

REDUCING INTERACTION FATIGUE IN AN INTERACTIVE EVOLUTIONARY SYSTEM

M.R.N. Shackelford – Advanced Computational Technologies, UK,
D.W Corne – University of Exeter, UK

ABSTRACT

Very large scale scheduling and planning tasks cannot be effectively addressed by fully automated schedule optimisation systems, since many key factors that govern ‘fitness’ in such cases are unformalisable. This raises the question of an interactive (or collaborative) approach, where the fitness is assigned by an expert user. To allow the Human User to specify his unformalisable intuitive fitness score previous research [1, 2] showed that a population of results was the most appropriate technique, suggesting the use of some form of Evolutionary Algorithm – in this case a Genetic Algorithm.

This paper considers the problems of integrating a human user as the fitness function for an Interactive Genetic Algorithm that may be running through 200+ iterations, and creating 10-20 solutions at each iteration. The user is faced with inspecting and scoring 2000 – 4000 individual solutions, as he has to determine and specify the fitness of each solution. This paper uses previously reported developments of an Automated Tester, and extends this to provide an Automated User – which is able to replace the Human User for a number of iterations.

Results show that the Automated User performed to within 5% of the Human User, and significantly reduced the level of interaction required by the user.

1. INTRODUCTION

The development of Interactive Evolutionary Systems (IES) for scheduling involves the integration of a human expert user as all or part of the fitness function, such that the human user can use his intuition, knowledge and experience to influence the fitness values, and thus drive the IES towards his preferred solutions. This paper reports on how we resolved the issues found in testing an IES with some “real” users, who were not impressed with the amount and intensity of interaction required of them.

As part of a project [1] involving a number of organisations that face large-scale planning problems we developed an IES for use by the Programme Managers. The IES generates candidate solutions in the form of GANTT charts (horizontal bar charts with the duration of tasks plotted against a date axis) overlaid with Resource profiles (showing total resource usage against date), which are displayed on the screen with an indication of their calculated fitness

(based on slippage in duration or start date) as shown in the diagram below. The planner supplies a score for each plan by clicking on a simple drop-down menu, which is then used by the IES in selecting the parents of the next generation.

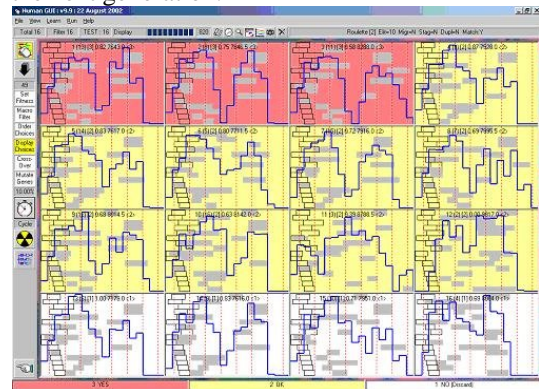


Figure 1.1 Example Screen Shot

The users found that having to interact at every iteration with around 10-20 GANTT charts soon became tiresome and their concentration tended to wander, causing more frequent errors in the selection of fitness scores. Previous work [2] showed that the IES could be objectively tested using a software-based Automated-Tester, which was able to simulate a human user. This functionality was able to be extended to provide a replacement for the human user, by adapting the Automated Tester to become an Automated User.

The Automated Tester is a software module which simulates the behaviour of a Human User by entering either fixed or random selections at each iteration. The Automated User extends this mechanism by learning the actual choices made for each population member, based on matching profiles.

2. REDUCING INTERACTION

Demonstrations of the IES program to a ‘focus group’ of project managers showed that the IES was felt to be extremely powerful, both in enabling the planners to quickly visualise the schedule’s meaning using the Resource profile charts, and also in helping them in influencing the IES to generate selected profile types.

However, as with all focus groups, there were still some negative observations, amongst which the most popular seemed to be:
“Too boring to use”

“Too much interaction required”

“Too much time required”

“Why can’t the system learn what I want it to do?”

The experimental results showed that each run of the IES took between 30 and 40 minutes.

These comments and results suggested a new area for experiment and development:

“Why should the program not be able to learn the style of Resource Profiles that the user prefers, and drive the GA directly?”

This would allow the user to let the program run by itself, and only interact occasionally to check the on-going results, and adjust the learnt profiles if necessary. The question then is – how to get the system to learn the user’s preferences, and whether this can be formally presented to the program.

