A Framework for Evolutionary Optimization Applications in Water Distribution Systems

Submitted by Mark Stephen Morley, to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Engineering, March 2008.

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

..... (signature)

Abstract

The application of optimization to Water Distribution Systems encompasses the use of computer-based techniques to problems of many different areas of system design, maintenance and operational management. As well as laying out the configuration of new WDS networks, optimization is commonly needed to assist in the rehabilitation or reinforcement of existing network infrastructure in which alternative scenarios driven by investment constraints and hydraulic performance are used to demonstrate a cost-benefit relationship between different network intervention strategies. Moreover, the ongoing operation of a WDS is also subject to optimization, particularly with respect to the minimization of energy costs associated with pumping and storage and the calibration of hydraulic network models to match observed field data.

Increasingly, Evolutionary Optimization techniques, of which Genetic Algorithms are the best-known examples, are applied to aid practitioners in these facets of design, management and operation of water distribution networks as part of Decision Support Systems (DSS). Evolutionary Optimization employs processes akin to those of natural selection and "survival of the fittest" to manipulate a population of individual solutions, which, over time, "evolve" towards optimal solutions. Such algorithms are characterized, however, by large numbers of function evaluations. This, coupled with the computational complexity associated with the hydraulic simulation of water networks incurs significant computational overheads, can limit the applicability and scalability of this technology in this domain.

Accordingly, this thesis presents a methodology for applying Genetic Algorithms to Water Distribution Systems. A number of new procedures are presented for improving the performance of such algorithms when applied to complex engineering problems. These techniques approach the problem of minimising the impact of the inherent computational complexity of these problems from a number of angles. A novel genetic representation is presented which combines the algorithmic simplicity of the classical binary string of the Genetic Algorithm with the performance advantages inherent in an integer-based representation. Further algorithmic improvements are demonstrated with an intelligent mutation operator that "learns" which genes have the greatest impact on the quality of a solution and concentrates the mutation operations on those genes. A technique for

implementing caching of solutions – recalling the results for solutions that have already been calculated - is demonstrated to reduce runtimes for Genetic Algorithms where applied to problems with significant computation complexity in their evaluation functions. A novel reformulation of the Genetic Algorithm for implementing robust stochastic optimizations is presented which employs the caching technology developed to produce an multiple-objective optimization methodology that demonstrates dramatically improved quality of solutions for given runtime of the algorithm.

These extensions to the Genetic Algorithm techniques are coupled with a supporting software library that represents a standardized modelling architecture for the representation of connected networks. This library gives rise to a system for distributing the computational load of hydraulic simulations across a network of computers. This methodology is established to provide a viable, scalable technique for accelerating evolutionary optimization applications.

Keywords

Evolutionary Optimization, Genetic Algorithms, Hydroinformatics, Caching, Multiple-Objective Optimization, Distributed Computing.

Acknowledgements

I am deeply grateful to my family for their enduring support over the period that I have been working towards this thesis. Their consistently positive attitude has ensured that this thesis has finally come to fruition.

In addition, I would like to thank my supervisor, Professor Dragan Savić, not least for his forbearance and perseverance in the face of persistent obduracy against completing this thesis, but also for (twice) rescuing me from the wilds of the commercial sector and introducing me to the academic environment at the Centre for Water Systems in Exeter.

Foremost amongst my (erstwhile) colleagues I would like to express my appreciation to Mr. Roger Atkinson and, latterly, Dr. Carla Tricarico for attempting to instil in me the belief that this thesis could be completed in a meaningful fashion – and that doing so was worthwhile. I would also like to thank Mr. Josef Bicik, Drs. Zoran Kapelan, Ed Keedwell, Francesco di Pierro and Darko Joksimović for their help and support along with all my other colleagues and friends that I have met over *so* many years at the Centre.

Finally, to my two sons, Alexander and Christopher, to whom this thesis is dedicated: without them, it would surely have been completed much sooner ☺

Mark S Morley
Exeter
March 2008

Table of Contents

List of Tables

List of Figures

Glossary

Definitions

Allele — The *alleles* of a gene are the set of values that this gene can take. The most straightforward example taken from biology is that of the gene that determines eye colour. The alleles of this gene are the different colours (Brown, Blue...etc...)

