FURTHER PRELIMINARY EXPERIMENTS WITH IIDLE

Technical Report 11-01

Jason Brownlee
jbrownlee@ict.swin.edu.au

PhD Candidate
Master of Information Technology, Swinburne University of Technology, 2004
Bachelor of Applied Science, Computing, Swinburne University of Technology, 2002

Centre for Intelligent Systems and Complex Processes
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Victoria, Australia

October 2005

Table of Contents

Figures

Tables

Equations

1 Introduction

The immunological inspired distributed learning environment (IIDLE) [6] is a novel learning system that belongs to the field of artificial immune systems. It is inspired by specific characteristics of the acquired immune system such as a spatially distributed population of discrete information packets (units) and internal regulatory processes. IIDLE is a relatively new platform and as such has been subject to a number of rounds exploratory experimentation to preliminary evaluate the suitability of the technique in terms of configuration and application [5,7]. This work provides an additional series of preliminary and exploratory experiments along the same lines. Specifically, this work investigates three potential fields of study that have been raised or rudimentarily address in previous work. They are as follows:

1. The behaviour of IIDLE whilst undergoing dynamic structural changes (adding and removing localities) – Section 2
2. The performance of IIDLE with three embedded and competing proliferation strategies (GA, PSO, and ACO) – Section 3
3. The potential for IIDLE on function approximation and classification problems where there is not a one-to-one mapping between units and solutions – Section 4

Finally, Section 5 provides concluding remarks and a discussion of future research, specifically ideas on how the work from the three experiments can be extended.

2 Experiment 1 – Dynamic Structural Changes

As was described in the introduction to IIDLE [6], the localities of IIDLE's data structure are loosely-coupled, requiring only connectivity to other localities to facilitate the movement of units. A benefit of the replicated discrete locality design that was identified in the introduction to IIDLE is that it permits the dynamic addition and removal of localities with an expected minimal impact on the efficacy of the system. This series of experiments preliminary tests this assumption.

The system was configured for a single instance of a common combinatorial optimisation problem – the Berlin-52 Travelling Salesman Problem (TSP), which has an optimal tour length of 7544.36. The search technique used for these experiments was discrete history ant systems (DHAS). A simple yet dramatic structure reconfiguration approach was evaluated where at a random point over the course of 200 iterations; a set number of empty localities were added or removed from the IIDLE data structure. A Gaussian probability distribution was used to select the random point of structure modification with a mean of 100 and a standard deviation of 20. The measure of interest was the minimisation of the final tour length scoring.

Three different configurations were tested for comparison. The first was a simple static IIDLE structure with a set number of localities that remained constant over the duration of the run. Five different structure sizes were tested from 50 to 10, in 10 locality decrements. The second configuration saw the addition of localities during the run. The system was initialised with 10 to 40 localities increasing in 10 locality increments, and at a randomly chosen iteration during the run, the number of localities was topped up to 50. The final configuration conversely always initialised the system with 50 localities and removed a set number of localities from 40 to 10 in 10 locality decrements at a selected point over the course of a run.

The following table provides a review of the three configurations and the changes made to the IIDLE data structure during the experiment concerning the number of localities.

#	Static	Added	Removed
1	50	10→50	50→40
2	40	20→50	50→30
3	30	30→50	50→20
4	20	40→50	50→10
5	10	-	-

Table 1 - Experimental configurations for the number of localities

2.1 Observations and Discussion

It should be noted that when localities are added to the structure they are empty of units and appended to the locality network topology at a single location, and the entire structure is rewired into one large loop. In the case of removing localities, a locality is selected at (uniformly) random and deleted, which includes all units in the locality at the time of deletion. After removal, the structure is again rewired into a large loop network topology.

The structure of the system is adjusted which has a direct impact on the probabilities of localities (and thus the localities units) being stimulated. The amplitude (number of stimulations) per iteration was kept constant, thus fewer localities (the removal) increases the probability of locality stimulation and the addition of localities results in the opposite effect. It is also interesting to note that the decay process used was configured to maintain total system energy at equilibrium. Thus the removal of localities and their unit's results in a short dip in overall unit population size, which is quickly returned to a stable point, whereas the addition of localities did not affect the population size, instead, causing the unit population to be more sparsely spatially distributed.

The following graph provides a summary of results for all three data structure configurations.