Considering the feasibility of specifying the user’s preferences as an algorithmic fitness function Banzhaf [3], in generating interactive images, suggests that it is difficult to directly encode the human user’s fitness criteria, as *“many design processes require subjective judgement relying on human intuition...[where] the fitness criterion cannot be formulated explicitly”*. Burke, Elliman & Weare [4], in respect of a timetable scheduling system, note that *“...in reality the timetabler... would be simultaneously trying to balance many different aims and objectives. Some of these may not even be easily expressed in the evaluation function”*.

Ochoa [5], working with L-systems, states that *“the human eye is good at selecting phenotypic effects, and to construct computer programs that directly select phenotypic patterns is a difficult and sophisticated task.”*

Poli & Cagnoni [6], working with a GP-based imaging system, state that *“in many real-world situations it is very difficult, if not impossible, to formulate mathematically a reasonable quality criterion...for example, when subjective or qualitative criteria have to be used”*.

Tuson, Ross & Duncan [7], consider that *“it is very common for the constraints and objectives of the problem to be ill-defined, or even to conflict”*.

The above research suggests that the user would not be able to define his preferences in any formal or algorithmic way, so getting the system to learn directly from the end user is arguably the most appropriate way forward. However, this is not necessarily an easy option either, as we have to cope with the foibles and boredom level of the human user.

Baker & Seltzer [8] confirm the ‘focus group’ findings where they conclude that *“including a human evaluator weakens the GA because the human’s speed and patience restrict the population size and possible number of generations”*.

Banzhaf [3] describes what is considered a potential difficulty, which is that *“due to psychological constraints, humans can normally only select from a small set (order 10) of choices...[and] only a limited number of generations can be inspected by a user before becoming tired”*.

Biles [9] realised this with GenJam and attempted (not entirely to his satisfaction) to remove the human bottleneck by using a library of pre-defined musical phrases with an algorithmic fitness function.

Johanson & Poli [10] had a similar problem with their GP-Music system in that the *“user must listen to and rate each musical sequence in every generation...this may be a tedious process.”* They extended their package with “Auto-Raters” using a Neural Network to learn the user’s preferences. However, although the basic results *“proved somewhat successful, ...the auto rater runs were not able to generate nice sequences with the reliability of human rated runs”*.

However, these ideas do suggest techniques to get around the human user’s boredom with the process, in terms of a learning mechanism that can detect and remember the user’s preferences and then apply them as extensions to the fitness function. The poor results with the auto rater also suggests that human intervention should occur at regular intervals to steer the auto rater back to the “nice” results.

Poli & Cagnoni [6], again, in developing their interactive GP-based system used to enhance Magnetic Resonance brain scans, found that that *“when the user is part of the EA only a limited number of evaluations can be performed, and therefore [interactive evolution] does not scale up well”*. The solution proposed is *“the development of techniques that reduce or eliminate User Interaction, but still follow the criteria applied by the user...by building a user model on-line...and using the model to process a majority of the individuals”*.

We concluded that the user should be able to “train” the system by interacting with the program for a number of iterations, and for the program to learn the kinds of profile that the user considers to be ‘best’. The user should also be able to interrupt the program, review the state of the solutions and make any adjustments necessary, as it is apparent that the ‘learnt’ profiles will be an inexact model of the user’s actual preferences.

3. LEARNING SHAPES

The basic requirement is for the IES to recognise the resource profile ‘shapes’ that the user selects for the different fitness values, and to develop some internal representation of these shapes so that they can be matched to subsequent population generations.

First attempts at getting the system to learn the user's shapes were based on collecting the mean values of the Resource Profile from ten vertical 'slices' through the profile over time. These were successful for the simple profiles (Front loaded, Flat, Rising), but failed when more complex (and realistic) profiles were attempted.

For example, if the user determined that a Front Loaded profile was fittest, then the averaging algorithm could successfully model that shape, and score generated shapes accordingly. However, if the user considered two or more dissimilar shapes to be fit, then the averaging technique could model neither shape, and would work with the average 'in-between' shape, matching none of the user's intended shapes.

The program needed a different and more sophisticated technique to learn from the user. Neural Nets came to mind (thanks to Johanson & Poli [10]), and these are a possible area for future work, although they were felt to be too complex for the basic learning techniques required.

