IMPLEMENTATION SPECIFICATION FOR IIDLE

Technical Report 10-01

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
Melbourne, Victoria, Australia

September 2005

Abstract

The mammalian acquired immune system is a robust and powerful information processing system that demonstrates features such as decentralised control, parallel processing, adaptation, and learning through experience. Artificial Immune Systems (AIS) are a class of machine-learning algorithms that are imbued with some principles of the immune system, and attempt to take advantage of some of the benefits of the immune system to tackle difficult problem domains. A novel artificial immune system called IIDLE – the Immunological Inspired Distributed Learning Environment has been introduced previously concerning the techniques inspiration, conceptualisation, rudimentary architecture, and processes. IIDLE is inspired by the spatially distributed nature of the acquired immune system, and the clonal selection principle that describes how the immune system learns and adapts in response to stimulation from its environment. The platform is new and untested, thus can be considered immature, a concern this implication specification attempts to address. This work provides a discussion of an implementation of the IIDLE platform in Java, as well as an implementation specification for the processes of IIDLE. Particular attention is paid as to how to embed popular biologically inspired search and optimisation procedures in IIDLE such as genetic algorithms (GA), a local search procedure, and an ant colony optimisation (ACO) algorithm. Comment is provided as to how to implement two additional and different algorithms; the learning vector quantisation (LVQ) algorithm and particle swarm optimisation (PSO). The work is rounded off with a discussion of two very interesting extensions of the platform – IIDLE as a platform for interactive and collaborative search, and IIDLE as a distributed adaptive hybrid search system.

Acknowledgments

It can be hard to communicate a dry topic such as the implementation specification for a new algorithm or system. Thanks to Irene Moser for suggesting and Daniel Angus and Professor Tim Hendtlass for encouraging and supporting the development of this technical report.

Table of Contents

List of Figures

List of Equations

List of Tables

1 Introduction

The mammalian immune system is an intriguing natural defence system that has been shown to be capable of learning, memory, and adaptation. Conceptually, the active acquired immune system can be taken as a spatially distributed yet circulating (immuno-surveillance) and heterogeneous population of specialised discrete units that provide a homogeneous defence against external pathogenic material. The clonal selection principle is a theory that is core to the understanding of the acquired immune system and is accepted as the most plausible explanation for the systems learning, adaptive, and memory behaviours. The Immunological Inspired Distributed Learning Environment (IIDLE) is a machine-learning platform inspired by the spatially distributed nature of the acquired immune system and the clonal selection principle. A good introduction to artificial immune systems (AIS) and the biological immune systems is provided by de Castro and Timmis [14]. The inspiration, conceptualisation, architecture, and processes of IIDLE are described by Brownlee [12]. Some preliminary experiments with IIDLE such as partitioning, multiple search techniques and multiple objective search are discussed by Brownlee [13].

Although some preliminary works have been written discussing IIDLE as a machine learning platform, as of yet there has been no work describing how to implement the various active components of the system. This work seeks to improve the maturity of the system by providing pseudo-code level-detail for the modular elements of IIDLE, and providing instruction as to how to embed population-based search procedures into the system. It is expected that this work can be used both as a guide for interpreting the existing software implementation of IIDLE, and as a guide to construct a new implementation of IIDLE with sufficient detail to reproduce previously observed results in [13].

Section 2 provides a high-level overview of an existing software implementation of IIDLE in the Java programming language. Five technical aspects of the implementation are discussed, highlighting interesting characteristics of the software such as the use of object-oriented software engineering principles in the design and construction of the platform, and the exploitation of open source Java components. Section 3 introduces the conceptualisation of a *system-process*, which has a frequency of execution that governs simulation time, and has a number of associated work-units that represent the active IIDLE processes. The *movement*, *decay*, and *expansion* processes are discussed and described in pseudo-code detail. An additional process is introduced which provides a template for a distributed movement process for transferring units in batch across slower communication links (such as between distributed IIDLE systems on a computer network).

Section 4 discusses three search strategies that have been embedded in the IIDLE system as proliferation sub-processes of the expansion process. These strategies include a mutation-based local search algorithm, a simple genetic algorithm, and a novel discrete history version of ant systems. Section 5 provides instruction as to how to implement two additional search strategies in IIDLE as proliferation sub-processes. These include the learning vector quantisation (LVQ) algorithm for classification, regression, and function approximation, and the particle swarm optimisation algorithm (PSO) for function optimisation. Section 6 rounds off the work, providing comment on two interesting observations made whist describing IIDLE related to the

future of the platform. The first comment pertains to use of IIDLE as an alternative to the conventional configure-run-analyse cycle that is so prolific in the field of search and optimisation as a philosophy for applying these algorithms. The second pertains to the potential pervasive use of adaptive processes in IIDLE, specifically adaptive movement, decay and expansion. It is speculated that such a configuration could transform IIDLE into a truly novel distributed and adaptive hybrid search platform.