Application Programming Interface — A specification for the public interface to a software library to be used by developers.

Chromosome — Many workers in the field of evolutionary algorithms use the term chromosome instead of *organism* above. As will be shown, it is convenient to maintain a distinction between the two (as well as maintaining the biological analogue) and to preserve the *chromosome* moniker to describe a group of *genes* that are related in some fashion. Thus, a *chromosome* is some sub-division of an *organism's genome*.

Common Object Model — A technology developed by Microsoft for the implementation of a componentized software architecture featuring a standardized API for the introspection of methods and members. Microsoft Windows specific. Largely supplanted by the .NET technology but still underpinning many Microsoft Products (e.g. Office).

Distributed Common Object Model. — As *COM* but with additional functionality to allow the instantiation of objects remotely, across a network.

Dynamic Link Library — A mechanism for dynamically linking software functions into an application. Promotes componentization through code reuse – applications can be composed from smaller building blocks. Unlike COM, however, DLLs do not have a standard mechanism for implementing object classes – which has

	resulted in compiler vendors implementing their own, incompatible techniques for doing so.
Elitism	Applied to generational GAs, elitism allows the most fit individuals from a source population to transfer directly to the destination population without undergoing recombination – ensuring that they are preserved from generation to generation.
Fitness	The measure by which individual organisms are compared to each other to judge their relative suitability for the optimization problem at hand. For single objective algorithms this is a single value, often combined with a *penalty function* to penalise constraint violation.
Gene	Genes are the fundamental unit of genetic algorithms. Each gene represents a specific attribute that is encoded within the genome at a specific location known as a *locus*. This attribute normally represents a decision variable to be considered in the optimization but can also convey other information specific to an *organism*.
Generational GA	A type of Genetic Algorithm used for both single and multiple objective optimization. Following selection and recombination, child organisms are inserted into a *new* population rather than replaced into the existing population as with a Steady-State GA. The selection and recombination process continues until the new population reaches the nominal size of the previous generation. The new population then forms the basis for selection for the next generation.
Genome	The total genetic representation of an organism is described as its genome. In mammalian biology, a genome is ordinarily sub-divided into chromosomes.
Genotype	The representation of the information contained within the genome in its native form. For example, a decision variable

	containing the value 5 may represent a pipe diameter of 80mm. The genotypic value of the decision variable is 5.
Locus	The locus of a *gene* is the position it occurs in a *chromosome*. In Genetic Algorithms, the locus is usually fixed for a given gene. However, in certain types of GA with variable-length or *heterozygous chromosomes* the locus can vary.
Network Calibration	An optimization problem to calibrate a Water Distribution System hydraulic model to match observed field data – commonly by varying pipe friction factors or diameters.
Network Design	An optimization problem to layout new pipes for a Water Distribution System.
Network Rehabilitation/Reinforcement	An optimization problem that determines intervention strategies in a WDS for rehabilitating existing pipes or reinforcing the network through pipe duplication.
Organism	This is the representation of an individual in the *population*. It is a term not normally associated with genetic algorithms and is a by-product of the underlying object-oriented library described here. Conventionally, the term "chromosome" has been the preferred term for an individual in GAs. However, because this use of chromosome is significantly at variance with the biological analogue – and the necessity for the object-oriented library to have an appropriate naming strategy for classes – it was decided to break with convention and to use "*organism*" as the fundamental unit of a *population* instead. Because each *organism* always has exactly one *genome* and that *genome* can only belong to a single *organism* they can be considered concomitant.
Penalty Function	A function commonly used in single objective optimization to represent constraint violation by an individual organism – normally used in combination with a *fitness* value to give an overall relative measure of fitness for a solution.

Phenotype	The representation of the information contained within the genome in its appliedform. For example, a decision variable containing the value 5 may represent a pipe diameter of 80mm. The genotypic value of the decision variable is 80 when translated into the domain of the solution.
Population	A collection of organisms – the pool of genetic material operated on by a Genetic Algorithm.
Recombination	The derivation of child solutions from their parents – ordinarily as a result of crossover and mutation.
Steady-state GA	A type of single-objective Genetic Algorithm in which a pair of solutions are selected from a population, recombined to form two children and those children inserted, according to some rule, back into the original population. c.f. Generational GA.