Lee & Street [11] outlined a technique used to learn a user's criteria for selecting shapes - in this case for medical diagnosis. Their technique mapped the shape outline on to a regular grid (in this case a polar chart, as the shapes tend towards circles or ellipses). A series of templates was then built up to handle the fact that the shapes are not all similar. If a new shape matched an existing template, within a threshold, then it was added to that template, otherwise a new template was created. The user interacted with the system by editing the template shape outlines, and also determining whether newly generated templates were in fact unique.

This technique maps well to the issues of learning the user's preference based upon the shapes of the various different resource profiles, and the ideas presented above - mapping shapes to templates, and generating new templates where necessary - was therefore implemented within the experimental IES program, and is described in the following sections. The algorithm is an example of Case Based Learning, in which individual instances of the generated solution profiles are linked to the user's scores.

3.1 Shape Processing Methodology

The IES program was extended to use the multiple-template technique for learning the user's profile shape preferences.

First, a "Learning" mode was implemented, during which the user runs the IES for a number of generations, and scores the displayed individuals using the user fitness values (e.g. 5=Best to 1=Worst) as normal. The system builds up a 'library' of template

shapes based on the resource profiles grouped by the user's score as described in section 11.3.1 below.

Then, when the program runs in Automatic mode without user intervention, at each iteration the individual solutions are compared to the stored templates, and the individual's scores determined from the closest matching templates, based on least absolute difference.

The basic data structure used to store the learnt templates is a number of lists (arrays) of templates, with one list for each possible score that the user can select. For example, when there are 5 possible fitness scores being used (5=Best to 1=Worst) the system will have 5 different lists of learnt templates.

3.2 Learnt Templates

At each iteration, in learning mode, the user sets the fitness scores for each individual solution in the population, and the program then checks the list of templates corresponding to the specified fitness score for each individual. The generated resource profile is compared with all those stored as templates in the list for the specific score. If the new profile matches an existing learnt template to within a threshold, then a Hit Count is incremented to indicate how popular the particular template has been, otherwise a new template is created with an initial Hit Count of 1. The Hit Count is intended to be used as a popularity measure, as the algorithm may implement the removal of least-used templates, either for house-keeping purposes (if the template arrays get too large), or in the situation in which the user makes occasional errors in scoring and the algorithm needs to be able to delete the ones that are referred to least.

3.3 Template Storage

The stored templates are based on a rectangular grid with 10 cells vertically and up to 20 cells horizontally (depending on the time-span of the programme), as shown in Figure 3-1.

In some initial experiments with the program these values were found to be acceptable. If many more than 10 units were used in the vertical scale the system had to store a large number of templates in the learning arrays, and if too few units were used then the system was unable to differentiate between profiles, and so did not build up a very accurate picture of the user's intentions. Again, for the horizontal scale, too few scale divisions meant that the system found it difficult to differentiate between profiles, and too many divisions caused the system to generate a large number of similar profiles, which were time-consuming to evaluate.

Each template is stored as a 1 dimensional array of up to 20 elements, each of which contains the 'y' dimension of the profile grid in which the segment appears. The vertical axis values of a resource profile

are first scaled to the range 0.0 to 10.0 (based on the maximum resource profile value of all individuals in the current population) and then rounded down to the nearest integer – giving possible values {0,1,...,9}.

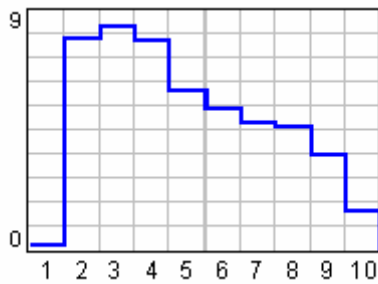


Figure 3-1 : Resource profile diagram

In the example shown, the resource profile is grouped into 10 horizontal cells (along the horizontal time axis) and with values scaled on the vertical ‘y’ axis in the range 0-9, the template array would be:

‘x row’ horizontal	1	2	3	4	5	6	7	8	9	10
Template values:	0	8	9	8	6	5	5	5	3	1

Table 3.1 Profile column values

Rounding the profile vertical scale values to the nearest integer, within the range {0,...,9} means that any two profiles will be compared to an accuracy of 10%.