2 Implementation Overview

IIDLE is a framework inspired by the mammalian acquired immune system that provides a novel model of computation suitable for machine learning applications. It is described as a system or framework as opposed to an algorithm as it is composed of a number of processes (algorithms) and data structures. The system is modular by design and was implemented as a collection of loosely coupled elements with respect to both threaded processes and the data storage (architectural) structures. The following lists five fundamental features, which characterise the IIDLE implementation in software as a machine-learning framework. The intent is to highlight from a high-level, interesting and functional features of the IIDLE software implementation that further attempt to facilitate the principles and benefits of the immunological inspiration, and the design goals of the platform discussed in [12].

1. **Java Programming Language**: The IIDLE framework was implement in the Sun Microsystems Java programming language [16]. Java was selected for a number of reasons, most notably its platform independence (permitting it to execute on different operating systems and machine architectures), it's pervasiveness in both academic and enterprise applications, and finally its notoriety for Internet/Networking and easy of multithreading. Further, more recent and continuous advancements to the just-in-time complication in the Java runtime provides byte-code execution speeds *on par* with C and C++ for many scenarios (although still hotly debated by language evangelists on both sides).

2. **Object-oriented software engineering principles**: There was an effective use of object-oriented software engineering principles in the design and implementation of IIDLE. Encapsulation of logic, exploitation of design-patterns and data and ubiquitous use of polymorphism facilitate strong modularity in the framework as well as ease of extension and specialisation of base-level IIDLE data structures, processes, and search strategies.

3. **Off-the-shelf components**: An additional desirable feature of selecting the Java programming language is both the extensive default application-programming interface (API), and the wealth and quality of Java-based open-source software products that could be exploited by the IIDLE framework. A selection of these off-the-shelf modules/components used in IIDLE include the Pastry peer-to-peer (P2P) substrate [2], the JFreeChart charting package [6], the Jung network graph package [10], the Open Source Physics library [1], the Colt distribution by CERN [18], and finally Jakarta Commons by the Apache Foundation [15].

4. **Script-based configuration**: A scripting engine was employed as the primary means of configuring and setting up IIDLE simulations, as well as ultimately

tying together the loose modules of the framework. The scripting system facilitates all parameterisation of the system as well as defining the special case components to use during a simulation such as movement, decay and expansion processes, as well as architectural objects. This flexible scripting driven approach to simulation was inspired by the renown and successful ECJ (Evolutionary Computation in Java) evolutionary computation library [15], and further facilitates much of the flexibility of the system.

5. **Multithreaded processes**: The flexibility of the modular design coupled with the scripting engine permit a "*system process*" to be defined which operates in its own thread and contains one or more work units. Each work unit can be an IIDLE process such as decay or expansion. IIDLE was written to be thread-safe from the ground-up. Each work unit then has a defined scope on the system data structure such as a single locality, a partition, or the entire system (all localities). This implies that independent of the given underlying hardware system being used (number of CPU's) the system can be configured anywhere from a sequential set of actions that operate upon the IIDLE structure, to as many threads as there are localities multiplied by the number of processes (on thread per process per locality). It should be noted here that some work on parallel evolutionary algorithms have noted that the systems can provide benefit to the search even on non-parallel system hardware (single CPU) [7].

As has been demonstrated, the IIDLE software implementation is highly flexible, cross platform and fundamentally extensible making it amenable to embedding a search strategy of choice and applying the framework to a selected problem domain. The remainder of this work provides pseudo-code (language neutral) descriptions as to how to implement the various components of IIDLE.

3 Processes

Processes are the active component of the system that operate upon the IIDLE data structure. As described, a generic "*system process*" can be defined that supports a number of work units (IIDLE processes). This section provides an overview of a system process, and then proceeds to discuss each IIDLE process in turn (movement, decay, expansion).

A system process is implemented as a single thread of execution that operates in discrete simulation time. A process has a frequency (how often the thread executes each work unit, in milliseconds) which represents the interval of simulation time between executions. Thus, it is possible for two different processes to operate at different time scales (frequencies). This may be useful for rapid movement or decay independent of the learning achieved through the expansion process. Each system process has a number of work units associated with it, as well as a number of listening objects (subject-observer design pattern), which are triggered each process cycle.

Each work unit by default specifies a specific unit to execute (a fully-qualified java class-name which is instantiated with reflection), as well as a magnitude which indicates the number of times to execute the work unit per process cycle. This flexibility permits some work units more CPU time per process cycle than others, which may be useful in dynamic problem domains where rapid-adaptation is required independent of other IIDLE activities. As well as a magnitude, each work unit

specifies a system scope over which to execute. This scope configuration can be specified as singular localities or groups of localities.

