gaussian mutation.

Gaussian mutation works by randomly choosing a number from the Gaussian distribution with mean 0 and a variable standard deviation. This random number is then added to the gene being mutated to find its new value. Due to the shape of the Gaussian distribution, the gene is more likely to make a small change than a large one. Exactly how much more likely is determined by the standard deviation of the distribution, which is evolved along with the genes themselves.

SAGA3ES performed extraordinarily well in the scenario where all values were allowed to float, but when the ranges of the parameters were limited it fell behind the other methods. This suggests that in spite of its ease of use (the only settings that a user needs to set are population size and number of evaluations because the rest are evolved),

it may not be very useful for real-world situations where there are restrictions on what can be allowed.

Experiments and Results

SAGA3 and the revised VB-SAGA were tested in two ways. First, we recreated the experiments performed by Wu (2002). There, VB-SAGA and the programs of the SAGA2 project were given three parameter sets to optimize (shown in Figure 2). Each program was run five times for each parameter set. For each run the fitness function was called 5,000 times. SAGA2 and SAGA2NN used a population size of 100, a generation gap of 20, a tournament size of 2, a crossover rate of 1.0, and a mutation rate of 0.1 per individual (decreasing over time).¹ We ran SAGA3 (using the SAGA2GA and OEGADO) over the same parameter sets, also using a population size of 100, generation gap of 20, a tournament size of 2, and a crossover rate of 1.0, but with a mutation rate of 0.01 per gene. SAGA3ES was run with a population size of 100. All fitness calculations assumed a VMDCenter of 100.

Our SAGA3 results and Wu's SAGA2 results for each run and parameter set are shown side-by-side in Figure 3. The **Run #** rows show the fitness of the best individual in the population after 5,000 calls of AgDisp in each of the 5 runs of each program. The **Mean** row contains the average result for the program over all 5 runs, and the **Worst** row contains the worst result of that program out of the 5 runs. The best of the 5 runs for each program is shown in bold type.

¹In practice, the mutation rate was only 0.01. SAGA2 and SAGA2NN check for mutation twice for each individual, and only actually perform mutation if both checks are positive. When they do perform mutation, they change every gene in the individual. The SAGA2GA plugin performs mutation per gene and does not check twice.

Our first trials using the revised VB-SAGA did not compare with the VB-SAGA results reported by Wu (2002). At present, we have not been able to exactly reproduce the circumstances that produced his results. The difference is not due to the changes between versions of Visual Basic because the older versions perform the same. By varying the GA parameters, we were able to bring our results within a small distance of Wu's. We used a population of 300 individuals over 50 generations, for a total of 15,000 fitness evaluations. We also used a tournament size of 2, 4, and 6, keeping track of results separately for each.

We performed the experiments using both the static and dynamic versions of VB-SAGA. The static version is the original VB-SAGA. Its GA parameters are fixed for the length of the simulation. The dynamic version is the self-adapting VB-SAGA. It uses fuzzy logic to attempt to adjust the crossover and mutation rates to optimal values during the simulation. For the static GA, we used a crossover probability of 0.75 and a mutation probability of 0.02. The results we obtained (using the same fitness function and VMDCenter as the previous tests) from the static GA are in Figure 4, alongside Wu's for comparison. For the dynamic GA, we used an initial crossover probability of 1.0 and mutation probability of 0.1. The results obtained using the dynamic GA are in Figure 5, again alongside Wu's for comparison. By the end of the run, the dynamic GA had adjusted the mutation rate to a very low value. We speculate the results might be improved by forcing the mutation rate higher toward the end of the run to better escape local optima.

For the second test, John Ghent provided us with eight plausible field scenarios, shown in Figure 6. We ran SAGA3 with SAGA2GA five times for each one using the same VMDCenter, population size, generation gap, tournament size, mutation rate, and

crossover rate as in the first test. The best individuals from each scenario are in Figure 7.

Note that boom height is measured in meters, and wind speed in meters per second.