3.4 Assigning Templates to Scores

When comparing templates, the system only has to check whether the new profile has values that match each of those in the existing templates, after rounding the actual values to the nearest integer. This method allows templates to be matched to profiles with a threshold of error of 10% in the user’s selections.

The system stores not only the template profiles against each fitness score, but also a running total (Hit Count) of the number of times a template matches a new individual, giving a ‘popularity’ factor as well. This Hit Count will be used as a weighting factor during the Auto-User mode in case several profiles appear as templates against different fitness scores, for example if the user selects a particular shaped profile for the wrong fitness score by mistake. The more frequently a profile matches the template of a given fitness score, the more likely that this is the appropriate match.

3.5 Template List Sizing

The size of the template lists was based on initial experiments that showed that between 10 and 15 iterations with a population of size 16 would create enough learned templates. This would mean that around 240 templates would be generated, with many being duplicates, giving a likely maximum of approximately 100 templates. These would also be

spread across 2 or more different user scores, so a reasonable expectation is that a maximum of 50 templates would be required for each score. The size was set to 100 in the expectation that this limit would only be reached if the program logic had an error.

3.6 Using Learnt Templates in Auto-User Mode

When running in auto-user mode the system generates a population of new individuals as normal, and creates an array of profile values for each individual in the same way as the learnt template profiles are calculated. The program takes the absolute value of the difference between the rounded values of each of the profile’s array elements and those of the learnt templates stored for each score. The template with the least difference determines the fitness score allocated to that individual. In the case of two templates, from different fitness scores, having the same calculated difference, then the Hit Count weighting factors are used to select the likeliest template.

3.7 Template Comparison Example

For example, let us take an individual from a population generated at some iteration in auto-user mode. Assume that it has the following profile (before scaling)

Time Period	1	2	3	4	5
Resource Usage	43	127	268	236	372
Time Period	6	7	8	9	10
Resource Usage	217	164	131	101	67

To scale the Resource Usage, we also need to know the maximum resource profile value of all individuals in the current generation, so let us assume that it is 432. The selected individual can then have its usage values scaled into the range 0 to 9, using the formula:

$$\text{Scaled Value} = \text{Round}(10 * \text{Resource Usage} / (\text{Maximum Value} + 1))$$

This would give an array of rounded values for our example individual of:

Scaled Values	0	2	6	5	8	5	3	3	2	1
---------------	---	---	---	---	---	---	---	---	---	---

This can now be compared with the stored template arrays, and we will use the values of Figure 11-1 for this example:

Template values:	0	8	9	8	6	5	5	5	3	1
------------------	---	---	---	---	---	---	---	---	---	---

The absolute (unsigned) difference between the individual’s scaled values and the stored template values are now calculated:

Difference:	0	6	3	3	2	0	2	2	1	0
-------------	---	---	---	---	---	---	---	---	---	---

Giving a total difference of 19. This value can then be compared with the differences calculated for each of the learnt profiles, with the smallest difference indicating the best match and thus selecting the appropriate fitness score for that individual.

4. EXPERIMENTS WITH GA BASED SHAPE LEARNING

A number of tests were run on the new Auto-User IES program to determine whether:

- the GA could learn from the user, and correctly build up the learnt profile templates
- the GA could correctly score each new generation using the learnt templates.
- the GA could keep enough diversity in the population to function effectively.
- the human user could cope with different population sizes when teaching the GA.
- the “Learning” GA could perform as well as the human controlled IES.
- The “Learning” technique could reduce the level of human interaction required.

4.1 Training the GA

The first set of tests were to determine whether the IES program could learn from the user, prior to further tests which would find out whether the IES could use those learnt profiles to run in automatic (Auto-User) mode. These tests checked that the algorithm implemented to scale and store the profiles worked properly.

Each experiment used a sample programme, labeled “TEST”, which consisted of 16 projects, a total of 269 tasks, and a total duration of 9735 days. For comparison with the results from a standard Serial Scheduler, the same programme was run through Microsoft Project, which returned an overall elapsed time of 5305 days.

To experiment with training the IES to learn the shapes preferred by the user, the program was set up to run through a number of iterations with typical starting populations – as used in previous experiments. The user viewed the profiles generated at each iteration, and assigned fitness scores depending on how close they were to the target shapes. The program used each fitness score to build up the arrays of templates as described above.