---

1. **Initialise** *System Process*
2. **While** *System Process* **Not** Stopped **And Not** Paused **Do**
    a. **Wait For** *FrequencyTimePeriod*
    b. **For Each** *WorkUnit*
        i. **For Each** *Amplitude*
            1. **Execute** *WorkUnit*
        ii. **End**
    c. **End**
    d. **For Each** *Listener*
        i. **Trigger** *Listener*
    e. **End**
3. **End**

Figure 1 - Pseudo code for a system process showing the execution of work units and triggering of object listeners

Upon the execution of a work unit and object listener, the system state is made available at the configured scope (although with implicit thread-safe locking) permitting a wide range of activity from manipulating units, to graphing performance. Finally, a process can be started, stopped, paused, and resumed.

It is interesting to note that a process, the listeners, and work units that belong to it can be added, removed, and reconfigured dynamically (in real time during a simulation). This is an impressive level of flexibility that permits not only the reconfiguration of existing movement and decay processes, but the intelligent reconfiguring or adding of entirely new search processes by a human operator during a simulation. This capability coupled with a distributed implementation of the system is likely to lead to some interesting collaborative optimisation experiments. This area of human-interactive search and optimisation with the IIDLE platform remains an area for further research, and is discussed further in section 6.

3.1 Movement

Movement is the process of removing units from the tail of one locality, and pasting them in the tail of a neighbouring locality. In the context of system processes described above, movement is a work unit. The activity is inspired by the circulatory system that provides the transportation of lymphocytes (a critical element of the acquired immune system) throughout the body of the host organism. The movement process in IIDLE facilitates the expendability of the discrete units across the system. Movement cut-and-pastes units as opposed to copy-and-pasting units between localities. The reasoning for this is two-fold:

1. Matching with the inspired biological system with the intent of facilitating the redundancy of information

2. Enforcing the only process to create new units is the expansion process (a copy is a creation of a new unit, impacting resource allocation and inturn maintenance)

The following pseudo code listing provides a generic implementation of the movement activity.

---

1. **For Each** *locality* **In** *scope*
    a. **If** *LocalityUnits* < 1
        i. **Continue**
    b. **Else If** *LocalityNeighbours* < 1
        i. **Continue**
    c. **Else If** *Rand() > MovementProbability*
        i. **Continue**
    d. **End**
    e. **Select And Remove** *n units*
    f. **Select** *LocalityNeighbour*
    g. **Place** *n removed* **In** *SelectedNeighbouringLocality*
2. **End**

Figure 2 - Pseudo code for the movement process work unit in IIDLE

The movement activity cannot be executed on a locality that either has no units in its tail at the time of execution, or has no neighbouring localities at the time of execution. A selection method must be defined to choose those elements of a given locality that will be migrated as well as how many units to migrate. A neighbour selection strategy is required to choose the locality neighbour (or neighbours) to which the selected units will be migrated. Finally, a replacement strategy is required to add the migrated elements to the neighbouring locality. The default action for all three of these strategies is as follows:

Strategy	**Default**
Selection Strategy	Random
Number Migrated	One
Neighbour Selection Strategy	Random
Number of Neighbours	One
Replacement Strategy	Concatenation

Table 1 - Default movement configuration

The reason random migration strategies were selected was because they are unit-neutral as in the inspiration metaphor (not considering quality measures). Low magnitude migration was used primarily because by keeping the value low and constant, the influence can be adjusted by changing the magnitude and/or frequency for which the overall work unit is executed (system process level). Both random and constant frequency migrations are suggested as a base configuration for parallel evolutionary algorithms as described in [7].

A potentially useful extension to the default probabilistic movement strategy is to use an adaptive movement strategy. As in the case of the conformer homeostasis approach used in decay, adaptive movement would seek to maximise stimulation of the better quality units in the scope of the process. The movement module would measure the amount of stimulation the localities in scope are exposed to and based upon the probability of a locality (in scope) being exposed to stimulation per movement cycle (frequency of system process), adjust the movement strategy accordingly to maximise the stimulation of units. In this scenario the probability of movement would be:

$$p(movement) = 1 - \left(\frac{stimulations\_per\_cycle}{localities\_in\_scope} \right)$$

Equation 1 - Probability of movement if movement was adaptive in response to the amount of stimulation per execution cycle

Thus, the less stimulation detected per cycle, the more movement. This may lead to efficiencies such as the implicit absence of movement once stimulation reaches a threshold (probability of one per locality in scope).

3.2 Decay

Decay in the system exists to maintain resources, specifically in response to the creation of new units by the proliferation strategy in the expansion process. Each unit has finite energy, and it is the task of the decay process to decay the energy of each unit, and remove those units that deplete all energy. There are two primary approaches to manipulating unit energy, they are; the *conformer homeostasis* approach and the *regulator homeostasis* approach.