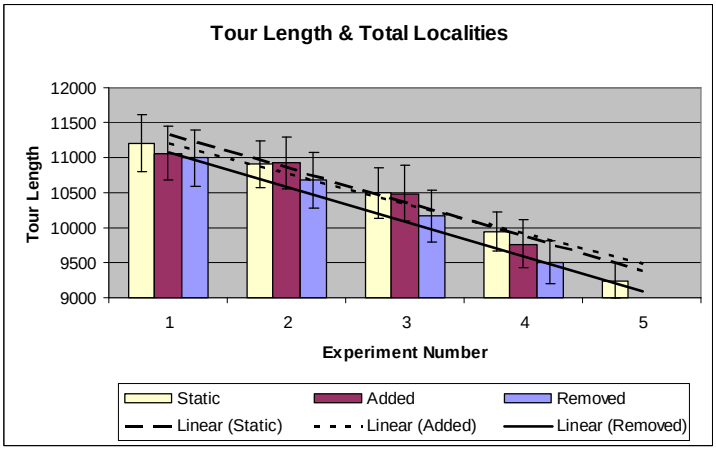

Figure 1 - Summary of results for dynamic structural changes to IIDLE

The graph shows the results of the static configurations in decreasing order of the number of localities from left to right, which clearly shows that the less localities (higher the probability) of being stimulated, improves the result (lower tour length) significantly (nearly 2000 points different between 50 localities and 10 localities). The stimulation amplitude was five for each iteration, meaning that with 50 locality interfaces the probability was 10%, whereas with 10 localities the probability of stimulation and the resulting expansion was 50%.

The next configuration was the addition of localities, and the results are ordered by the number of localities added from 10 to 40, (which tops the data structure up to 50 during the

run). The results show that as more localities are added to a smaller initial locality configuration the results improve. The reason for this affect may be that by starting with fewer resources the system is permitted to provide a lots of attention to the search early (increased competition between units), which is then relaxed at a random point over the search. This is the opposite of a conventional wisdom for search strategies that seek specialisation (increased competition) at the end of the search after a courser survey of the search space.

The final set of results demonstrated in the graph is for the removal of locality resources, ordered by the number of resources removed from 10 to 40. As in the case of the previous two sets of results, a linear increase in result quality is observed, although in this case, the observation occurs as more localities are removed. Unlike the addition of resources, all removal experiments started with a locality size of 50 providing opportunity for the system to broadly sample the problem space. The removal of resources provided a focus for the search or a trigger for increased competition resulting in results that performed better than either of the other two approaches.

The major outcome from these results concerns the role of localities as an interface to the search process, in addition to the expected demonstration of the systems robustness (which was also demonstrated). The results showed that the DHAS proliferation approach achieved an improved (final) result when the history in the system (units) are consolidated into a fewer number of localities, specifically at a point after the search has had a chance for a course exploration of the problem space. IIDLE demonstrated that rather than being efficaciously unaffected with regard to dynamic structural changes, that specific structural changes can improve the quality of result compared to the static configuration counterparts. This is likely dependent upon the specific search strategy (ACO that pays a lot of attention to search history), configuration, and problem domain being addressed (search and optimisation).

3 Experiment 2 – PSO and Parallel Hybrid Search

An outcome of preliminary experiments in [7] indicated that a potentially viable area of research with IIDLE is the application of multiple search strategies (proliferation strategies) in a parallel hybrid search configuration. This section provides initial experimentation of three embedded, well-known, and widely used search and optimisation strategies in a parallel hybrid search configuration in IIDLE. Specifically a genetic algorithm (GA), an ant colony optimisation algorithm (ACO), and a particle swarm optimisation algorithm (PSO) applied to a function optimisation problem. (Schwefels function in five dimensions). The optimal solution to this function is -2094.914.

The details as to how to embed a genetic algorithm and ant colony optimisation algorithm (discrete history ant systems) in IIDLE has previously been discussed [5], and preliminary demonstrated [7]. This section provides a summary of the modifications to canonical particle swarm optimisation (PSO) that were made to embed the search strategy into IIDLE as a proliferation strategy. This implementation summary if followed with the results and discussion of preliminary experiments embedding each of the selected search strategies in IIDLE both independently and hybridised in parallel configuration.