As the iterations proceeded, the program scored each new generation using the learnt profiles, and so the user was able to observe how effectively the process was working by how accurately the program managed to score the individuals of each new generation. Each experiment was halted when the user was satisfied that the latest generation’s individuals were all correctly scored.

Figure 4-1 shows the results of a user attempting to teach the GA each of the main Resource profiles, with the profiles reasonably reflecting the target shapes.

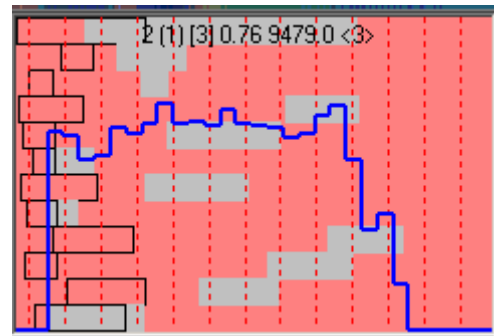


Figure 4-1.1 Flat Profile

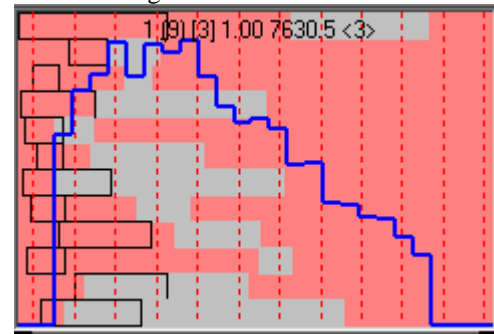


Figure 4-1.2 Front Loaded Profile

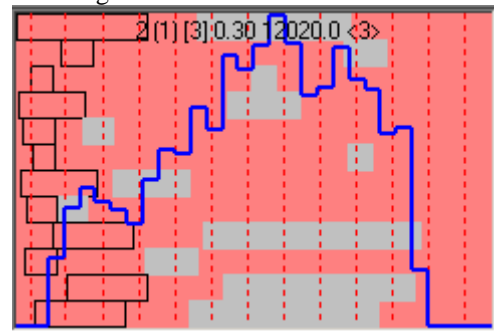


Figure 4-1.3 Back Loaded Profile

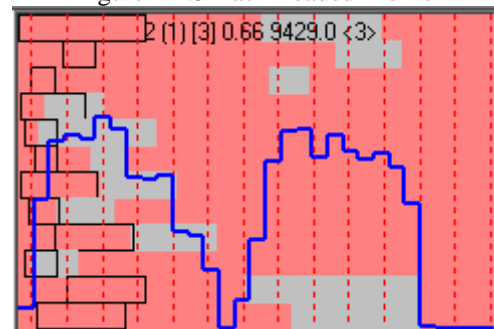


Figure 4-1.4 Valley Profile

Tests were carried out with a range of different population sizes, from 9 to 25, and 8 trials were performed for each combination of target shape and population size. The mean number of iterations to reach recognisable shapes are shown in Table 4-1:

Size	Flat Mean	Flat SD	Front Mean	Front SD
9	26	6.9	16	5.2
12	23	6.5	14	5.9
16	10	3.1	10	3.8
20	14	4.5	13	4.7
25	30+		30+	

Size	Back Mean	Back SD	Valley Mean	Valley SD
9	28	6.8	20	5.1
12	26	6.1	19	3.4
16	20	4.1	14	4.3
20	24	4.8	17	5.1
25	30+		30+	

Table 4-1: Population size tests

It was noted that the smallest populations (9 and 12) were too small to generate enough diversity, whilst the larger populations (25+) were unstable, generating too much diversity, and also prone to user input error with so many individuals to select. A population of 16 was used for the remainder of the testing, as this can be seen to have the best mean results and the lowest standard deviation.

The experiments suggest that the human user only needed between 5 and 10 minutes to “train” the IES, which was seen as an acceptable time frame by the users. These times compare well with the 30-40 minutes required for a fully interactive IES session.

Due to the date-based nature of the internal fitness values, which tends to favour the least slipped individuals during the selection and recombination process, the GA found the Front loaded profile to be the easiest to generate and learn. The Flat and Valley profiles were the next easiest to generate. The Back loaded profile was the hardest, as this goes against the tendency of the internal fitness values.