The first approach (*conformer homeostasis*) is a closer fit to the processes within the inspired immunological metaphor. The system regulates (decays) energy uniformly across all units in the scope of the process. Thus, in times of increased load, the affected localities will also have an increased number of units, as the decay process will make no effort to return the system to a point of equilibrium. Instead, the stimulation, the proliferation (creation) process, and ultimately the environment of the situated system define the amount of resources the system will allocate and sustain. Alternatively, is the case of little stimulation or complete inactivity with regard to stimulation, the decay process will eventually decay away and remove all units from its scoped localities. The following pseudo code listing provides an overview of a conformer-based decay process.

---

1. **For Each** *Locality* **In** *Scope*
    a. **If** *LocalityUnits* < 1
        i. **Continue**
    b. **Else If** *LocalityNeighbours* < 1
        i. **Continue**
    c. **End**
    d. **For Each** *Unit* **In** *Tail*
        i. *UnitEnergy* -= *EnergyConstant*
        ii. **If** *UnitEnergy* <= 0
            1. **Remove** *Unit*

```
            iii. End
        e. End
2. End
```

Figure 3 - Pseudo code for the conformer (homeostasis) approach to decay in IIDLE

In the case of conformer-based decay, additional constraints may be implemented to overcome the two stimulation extremes (over and under stimulation). In the case of excessive stimulation, a maximum number of units may be specified for each tail, thus setting the locality of maximal stimulation for the locality where no new units can be created or migrated to the point until the decay process is given a chance to catch up. In the case of lack of stimulation, the system may impose a minimum number of units, which must be maintained, and thus decay is not executed when this lower bound is reached. An alternative to imposing a *lower unit bound* is to add an additional "insertion" process that generates and adds new random units, sustaining a state of entropy and readiness for renewed stimulation. This final configuration may be suitable for dynamic problems in which minimal resource allocations and rapid adaptation are required.

The second type of decay is *regulator homeostasis* in which the process attempts to maintain a system variable at equilibrium, in this case, the total unit energy in the scope of the process. Firstly, this obviously requires the specification of a total unit energy, which does not directly correspond to the total number of units. In this configuration the system fist evaluates the total energy across all scoped localities and counts the total number of units. The remainder of the approach is the same as conformer-based decay in which a energy constant is subtracted from each unit, except in this case the constant is taken as the ratio of the difference between measured and ideal energy across all scoped units. Thus, after execution, the process assumes the scoped localities contain the ideal amount of energy. If energy is below the ideal, then the decay process is not executed. The following provides a pseudo code listing of the regulator-based decay process.

```
1. For Each Locality In Scope
        f. If LocalityUnits < 1
            i. Continue
        g. Else If LocalityNeighbours < 1
            i. Continue
        h. End
        i. SystemEnergy = 0
        j. TotalUnits = 0
        k. For Each unit In tail
            i. SystemEnergy += UnitEnergy
            ii. TotalUnits++
        l. End
        m. If SystemEnergy > IdealEnergy
            i. EnergyDec = (SystemEnergy - IdealEnergy) / TotalUnits
            ii. For Each Unit In Tail
                1. UnitEnergy -= EnergyDec
```

2. **If** *UnitEnergy* <= 0
    a. **Remove** *Unit*
3. **End**

iii. **End**

n. **End**

2. **End**

Figure 4 - Pseudo code for the regulator (homeostasis) approach to decay in IIDLE

An extension to these decay processes may be some hybrid approach that is dynamic as in the regulator approach, though perhaps slow to adapt to facilitate some of the effects of the conformer-based approach.

3.3 Expansion

The expansion process is responsible for the adaptation and ultimately the learning within the system. Broadly, this process is referred to as expansion, after its inspiration from the clonal expansion principle (a.k.a. the clonal selection principle). It can be conceptualised as three sub-process as follows; *stimulation*, *selection* and *proliferation*.

Complete control over the stimulation implies that it is the discretion of the user as to the amount of stimulation to apply to the system per cycle from one per locality, to one per locality scope. Such a configuration is suitable for conventional static search or optimisation problems. For a static objective function, those units that have already been evaluated do not require revaluation, thus can be skipped by the stimulation process. Stimulation equates to proliferation in a locality, and thus not only does the stimulation frequency and magnitude reflect the amount of objective function evaluations, it also reflects the amount of re-sampling (resources) allocated each cycle.

The question as to how to best configure the system for static objective functions has not been resolved. A default stimulation approach is employed based upon the triggered *stimulation-activity* nature of the inspiration biological metaphor. A single expansion process is configured for a number of localities, of which one is selected for stimulation per execution (note, the configured magnitude of the work unit permits multiple executions per cycle). A parametric stimulation function is used to select the locality to stimulate each execution, such as a simple Uniform or Gaussian distribution. This simple stimulation configuration is flexible enough to be scaled to various alternative configurations (such as one per locality per cycle).

The following figure provides a pseudo code overview of the expansion process with the three key sub-processes.