3.1 Particle Swarm Optimisation

Particle swarm optimisation (PSO) [2,3] is a function optimisation technique inspired by the group feeding and communication behaviour of schools of fish, flocks of birds, and plagues of insects (so called swarm intelligence [4]). The approach typically models a problem using a number of discrete particles and a multidimensional Euclidean space. The essence of PSO is a collection of particles that "*fly*" through a search space like a flock of birds, settling in on areas of interest in the context of the problem domain (such as function extrema).

The standard PSO algorithm was modified to facilitate its embedding in IIDLE. An evolutionary computation-like approach was taken, where only a select few units (particles) are updated, and rather than adjusting the actual selected units, progeny are created and the calculated changes applied. This is different from conventional PSO where a set number of particles are maintained and updated for each algorithm iteration. A progeny unit is created as a clone of a selected parent, and the cloning includes the parents current velocity, position, and personal best (remembered best position every visited) information. Thus, the personal best position in the search space is maintained through a hereditary history for a line of units that have been selected and expanded upon. Each locality is considered a neighbourhood of coincidence, and thus the global best calculated for the locality may be somewhat like a local best in a more conventional spatial neighbourhood. The following provides a pseudo-code summary of the adjusted particle swarm optimisation algorithm.

```
1. Evaluate unit positions
2. Update personal best positions
3. Perform greedy selection
4. Locate global best (gBest)
5. For each selected unit
      a. For the number of progeny
            i.  Clone parent unit
           ii.  Update progeny velocity
          iii.  Update progeny position
      b. End
6. End
7. Add progeny to locality
```

Figure 2 - Pseudo-code summary of the adjusted PSO proliferation strategy

The varied algorithm requires the specification of a unit selection strategy and the number of parental units (particles) to select. A greedy selection approach was used to drive competition and direct the search. The algorithm also requires the specification of the number of progeny to create for each selected parent. The following specifies the equations used to update progeny velocity and positional vectors.

$$\vec{v}_{t+1} = (m \cdot \vec{v}_t) + (1-m) \cdot \left(\left(c1 \cdot r1 \cdot \overrightarrow{(pbest_t - pos_t)} \right) + \left(c2 \cdot r2 \cdot \overrightarrow{(gbest_t - pos_t)} \right) \right)$$

Equation 1 - velocity update for a progeny unit under PSO proliforation

$$pos_{t+1} = pos_t + v_{t+1}$$

Equation 2 - position update for progeny under PSO proliferation

Parameter	Description
vmax	The maximum velocity a particle can achieve
c1	Personal best coefficient, typically set to 2
c2	Global best coefficient, typically set to 2
m	Momentum coefficient to determine the amount of the previous velocity and the new velocity to update the particle with
r1, r2	A random number in the range [0,1]
pos	The particles position in search space
v	The particles velocity
pBest	The particles personal best position in the search space. This is the best quality position encountered in a particles history

	(hereditary history)
gBest	The global best particle in the stimulated locality, specifically the globally best personal best position of all particles in the current locality.

Table 2 - Parameters used in updating progeny velocity and positional vectors under a PSO proliferation strategy

A maximum velocity is specified that bounds to particle to not move more than half the search space in a single update. Upon collision with the boundary of the search space (on bounded problem domains), the velocity, and position is reflected and no energy is subtracted (elastic collision).

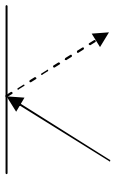

Figure 3 - Shows elastic collisions of particles with the search space boundary

3.2 Parallel Hybrid Search

This section provides baseline optimisation results for the three selected search strategies implemented in IIDLE as proliferation strategies, as well as a partitioned and mixed parallel hybrid search configurations. The run configuration information is provided in Appendix A for reproducibility. It should be noted that a smaller than usual number of localities was used (12) given the results from Section 2, and the amplitude of stimulations by each strategy was adjusted to remain consistent across the entire system over all tests. The system configuration was not tuned for performance, the results are provided as a preliminary and proof of concept performance benchmarks. The following figure provides a summary of results.