An important feature to note is that the user was able to generate preferred profiles by selecting profiles that were partially correct (i.e. flat at one end or the other) and then relying on the cross-over mechanisms of the GA to breed shapes closer and closer to the desired profile.

4.2 Using the “Learnt” shapes - the “Auto User”.

The “Auto-User” is an extension to the basic IES, allowing the human user to “take a coffee-break” and leave the GA running for a specified number of iterations under automatic control. The human user selects a set of ‘learnt’ templates from a saved file, and the “Auto User” uses these templates to determine the user’s likely preference when selecting the fitness of individuals.

At any point the human user can “Break-In” to the automatic cycle and make any adjustments to the profiles or selected individuals, and then let the “Auto-User” continue.

The Internal Fitness value based on Date Slippage or Duration Extension is still used, to select the best individuals from the population, but these values are grouped by the overall Auto-User defined fitness

values so that individuals matching the Best profile are more likely to be selected for cross-over, but within the Best profiles, selection is based on the standard internal fitness value calculated from the scheduled date and duration slippages.

4.2.1 Valley shaped templates (“Twin Peaks”)

The first target shape defined by the user was a Valley shape, and 10 iterations were used to train the program in learning the profile. Learning mode was switched off, leaving the system to run in automatic mode, with no user interaction. The run was halted when the solutions had settled on a shape, with the results for the set of 8 trials are summarised in Table 4-2 below:

Trial	[1]	[2]	[3]	[4]
Iterations	49	46	34	49
Fitness	7463	6677	6785	6764
Trial	[5]	[6]	[7]	[8]
Iterations	50	48	38	50
Fitness	8666	7981	7188	6948

Table 4-2 : Valley Auto-User test results

The Mean Fitness = 7309, with a standard deviation of 700. The Serial Scheduler derives a fitness of 5305 for the “TEST” programme, but this is a front loaded profile, determined from the earliest possible schedule dates for the given sequence of projects within the programme.

As the tests above are trying to deliver a Valley shaped profile, it is expected that the overall fitness would be lower than the benchmark, as several projects will have been slipped to allow for the valley, where fewer resources are required to be assigned. Even so the best solution (6677) is only 25% worse than the benchmark.

4.2.2 Back loaded Profile tests

The next set of trials ran the Auto-User with a different set of learnt profiles, aiming to create a “Back-Loaded” profile, where the resource usage starts low and builds up over time.

This is intended to simulate a programme start-up scenario, in which a new programme has to wait for resources to become available from other programmes which are running down. The Programme Manager will have some idea of the overall timescale over which he wishes to build up to full resource capacity.

Table 4-3 summarises the results of 8 trials:

Trial	1	2	3	4
Iterations	29	42	48	44
Fitness	8722	9399	9788	9974
Trial	5	6	7	8
Iterations	50	48	38	50
Fitness	11091	9457	8676	10189

Table 4-3 : Back loaded Auto User test results

The Mean Fitness = 9662, with a standard deviation of 792. Again we can see that the mean results are worse than the standard Serial Scheduler, but this is again because the shape required means that the projects must have large delays to allow the back-loaded profile to be generated.

4.2.3 Tests with Flat Profiles

The final set of tests used a learnt “Flat” profile. The Flat loaded profiles were not smooth, as the system has to generate feasible solutions, which restricts the actual layout of the plans and tasks. Table 4-4 summarises the results of 8 trials:

Trial	1	2	3	4
Iterations	32	36	49	48
Fitness	8359	8253	8306	7907
Trial	5	6	7	8
Iterations	44	50	38	49
Fitness	8245	8931	8714	9780

Table 4-4 : Flat profile Auto User test results

The Mean Fitness = 8561 with a standard deviation of 582.

As expected the mean results are worse than the standard Serial Scheduler, as the Flat profile requirement needs to spread the projects out over time.

4.2.4 Comparison with Interactive User results

The system was then tested to determine how well the Auto User results compare with the human user using the Interactive GA. A series of tests was run using the same programme (TEST) with a human user attempting to generate the same target profiles as defined for the Auto User tests above.

The following table summaries the results of these tests and shows the fitness of the best solutions that were able to be generated in no more than 50 iterations.

The tests stopped when the human user felt that the solutions had stagnated, or when the user subjectively considered the solutions to be acceptable.