---

1. Select *locality* **In** *scope* **For** *stimulation*
2. **For Each** *unit* **In** *tail*
    a. Stimulate *unit*
3. **End**
4. Select *n units*

5. Proliferate *n* selected *units* **Into** *m progeny*

Figure 5 - Pseudo code for the high-level expansion process in IIDLE

There is no evidence to support the specified probabilist stimulation approach as being the best for static objective functions as mentioned, although this configuration does provide an interesting additional configuration benefit besides its ability to mimic many different configurations. The probabilist-based stimulation (using a parametric function) can be easily adjusted dynamically by a user independent of the selection strategy, the proliferation strategy, and other processes in the system. Such reconfigurations may include manipulation of the stimulation function parameters (changing distributions), or replacement of the locality-selection strategy itself.

A further additional interesting extension facilitated by the probabilistic stimulation approach is a configuration that adapts the locality selection strategy in response to the efficacy of units in localities. This approach would both redistribute computation time and memory resources in response to the spatially distributed quality of solutions. This intriguing configuration is expected to be most beneficial in scenarios that partition the data structure using either different objective functions and or different search strategies, providing a potentially powerful adaptive hybrid search technique. This proposal is discussed further in section 6.

The stimulation strategies discussed thus far pertain to problem domains in which there is control over the systems exposure to stimulation (external information from the environment in which the system is situated). This may not be the case as in domains that use information sources that have variable load or geographically distributed information sources. The probabilist stimulation described above is nothing more than an approximation of these online dynamic problems. In these scenarios, an information feed can be designated a single locality, or perhaps equally distributed across a number of localities in a partition. This final point highlights that using the specified stimulation strategy, the system facilitates both stimulation sources effectively be it a contrived external feed as in the case of search and optimisation or a feed from an authentic external system.

3.4 Distributed Movement Process

There are a number of ways in which the system architecture could be implemented on parallel and distributed hardware, as has already been discussed. A recommended configuration for IIDLE when exploiting a homogeneous computer network (as in a computer lab at a university) or a heterogeneous computer network (as in a collection of desperate server platforms) is to have a collection of localities on each system. Individual IIDLE systems are connected using a network topology (such as a small-world peer-to-peer orientation). The conventional IIDLE movement process is used to move units around within each IIDLE setup, though is an inappropriate migration strategy across slower communication mediums such as network connections. An additional network-friendly movement strategy is proposed for use in such a scenario called *store and forward*.

In each IIDLE systems configuration a number of localities are selected to become "*portals*". These portals provide a gateway to network topological neighbouring

systems to which units cane be transmitted to using an *Outgoing* movement process, and from which units can be received using an *Incoming* movement process.

The following figure provides a pseudo code listing of the outgoing movement process. Both the incoming and outgoing processes assume an underlying network communication API to send and receive collections of units between hosts and facilitate the desired network topology. In the Java implementation of IIDLE, preliminary experiments used the Java 2 Enterprise Edition (J2EE) Remote Method Invocation (RMI) API for direct communication between hosts. Later preliminary experiments used the Pastry P2P substrate [2].

---

1. **For Each** *PortalLocality* **In** *Scope*
    a. **If** *LocalityUnits* < 1
        i. **Continue**
    b. **Else If** *RemoteLocalityNeighbours* < 1
        ii. **Continue**
    c. **End**
    d. **Select** *RemoteLocalityNighbour*
    e. **Transmit** *Units*
2. **End**

Figure 6 - Pseudo code for the outgoing distributed movement process

A portal locality has a tail, stores units and is connected to local neighbouring localities as in a normal locality. When distributed, conceptually, IIDLE systems are connected to each other using a network-topology as discussed, and the portal localities provide an entry and exit point for units across the communication link (note other low-speed links could be used such as Inter-Process Communication (IPC), or remote database access). Both the incoming and outgoing processes are nothing more than work units that are configured within the context of an IIDLE system process as with previously discussed processes (movement, decay, expansion).

When the outgoing process executes, all units currently in the localities tail are removed from the system and transmitted to a neighbouring network IIDLE system. A default random neighbour selection process is employed, although more intelligent or adaptive neighbour selection strategies could be used. The following figure provides a pseudo code listing for the incoming process.

---

1. **For Each** *PortalLocality* **In** *Scope*
    a. **If** *TotalIncomingUnits* < 1
        i. **Continue**
    b. **End**
    c. *UnitsPerPortal = TotalIncomingUnits / TotalPortalLocalitys* **In** *Scope*
    d. **While** *TotalIncomingUnits > 0* **Do**
        i. **Select** *PortalLocality*
        ii. **Select** *LocalNeighbouringLocality* of *PortalLocality*
        iii. **Place** *UnitsPerPortal Units*

iv. *TotalIncomingUnits -= UnitsPerPortal*

e. **End**

2. **End**

Figure 7 - Pseudo code for the incoming distributed movement process

The incoming process is responsible for distributing the units received by the system from potentially multiple neighbouring systems. Each portal of the scope of the process is allocated a portion of the received units, which are distributed to the portal's local neighbours. Those units received from remote systems are marked as such, and cannot be re-distributed to another system (cannot be sent by an outgoing movement process). The resulting effect is an efficient store and forward distributed movement process. Portal localities capture and transmit units, as well as receive and pass on received units to neighbouring localities.