Scenario	Constraints
Parameter Set I	None
Parameter Set II	Aircraft ID = 6, Swath Width = 1.2, VMD = 100, Block Width = 400
Parameter Set III	Aircraft ID = 106, Swath Width = 2.25

Figure 5. Lei Wu's Scenarios

Parameter Set I						
Run #	SAGA2*	SAGA2NN*	SAGADO*	SAGA2GA	OEGADO	SAGA3ES
1	9966.31	9927.87	9973.24	9960.23	9657.28	9971.58
2	9967.71	9963.99	9951.65	9960.34	9657.54	9982.95
3	9971.04	9969.70	9975.32	9949.58	9621.45	9970.81
4	9963.79	9938.80	9973.34	9927.75	9619.48	9972.54
5	9952.54	9970.73	9982.42	9935.62	9642.80	9956.91
Mean	9964.28	9954.22	9971.19	9946.70	9639.71	9970.96
Worst	9952.54	9927.87	9951.65	9927.75	9619.48	9956.91
Parameter Set II						
Run #	SAGA2*	SAGA2NN*	SAGADO*	SAGA2GA	OEGADO	SAGA3ES
1	9655.79	9656.42	9656.15	9649.86	9512.48	9394.15
2	9650.91	9658.32	9649.60	9648.65	9564.25	9624.58
3	9650.61	9632.53	9651.33	9642.90	9561.41	9604.28
4	9655.79	9655.79	9658.13	9658.31	9551.15	9582.65
5	9632.07	9651.44	9654.29	9639.89	9567.27	9599.30
Mean	9649.03	9650.90	9653.90	9647.92	9551.31	9560.99
Worst	9632.07	9632.53	9649.60	9639.89	9512.48	9394.15
Parameter Set III						
Run #	SAGA2*	SAGA2NN*	SAGADO*	SAGA2GA	OEGADO	SAGA3ES
1	9582.10	9583.19	9598.99	9584.81	9512.56	9530.48
2	9588.06	9553.53	9608.46	9586.00	9512.84	9515.60
3	9599.51	9575.86	9589.43	9619.22	9510.31	9567.28
4	9585.76	9699.12	9574.29	9603.30	9517.48	9595.01
5	9589.19	9622.69	9601.99	9581.61	9584.87	9545.70
Mean	9588.92	9606.88	9594.63	9594.99	9527.61	9550.81
Worst	9582.10	9553.53	9574.29	9581.61	9510.31	9515.60
* Results from Wu (2002)						

Figure 6. Results Using Lei Wu's Scenarios

Run #	Wu	Tournament Size		
		2	4	6
1	9926.20	9765.88	9828.57	9870.45
2	9943.15	9902.63	9884.27	9803.60
3	9951.56	9884.28	9842.00	9801.75
4	9926.03	9888.11	9880.30	9791.23
5	9925.44	9878.97	9846.20	9849.02
Mean	9934.48	9863.98	9856.27	9823.21
Worst	9925.44	9765.88	9828.57	9791.23

Figure 7. Static VB-SAGA Results

Run #	Wu	Tournament Size		
		2	4	6
1	9926.20	9852.55	9828.03	9809.46
2	9943.15	9783.95	9847.55	9745.49
3	9951.56	9825.22	9819.61	9870.95
4	9926.03	9842.34	9933.88	9790.37
5	9925.44	9776.32	9673.81	9866.69
Mean	9934.48	9816.08	9820.58	9816.59
Worst	9925.44	9776.32	9673.81	9745.49

Figure 8. Dynamic VB-SAGA Results

Future Work

The SAGA3 Framework has been revised in light of the experience of creating the first few plugins for it. Two more SAGA2 programs, SAGA2NN and SAGADO, remain to be ported to the revised framework. Once this is complete, the best SAGA algorithms so far will be unified under SAGA3.

There are several directions for new work under SAGA3. One of these is the further development of multi-objective engines. Unlike most other evolutionary computation methods that tend to identify a small portion of the individual space as interesting and focus on it, multi-objective methods explore the Pareto front – the edge of the fitness landscape where increasing one fitness function decreases the value of another. So far, OEGADO is the only multi-objective optimization method we have

tried. It performed poorly overall, but did find individuals with interesting characteristics, ones with a single very good fitness function. An improved OEGADO or other multi-objective method may be better able to explore the Pareto front of the forest pest control domain.