Valley	1	2	3	4		
Fitness	7009	6935	7548	6868		
Valley	5	6	7	8	Mean	SD
Fitness	6912	7720	7135	7258	7173	314

Back	1	2	3	4		
Fitness	8906	9197	8371	7990		
Back	5	6	7	8	Mean	SD
Fitness	9748	10041	9531	9068	9106	686

Flat	1	2	3	4		
Fitness	8666	8445	7703	8417		
Flat	5	6	7	8	Mean	SD
Fitness	8552	8653	7855	8119	8301	367

Table 4-5 : Interactive User Results

We can now compare the Interactive User controlled GA with the IES in Auto User mode:

	Auto User		Human User		Auto vs Human
Profile	Mean	SD	Mean	SD	Mean
Valley	7309	700	7173	314	1.8%
Back	9662	792	9106	686	5.7%
Flat	8561	582	8301	367	3.0%

Table 4-6 : Auto User vs. Human User

In table 4-6, the column labelled “Comparison” suggests that the Human User produces slightly better results on average than the Auto User, and comparison of the standard deviations shows that the human user produces more consistent results.

5. CONCLUSIONS

This paper has demonstrated that an IES with a template-based Automated User can reduce the level of human interaction required, without significant loss of accuracy. The tests show that the program quickly (within 10 to 20 iterations) learns from the user, and can build up templates that allow enough accuracy to drive the GA fitness.

The experiments suggested that the smallest populations (9 and 12) were too small to generate enough diversity, whilst the larger populations (25+) were unstable, generating too much diversity, and also prone to user input error with so many individuals to select. However, a population of 16 provided an acceptable level of diversity. The population size that was most appropriate for the experiments in this paper (16), is also the size that was determined as the most successful for user interaction [2].

The final experiments suggest that the Auto User and the Human User, running tests with the same conditions, produce statistically similar results (to within 5% on average). This means that the Auto User mode can be used without sacrificing the acceptability of the solutions.

The learning IES only needed 5-10 minutes of user interaction compared to the 30-40 minutes of a fully interactive session. Also, with the ability to store learnt templates, for subsequent runs the user would need minimal interaction, other than selecting a set of appropriate templates.

This technique of learning from a user’s preference for shapes is likely to be applicable to a wide range of Interactive Evolutionary Computation algorithms, providing they have some form of graphical representation of the solutions that can be graded by a human user. This might include Interactive Art, Photo-Fits, Engineering surface design, car body design etc.

REFERENCES

1. Shackelford M R N and Corne D W (2001), Collaborative Evolutionary Multi-Project Resource Scheduling, in Proceedings of the 2001 Congress on Evolutionary Computation, IEEE Press, pp. 1131-1138
2. Shackelford M R N & Corne D W (2004), A Technique for the Evaluation of Interactive Evolutionary Systems, in Proceedings of the Sixth International Conference on Adaptive Computing in Design and Manufacture (ACDM 2004).
3. Banzhaf W (1997), Interactive Evolution, Handbook of Evolutionary Computing, C2.10
4. Burke E, Elliman D and Weare R (1994), A Genetic Algorithm Based University Timetabling System, Proc. 2nd East-West Int'l Conf on Computer Tech. In Education
5. Ochoa G (1997), On Genetic Algorithms and Lindenmayer Systems, In Parallel Problem Solving from Nature (PPSN V), pp 335-344, Springer
6. Poli R and Cagnoni S (1997), Genetic Programming with User-Driven Selection: Experiments on the Evolution of Algorithms for Image Enhancement, in Proceedings of the Second Annual Conference on Genetic Programming, pp.269-277, July'97
7. Tuson A, Ross P and Duncan T (1998b), On Interactive Neighbourhood Search Schedulers, 16th Workshop of the UK Planning and Scheduling Special Interest Group, ed M.Fox
8. Baker E and Seltzer M (1994), Evolving Line Drawings, Graphics Interface '94 Proceedings
9. Biles J A (2001), Autonomous GenJam: Eliminating the Fitness Bottleneck by Eliminating Fitness, Proc. GECCO-2001 Workshop on Non-Routine Design with Evolutionary Systems
10. Johanson B and Poli R (1998), GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters. In Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22-25 98
11. Lee K-M and Street W N (2000), Learning Shapes for Automatic Image Selection, INFORMS & KORMS 1468 Seoul