A final thought on distributed movement is the application of more intelligent processes. Portals are an architectural feature of the system (data structure), and as described movement is a process. A more intelligent distributed movement process could actively move around a local data structure, filtering and collecting encountered units. This "spider" like process (analogy from processes that craw the internet for changes to web pages) would transmit gather data when it migrated to a portal. This extension is intriguing when it is considered that all manner of greedy, and probabilistic unit-selection routines can be embedded in these spider processes.

4 Search Strategies

Proliferation is one of the three sub-process of expansion which defines the way in which existing search-space sampling (units) are exploited to generate new samples in the search space. Thus, proliferation is a search strategy (sampling strategy), and is critical to the learning, and adaptation fundamental to the function of the IIDLE system. The proliferation sub-process facilitates the embedding of existing successful search strategies, as well as newly developed strategies specialised to IIDLE. This section details three proliferation approaches, which have been implemented in IIDLE.

In the following cases, only the proliferation strategy is discussed, thus it is assumed that a stimulation procedure and selection strategy are executed prior to the execution of the proliferation strategy. It is important to note that both of these prior sub-process have a marked impact on the sampling procedure of a proliferation strategy. In the case of a new search strategy embedded in IIDLE, it is expected that a specialised selection strategy will also be embedded in the same expansion process (for example tournament selection with a genetic algorithm, and neighbourhood selection with particle swarm optimisation).

4.1 Mutation-based local search

Mutation-based search is a rudimentary variation of directed random search. Given existing good candidate solution's (in this case units), the mutation-based local search will clone these solutions and vary them in some small way (the degree of which is likely defined by a user parameter). The intent of this directed random search procedure is to generate new samples within the "neighbourhood" of the base samples

in the context of the search or parameter space of the problem domain understudy. The following figure provides a pseudo-code overview of the mutation-based local search proliferation strategy.

---

1. **If** *SelectedUnits* **Is** < 1
    a. **Continue**
2. **End**
3. **For Each** *Unit* **In** *SelectedUnits*
    a. Duplicate *Unit*
    b. Mutate *Unit*
4. **End**

Figure 8 - Pseudo code for a generic mutation-based local search proliferation strategy

The mutation function is problem specific and may be as simple as the flipping of bits in a bitstring, the swapping of choices in a permutation or the generation of an n-dimensional point in a bounded continuous space using a transition function. The strategy should be used in conjunction with either a probabilistic selection strategy with a heavy bias, or a greedy selection strategy that selects the best units in the localities tail. The reason for this is to provide the directedness to this local search procedure.

4.2 Genetic algorithm

The genetic algorithm (GA) is a widely used global search procedure from the field of evolutionary computation (algorithms inspired by Darwin's theory of natural selection [3]). The standard or canonical genetic algorithm [5] was selected to be embedded into the IIDLE system given both the easy of implementation and mass of research into the approach in the field.

---

1. **If** *SelectedUnits* **Is** < 1
    a. **Continue**
2. **End**
3. **If** (*SelectedUnits* % 2) <> 0
    a. **Continue**
4. **End**
5. **For Each** *Pair of Units* **In** *SelectedUnits*
    a. Crossover *ParentUnits* (create two *progeny*)
    b. Mutate *progeny*
6. **End**

Figure 9 - Pseudo code for a genetic algorithm based proliferation strategy

The standard genetic algorithm uses a bitstring representation. In this simple pseudo-code description an even number of parental units are required. Each pair of parents produces a pair of progeny using one-point crossover. After creation, the progeny are mutated using a low probability bit-flipping based mutation strategy. A probabilistic

selection strategy is recommended with this proliferation strategy (such as tournament selection), although given the fundamental diversity of the IIDLE system, a greedy strategy as in mutation-based local search may also be useful (experimentation recommended).

4.3 Discrete history ant systems

Ant colony optimisation (ACO) refers to a field of research into course-grained combinatorial optimisation algorithms inspired by the foraging behaviour of real ant insects. An implementation of ACO called ant system (AS) was embedded in the IIDLE system as a proliferation strategy. The default specification of ant systems uses what is referred to as a pheromone matrix (history matrix) to store and represent historic samples from solution space. Given that information or samples in IIDLE are represented as discrete units, the AS algorithm was modified to exploit this population-based history. This novel variation of AS is referred to as discrete history ant systems (dhAS), and is explained and extended in an upcoming paper by Angus and Brownlee [4].