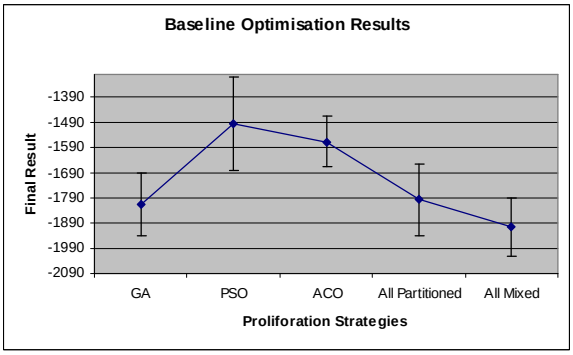

Figure 4 - Summary of results for the selected three proliforation strategies across five different configurations

The results show that the genetic algorithm proliferation strategy out performed PSO and ACO, although the overall efficacy leader was the configuration that used all three strategies across the entire IIDLE data structure. This is an interesting result as it was expected that PSO would outperform the GA on this problem, although with tuning of the algorithm this there is little doubt that this expected result will be achieved. It is speculated that the modification made to the PSO algorithm to embed it into IIDLE may be the cause of its less than satisfactory performance. This tuning is left as an exercise for future research.

It was expected that combining the three strategies together would provide superior results and as shown, this was the primary outcome of this round of experimentation. The results are promising for the argument for IIDLE as a hybrid parallel search platform, and are an vast

improvement over previous results with two complementing rather than competing search strategies [7]. This indicates the potential benefits of parallel hybrid search highlights the need for further investigation. The final interesting point to stress is the application of an ACO algorithm conventionally used for combinatorial optimisation problems, applied to a continuous function optimisation problem. Although the results of the ACO algorithm used in this way were less than impressive, it further demonstrates the power of the discrete history ant systems technique which was first introduced [5] embedded in IIDLE, and later demonstrated as a standalone technique [1].

4 Experiment 3 – LVQ and a Classification Problem

To date, the vast majority of experimentation with IIDLE has been on search problem domains, specifically function optimisation and combinatorial optimisation problem domains. One of the visions that inspired the initial development of the IIDLE learning platform was the application of artificial immune system to novelty detection problem domains. More generally, these domains can be considered classification or function approximation. In the spirit of the original vision of IIDLE, this section provides a specification for embedding a widely used and well-known classification algorithm in IIDLE called Learning Vector Quantisation (LVQ). This section then goes on to provide an evaluation of a proof-of-concept implementation of the algorithm in IIDLE on the 'Iris plants' classification dataset.

4.1 Learning Vector Quantisation

Learning Vector Quantisation is a classification algorithm that maintains a collection of exemplars (also called prototype vectors or codebook vectors) in the input space, where each exemplar has a nominal or continuous output value [8]. LVQ is not limited to classification problem domains (although this is where the technique is predominantly used), the principle can be used in regression problems and more broadly as a function approximation technique. The essence of LVQ is that the collection of exemplars are exposed to input patterns one at a time, a best matching unit in the collection is selected and moved closer or further away to the input pattern depending on whether the exemplar is of the same or differing class as the input pattern.

Each exemplar (also called prototype or codebook vector) is represented by a unit, which has an associated vector of continuous values that matches the dimensionality of the input patterns, as well as an associated classification output. Like PSO, the LVQ algorithm was modified to create progeny units as opposed to modifying a static population of units as specified by the algorithm. Progeny units are duplicates of the selected parent units (best matching units or BMU's) which then have their vector component adjusted either closer or further away to an associated input pattern. The classification label (class) associated with a unit and all of its progeny is immutable. The expected interesting side effect of this is that the input patterns the system is exposed to will define the class mixture in the overall population of units. This expectation was not tested, although remains an interesting area for further investigation.

The following provides a pseudo code summary of the modified learning vector quantisation algorithm embedded in IIDLE for classification. The summary is provided from an expansion point of view given the specialisation of the algorithm (designed primarily for classification rather than general optimisation), for the sake of clarity for reproducibility.

Stimulation

```
1. Draw n input patterns from the training set
2. Select a locality
3. For each selected pattern
      a. Locate best matching unit in locality
      b. Mark unit as BMU
      c. Store input pattern in BMU
4. End
```

Figure 5 - Summary of the stimulation procedure for LVQ-based stimulation

The interesting point to note about stimulation is that a single unit can be selected as a best matching unit for more than one input pattern. This is obvious in the canonical LVQ algorithm, though must be explicitly facilitated when embedded in IIDLE, given that selection of BMU's (more specifically the exposure of the system to input patterns) occurs in batch, prior to the learning (creation and adaptation of progeny). The next step of expansion after stimulation is selection, which in this case is nothing more than the gathering of all units in the locality that have been marked as best matching units.