Self-adaptation is another area we could continue to explore. It is possible for an optimization engine to automatically find the best GA parameters (such as mutation rate), sparing the user from having to figure them out. VB-SAGA 2.0 used self-adaptation to considerable advantage over VB-SAGA 1.0, and SAGA3ES performed very well in certain situations. SAGA3ES could probably be improved by implementing correlated mutation, which allows the standard deviations for each gene affect one another as they evolve. This increases the ability of the mutation method to home in on good values. Evolutionary programming is another nature-inspired optimization method that takes advantage of self-adaptation and may be worth investigating.

The final area of possible development is to use SAGA to address different problems. The modularity of the SAGA3 Framework allows one to use the same optimization engines with different domains. Although at the current time we only have FPC, it is a straightforward task to develop domain plugins given a pre-existing fitness function. Rangeland and forest herbicide are two relevant domains to which SAGA could be adapted, and SAGA3 could be useful for domains other than pesticide spraying.

Scenario	Aircraft	Boom Width	Block Width	Nozzles	Swath Width
1	Bell 206 III	0.6 - 1.0	50 - 400	6	1.5 - 3.0
2	Bell 204	0.6 - 1.0	50 - 400	8	1.5 - 3.0
3	Hiller Soloy	0.6 - 1.0	50 - 400	6	1.4 - 2.1
4	Air Tractor 400	0.6 - 1.0	50 - 400	8	1.5 - 3.0
5	Air Tractor 802A	0.6 - 1.0	50 - 400	10	1.5 - 3.0
6	Ag-Cat Turbo	0.56 - 1.0	50 - 400	6	1.52 - 2.95
7	Antonev An-2	0.6 - 1.0	50 - 400	8	1.5 - 3.0
8	AgTruck 188	0.6 - 1.0	50 - 400	6	1.4 - 3.0

Figure 9. John Ghent's Scenarios

Gene	Scenario							
	1	2	3	4	5	6	7	8
VMD Input	100	100.054	100.368	100.522	114.659	101.114	100.65	100
Nonvolatile fraction	0.865827	0.864996	0.859818	0.763265	0.297331	0.745144	0.401564	0.745297
Wind speed	0.517896	0.230062	0.488683	0.23	0.23035	0.23	0.346443	0.230043
Temperature	1	20.1869	5.61062	23.1617	22.926	27.0711	1	1.00076
Relative Humidity	74.6122	15.5145	15	15	16.1569	30.4856	19.9134	46.3439
Boom Length	0.876057	0.769963	0.8774	0.931514	0.602167	0.891479	0.6	0.600906
Boom Height	8	12.1925	8.30444	15	8	8	8	10.5967
Block Width	397.988	319.033	399.926	399.99	389.393	400	399.851	399.962
Swath Width	1.5356	1.50365	1.50251	1.99626	1.5	1.58226	1.50456	1.6173
Fitness								
Drift Fraction	0.023098	0.017702	0.016199	0.047407	0.036977	0.031474	0.049645	0.041839
COV	0.063422	0.047792	0.06198	0.095344	0.192347	0.12869	0.134453	0.126208
VMD	100.259	100.227	100.027	100.792	99.9999	99.9883	99.9989	99.9951

Figure 10. Results from John Ghent's Scenario

Bibliography

- Beyer, Hans-Georg. *The Theory of Evolution Strategies*. Springer, 2001.
- Bi, Wanying. “Intelligent decision support system for USDA Forest Service aerial spray management.” M.S. thesis, University of Georgia, 2000.
- Chafekar, Deepti, Jiang Xuan, and Khaled Rasheed. “Constrained Multi-objective Optimization Using Steady State Genetic Algorithms.” To appear in *The Genetic and Evolutionary Computation Conference (GECCO'2003)*.
- Rasheed, Khaled. “GADO: A Genetic Algorithm for Continuous Design Optimization.” Ph.D. diss., Rutgers University, 1998; available at: <http://www.cs.uga.edu/~khaled/thesis.ps>; Internet.
- Wu, Lei. “A Comparison of Nature Inspired Intelligence Optimization Methods in Aerial Spray Deposition Management.” M.S. thesis, University of Georgia, 2002; available at: http://www.cs.uga.edu/~potter/dendrite/wu_lei_200212_ms.pdf; Internet.