The essence of the variation is that a population of discrete solutions is maintained and compressed to a pheromone matrix for use in a standard probabilistic stepwise construction strategy exploited by all ACO algorithms. The novelty of the history representation is that it permits a finer-granularity manipulation of historic samples that make up the pheromone matrix. The following figure provides a high-level pseudo-code overview of the discrete history ant systems algorithm for use in IIDLE.

---

1. **If** *SelectedUnits* **Is <** 1
    a. **Continue**
2. **End**
3. Compress *SelectedUnits* **Into** *PheromoneMap*
4. **For Each** *Unit* **In** *ProgenyToCreate*
    a. **For Each** *Choice* **In** *SolutionPermutation*
        i. Combine *PheromoneMap* **And** *Heuristic*
        ii. Make *StepWiseChoice*
    b. **End**
5. **End**

Figure 10 - Pseudo code for a discrete-history ant systems based proliferation strategy

The heuristic may require initialisation before the search, such as in the case of the Travelling Salesman Problem (TSP), where a city-city distance matrix must be prepared. A complete (all) selection strategy is used that makes use of the entire population of units in the tail of the stimulated locality. On creation of the pheromone matrix, the contribution of each unit to the matrix is taken as a ratio of the best scoring of the current units scoring.

$$contribution = \frac{bestScore}{curentScore}$$

Equation 2 - Equation used to determine each units contribution to the pheromone matrix

This is different to the contribution scheme proposed in the upcoming paper, which is based upon solution age and quality [4]. The reason for the variation is to provide a quality rank-based (fair) contribution scheme given the transient contents of the localities tail, which was independent of the short life spans of individual units in the system. The scheme proposed by Angus and Brownlee [4] uses unit age as the scheme is tightly coupled with the stack-based population maintenance mechanism (decay).

5 Additional Search Processes

The previous section demonstrated the easy in which popular search strategies can be embedded and adapted into IIDLE as proliferation strategies. This section proposes how two additional search strategies can be embedded in IIDLE. Specifically the algorithms discussed are the Learning Vector Quantisation (LVQ) classification algorithm, and the Particle Swarm Optimisation (PSO) algorithm for function optimisation.

5.1 Learning Vector Quantisation

Learning Vector Quantisation (LVQ) 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 [17]. 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(s) in the collection are 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.

A variation of the LVQ algorithm can be embedded into IIDLE that, instead of the moving best matching units toward input patterns, would adjust the exemplar pattern of progeny of best matching units. In scenarios in which there is control over the test patterns (finite size dataset), random groups of patterns could be drawn upon from the training set and exposed to randomly selected localities. The effect would be the entire system (all localities) acting as the classification system. Thus, the expectation is that test and validation patterns could be evaluated at any randomly selected locality of the system. Such redundancy is expected to require an increased allocation of resources, and thus a reduced number of localities may be a necessary configuration trade-off. If demonstrated successfully, IIDLE would provide a powerful and robust classification, regression, and function-approximation system for dynamic domains and perhaps for larger scale applications such as novelty detection and intrusion detection.

5.2 Particle Swarm Optimisation

Particle swarm optimisation (PSO) [8,9] 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 [11]). 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).

Like the genetic algorithm (GA), the PSO algorithm is amenable to implementation in IIDLE given that the algorithm is already population based, and does not required modification to make it so, as in ant systems (AS). Each unit in the system would represent a PSO particle and posses both a position in the search space and a directional-vector for movement in the search space. External stimulation will provide the trigger to advanced each particles movement (resample the search space) based upon those units in the current locality. Like the genetic algorithm, this algorithm is expected to provide an intriguing system for analysis given that a "population" does not truly exist, instead is an instantaneous notion.

An embedded PSO proliferation strategy is expected prove most beneficial in function optimisation given it is purported to be more efficient than a genetic algorithm in the same domain. Further, given that each particle has a trajectory, it is expected that the algorithm will prove beneficial in dynamic problem domains that are consistent concerning changes over time (particles are expected to effectively track objective peaks).

6 Conclusions and Further Research

A software implementation of IIDLE has been written in the Java programming language exploiting many of the benefits that platform offers such as off-the-shelf components, strong object-oriented design principles, and a strong focus on multithreading and network communication. The active elements of IIDLE were discussed in depth. A conceptual *system process* was introduced which is configured with a frequency (how often it executes), and a number of work unit which represent the active "processes" of the IIDLE system such as *movement*, *decay*, and *expansion*. A distributed movement process was introduced as a more efficient manner of moving units in batch across slower communication links. Three proliferation (search) strategies were introduced and described in terms of their broad implementation details when embedded in IIDLE. These strategies included a mutation-based local search strategy, a simple genetic algorithm, and a novel variation of the ant systems algorithm. Two additional search strategies were proposed as good candidates for future implementation in the platform. They were the Learning Vector Quantisation algorithm for classification, regression, and function approximation, and the Particle Swarm Optimisation algorithm for function optimisation.