Proliferation

```
1. For each selected unit
      a. For each input pattern unit was marked BMU
            i.  Create progeny unit (clone)
           ii.  If classification is match
                   1. adjust progeny closer to pattern
          iii.  Else
                   1. adjust progeny away from pattern
      b. End
2. End
3. Add progeny to locality
```

Figure 6 - Summary of the proliferation procedure for LVQ-based proliferation

$$\vec{v} = \vec{v} + \left(lrate \cdot \left(\vec{p} - \vec{v} \right) \right)$$

Equation 3 - Unit (exemplar) update procedure in response to an input pattern of the same class

A unit's internal vector is represented as *v*, and the input pattern is represented as *p*. The learning rate (*lrate*) used to adjust progeny is fixed for the duration of a run. An interesting point to note about the LVQ proliferation strategy is that the number of progeny is defined by the number of patterns the locality is exposed. This simplified approach may result in some redundancy, which may or may not be a useful feature in IIDLE's distributed population structure.

4.2 Classification

Classification in IIDLE is distinctly different from representing optimisation problems in the platform. Rather than there being a one-to-one mapping of units to candidate solutions as in optimisation, classification permits the system in its entirety (all units across all localities) to represent a single solution that is manipulated to achieve a desirable result. The manner in which the LVQ algorithm was used in IIDLE differs from the conventional usage. Specifically, when approaching a static classification problem with LVQ, the procedure is to first construct a model from the training data, then evaluate the validity that model using test data. Such an application of IIDLE is not suited, and in fact would be considered excessive. The interest here is IIDLE's capability to learn to distinguish classes with a distributed representation with all the benefits that the framework provides. As such, IIDLE is likely

suited to classification tasks in environments in which there is expected component failure and unknown load requirements.

The experimental setup was configured to meet these proposed scenarios. Both the training set and testing set were exposed to the system at the same time, although obviously in the case of the testing set, the system was not made aware of the input patterns class and thus was not able to learn from the pattern set (runtime evaluation purposes only). The system was probabilistically exposed to all training patterns and probabilistically evaluated with all test patterns each algorithm iteration, and the system was executed for 100 iterations. A number of runs were executed and the system was observed.

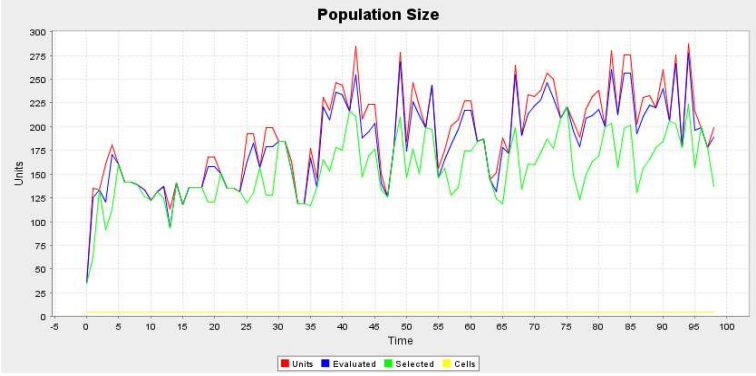

Figure 7 - Snapshot of total unit population size from a single run, showing 100 algorithm iterations

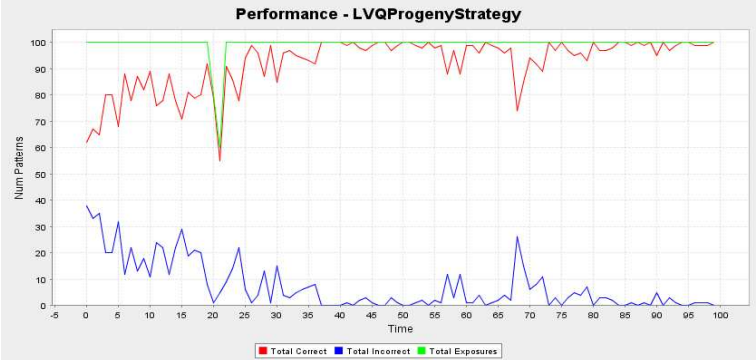

Figure 8 - Training performance with regard to the total correct and incorrect BMU's over 100 iterations, taken from a single run

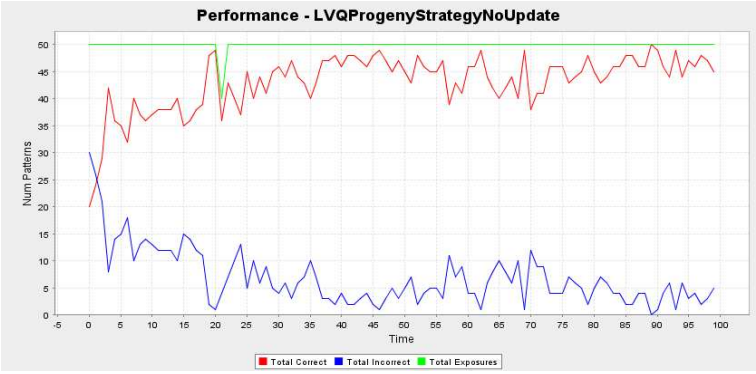

Figure 9 - Testing performance with regard to the total correct and incorrect BMU's over 100 iterations, taken from a single run

The first interesting observation concerns the total number of units maintained by the system. A regulator decay strategy was used that sought to maintain total energy at 20 points, although the running of the algorithm revealed that the total number of units maintained by the system over 100 iterations appeared to stabilise between 200 and 300 (approximately double the number of training and test patterns combined). A possible explanation for this behaviour is that the system was exposed to 100 training patterns per iteration, thus resulting in the creation of 100 progeny each iteration. This creation of a large number of units each iteration was managed by the decay process by keeping each unit in the system at a low level of energy. That is a large population of low energy units as opposed to a smaller system population of high-energy units, an interesting result demonstrating the self-regulating nature of the system.

The large number of system units appears inefficient when considering there are nearly double the units than there are patterns in the dataset. It is important to note that the system is a collection of loosely coupled populations, and in the configuration used for the experiment had a probability of being stimulated 2.5 times per iteration. That is a probability of each locality being exposed to 25% - actually 25 patterns or one quarter of the training dataset (100 of the 150 patterns) per iteration. This is given that the stimulation amplitude was 10 (10 stimulations of the system per iteration), with 10 patterns per stimulation, (10 patterns exposed to the locality for each single stimulation).

The final observation of interest is that the system was successful in its role learning the classification between the three iris plant types. Both on the training and testing datasets good results (>90%) of patterns were classified correctly after approximately 25 iterations, although it should be noted with fluctuations. It is proposed that the observed fluctuations in classification accuracy were due to the harsh decay (removal of units) imposed by the specific configuration chosen. This case study clearly highlights the trade-off between information redundancy (robustness) against the increased resources (unit population sizes) to support such redundancy. More detailed experimentation is required to better gauge the efficacy and efficiency concerns of the IIDLE system on function approximation and classification problem domains.

5 Conclusions and Further Research

This work covered preliminary experimentation for three interesting areas of research for the IIDLE. Although the experiments were simplistic and the results less than conclusive, the experimentation provided at least a proof of concept for dynamic structural changes, parallel hybrid search with three canonical search algorithms and applicability of the IIDLE platform to classification and function approximation problems.

The following lists the major outcomes and points of interested from the experimentation reported in this work.

1. The robustness expected by the distributed nature and decentralised control of the system was clearly shown by the continued functioning of the system entire localities of units were removed from the system, or the locality capacity of the system was increased. The trade-off for this redundancy was highlighted in the following two points.
    a. The local knowledge required in each locality as to how to reconnect to the broader system after a change.
    b. The decrease in efficiency and ultimately efficacy of the search (at least in the case of discrete history ant system on a TSP)
2. Consolidation of history increased the efficiency of the search and in turn the efficacy of the search. This was highlighted in the above-mentioned trade-off, though also is a critical point when considering the suitability of the IIDLE platform to

problem domains. This is a **critical outcome** from this work. The characteristics of the IIDLE platform, specifically in this case distributed, decentralised, redundancy and robustness must be match up to requirements of the selected problem domain. IIDLE is **not suited** to conventional static optimisation problems in its current form as demonstrated by the results of experiment one and three.