IIDLE is a novel machine learning system inspired by the acquired immune system, though it is new and not yet mature. The intent of this work was to improve the maturity of the platform by communicating sufficient detail to interpret the existing IIDLE implementation, and with the help of the introduction to IIDLE [12], design and construct a new implementation of the system. In addition to the implementation and investigation of the proposed additional search strategies, there were two primary outcomes of this work for future research.

1. **Alternative to Configure-Run-Analyse cycle**: Conventionally, search and optimisation problems (at least for static problems) are addressed by executing a configure-run-analyse (CRA) cycle, which typically involves performing a sensitivity analysis across multiple parameter configurations. A widely known

critique of the biologically inspired techniques discussed (which extends to IIDLE) is that although the systems are parameterised, there are complex non-linear relationships between parameters making the system very difficult to configure, thus leading to the CRA cycle. IIDLE provides an alternative to this cycle, which is real-time user interaction and collaboration.

Users configure IIDLE with search processes (elements of the expansion process) they are most familiar with (experts in) and refine the search process by interacting with the system in real-time. This interaction and reconfiguration of the system by a single user on a single system is a continuous process, which is different to the discrete and sequential CRA cycle. This alternative can be made more robust by capturing the results of interaction by multiple experts in parallel across a number of IIDLE systems connected together in a network configuration. This simple premise for real-time interactive collaborative search both demonstrate the flexibility of IIDLE and provides a potential glimpse at the future of large-scale Internet-based optimisation tasks, that can be addressed with IIDLE.

2. **Adaptive Strategies**: Embedding a single search in IIDLE such as a Genetic Algorithm or Particle Swarm Optimisation is somewhat interesting when considered from a *stimulation-activation* (immunological) context, although conceptually differs very little from the canonical definition of the algorithm. As discussed in [13] the potential benefits of IIDLE are not made apparent until configurations are considered that exploit multiple search techniques, multiple objective functions, and multiple representations in parallel. An extension to this novelty of hybrid search is in making aspects of the system themselves adaptive.

   The description of the IIDLE processes highlighted that point that each process can easily be adjusted so that its behaviour is adapted in response to the directedness and efficacy of the search. The frequencies and amplitude of the movement process adapt based on observed stimulation patterns, increasing the probability of good-quality units being stimulated and proliferated. Decay can be configured to operate in a *conformer* or *regulator* modes to self-regulate resource allocations in response to the behaviour of external stimulation. Finally expansion, in particular the stimulation procedure (if it can be controlled) and the proliferation strategy can be adjusted to focus more or less attention across localities based upon the directedness or quality of the underlying search process. This final point further highlights a second exciting glimpse of the potential future of IIDLE as a distributed adaptive hybrid search platform.

There is still much work to do on IIDLE including a description of the novelty that the platform offers in the context of immunological and hybrid search, as well as additional preliminary experiments described in [13].

7 Bibliography

[1] Open Source Physics. [Online] http://www.opensourcephysics.org/ . 2005. 2005.

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *Proceedings IFIP/ACM International Conference on Distributed Systems Platforms (Middleware),* Heidelberg, Germany, pp. 329-350, 2001.

[3] Charles Darwin. *On the origin of species by means of natural selection, or, The preservation of favored races in the struggle for life*, Champaign : Champaign, Ill. (P.O. Box 2782, Champaign 61825) : Project Gutenberg, [199-?]., 1859.

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

[5] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, USA, Canada: Addison Wesley Publishing Company, Inc., 1989.

[6] David Gilbert and Thomas Morgner. JFreeChart. [Online] http://www.jfree.org/jfreechart/index.php . 2005. 2005.

[7] E. Alba and M. Tomassini, Parallelism and evolutionary algorithms *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 443-462, 2002.

[8] 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.

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

[10] J. O'Madadhain, D. Fisher, P. Smyth, S. White, and Y.B. Boey, Analysis and Visualization of Network Data using Jung *Journal of Statistical Software*, vol. 2005.

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

[12] Jason Brownlee, 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.

[13] Jason Brownlee, 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.

[14] Leandro N. de Castro and Jon Timmis. *Artificial Immune Systems: A new computational intelligence approach*, Springer-Verlag, 2002.

[15] Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Jeff Bassett, Robert Hubley, and Alexander Chircop. ECJ - A Java-based Evolutionary Computation and Genetic Programming Research System . [Online] http://cs.gmu.edu/~eclab/projects/ecj/ . 2005. George Mason University's ECLab - Evolutionary Computation Laboratory. 2005.

[16] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*, USA: Addison-Wesley Professional, 1996.

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

[18] Wolfgang Hoschek and High Energy Physics at CERN. The Colt Distribution - Open Source Libraries for High Performance Scientific and Technical Computing in Java. [Online] http://hoschek.home.cern.ch/hoschek/colt/ . 2002. 2005.