3. Parallel hybrid algorithms (mixed PSO, ACO, GA) provided better results over any of the three algorithms alone – an outcome required to demonstrate the viability of competitive parallel hybrid search. IIDLE facilitates this approach, though is not required for the desired effect. It is suggested that a modified version of IIDLE be developed whose architecture is simplified to that of conventional island-based search to demonstrate this area of novel research. As mentioned in point two, IIDLE is not suited to conventional static optimisation in its current form, and this is precisely the type of problems that parallel hybrid search should be demonstrated upon (given the selected algorithms).
4. IIDLE can be configured for classification problems, which is promising for two reasons. The first is the similarity of classification problems to function approximation, novelty detection and regression problems. The second is that this opens yet another interesting and relevant (in the field of AIS) area of research for the IIDLE platform. It also demonstrates that IIDLE may yet meet its original vision of a distributed, decentralised, and self regulating novelty detection platform.

The work on IIDLE continues.

6 Bibliography

[1] Daniel Angus and Jason Brownlee, " Niching Ant Systems," (unpublished), 2005.

[2] J. Kennedy, "Small worlds and mega-minds - Effects of neighborhood topology on particle swarm performance," *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 99.),* Washington, DC, USA, pp. 1931-1938, 1999.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of the IEEE International Conference on Neural Networks,* Perth, WA, Australia, pp. 1942-1948, 1995.

[4] James Kennedy and Russell Eberhart. *Swarm Intelligence*, USA: Morgan Kaufmann Publishers, 2001.

[5] Jason Brownlee, "Implementation Specification for IIDLE," Centre for Intelligent Systems and Complex Processes (CISCP), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report ID: 10-01, Sep 2005.

[6] Jason Brownlee, "Introduction to IIDLE - The Immunological Inspired Distributed Learning Environment," Centre for Intelligent Systems and Complex Processes (CISCP), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report ID: 8-01, Sep 2005.

[7] Jason Brownlee, "Preliminary Experiments with IIDLE ," Centre for Intelligent Systems and Complex Processes (CISCP), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report ID: 9-01, Sep 2005.

[8] Teuvo Kohonen. *Self-Organizing Maps*, Berlin Heidelberg: Springer-Verlag, 2001.

7 Appendix A – Experimental Configuration

7.1 Experiment 1 Configuration

Variable	**Configuration**
Total Runs	100
Movement amplitude	1
Movement probability	1
Decay amplitude	1
Decay mode	Regulatory homeostasis (energy)
Ideal energy	200
Total cells	50
Initial units	Random [1, 10]
Initial energy	Random [0, 1]
Stimulation amplitude	5 (1/10)
Stimulation type	Tour length
Stimulation partitions	one
Selection	All
Proliferation	Discrete history ACO (pheromone exponent 1.0, heuristic exponent 1.5, multiplication, progeny 3)
Stop Condition	Iterations (200 steps)
Triggered Locality Change	Mean 100, stdev 20

7.2 Experiment 2 Configuration

Variable	**Configuration**
Total Runs	100
Movement amplitude	1
Movement probability	1
Decay amplitude	1
Decay mode	Regulatory homeostasis (energy)
Ideal energy	200
Total cells	12
Initial units	Random [1, 10]
Initial energy	Random [0, 1]
Stimulation amplitude	6, (2 when partitioned)
Stimulation type	Function evaluation, Schwefels, 5 dimension, 32 bits per dimension
Stimulation partitions	One, three
Selection	GA=Tournament (size 3, total 4) ACO=Tournament (size 3, total 4) PSO=Tournament (size 3, total 4)
Proliferation	GA (cross 0.95, mut 0.015625, total 4) ACO (history 1.0, heuristic 1.5, multi., progeny 3) PSO (vmax=500, c1,c2=2, m=0.5, total 4)
Stop Condition	Iterations (300 steps)

7.3 Experiment 3 Configuration

Variable	**Configuration**
Total Runs	10
Movement amplitude	1

Movement probability	1
Decay amplitude	1
Decay mode	Regulatory homeostasis (energy)
Ideal energy	200
Total cells	12
Initial units	Random [1, 10]
Initial energy	Random [0, 1]
Stimulation amplitude	10, 5
Stimulation type	Classification of Iris plant data, 150 patterns split 66% (100) training, 34% (50) testing,
Stimulation partitions	1
Selection	BMU based
Proliferation	LVQ 10 patterns per exposure
Stop Condition	Iterations (100 steps)