List of Abbreviations

ACS	Ant Colony Simulation
ADSL	Asymmetric Digital Subscriber Line
AMS	Asset Management System
ANSI	American National Standards Institute
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CA	Cellular Automata
COM	Common Object Model.
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CWS	Centre for Water Systems, University of Exeter
DCOM	Distributed Common Object Model
DLL	Dynamic Link Library
DMA	District Metered Area
DSS	Decision Support System
DTD	Document Type Definition (XML)
EPS	Extended Period Simulation
FLV	Float Valve
GA	Genetic Algorithm
GAP	Generalized Assignment Problem
GIS	Geographic Information System
IP	Internet Protocol
ISO	International Standards Organisation
LAN	Local Area Network
LP	Linear Programming

LSB	Least Significant Bit (binary number)
MGA	Messy Genetic Algorithm
MOGA	Multiple Objective Genetic Algorithm
MSB	Most Significant Bit (binary number)
MTV	Motorised Throttle Valve
NLP	Non-Linear Programming
NRGA	Non-Repeating Genetic Algorithm
NRV	Non-Return Valve
NSGA	Non-dominated Sorted Genetic Algorithm
NYT	New York Tunnels (benchmark problem)
OLE	Object Linking & Embedding
OOTEN	Object-Oriented Toolkit for EPANET
OSGB	Ordnance Survey of Great Britain
PBV	Pressure Break Valve
PDD	Pressure-Driven Demand
PDF	Probability Density Function
PRV	Pressure Reducing Valve
PSG	Piedemonte San Germano (benchmark problem)
PSO	Particle Swarm Optimization
PSV	Pressure Sustaining Valve
RCV	Remote Control Valve
RDBMS	Relational Database Management Software
rNSGA-II	Robust NSGA-II
SA	Simulated Annealing
SDSS	Spatial Decision Support System
SFLA	Shuffled Frog Leaping Algorithm

SGML	Standard Generalized Markup
SMGA	Structured, Messy Genetic Algorithm
SMP	Symmetric Multi-Processing
SOGA	Single Objective Genetic Algorithm
STL	Standard Template Library
SWMM	Storm Water Management Model
TCP	Transmission Control Protocol
THV	Throttle Valve
UDP	User Datagram Protocol
UML	Unified Modelling Language
WAN	Wide Area Network
WAP	Wireless Access Point
WDS	Water Distribution Systems
XML	eXtensible Markup Language
XOR	eXclusive OR *(logical operation)*

Chapter 1. Introduction

1.1. Background

Automated analysis and optimization tools have been used to provide a mechanism for improving various facets of water system networks including, amongst others, model calibration, network design and rehabilitation, leakage detection and pump scheduling. Such tools represent an attempt to provide assistance to practitioners through the means of intelligent, knowledge-based techniques. These Decision Support Systems (DSS) encompass a wide-range of computer-enabled applications that are conventionally based on some form of analytical model, coupled to some form of optimization. The application of optimization to Water Distribution Systems is generally characterised, however, by extended runtimes owing to the computational load imposed by the numerical solution of a hydraulic network model in order to determine the pressures and flows throughout the system. This solution is iterative in nature and the time taken to compute the solution is largely dependent on the size and configuration of the network model itself. Evolution Algorithms, which, by their nature require large numbers of evaluations of an objective function would, on first sight, appear to be ill-suited to Water Distribution System applications. However, their ability to converge rapidly on an optimal or near-optimal solution, whilst having analysed a mere fraction of the total solution space, has made such algorithms popular subjects for research. Despite inexorable improvements in computer power, there is still a need to improve the performance of these algorithms to allow for more complex optimizations to be undertaken with acceptable efficiency and effectiveness. If multiple settings of the hydraulic network are to be considered, for example to allow an optimization to take account of variable network conditions during a 24-hour period or to allow for conditions of uncertainty to be evaluated, the computational workload associated with this hydraulic simulation becomes even more significant.

A software framework for implementing Genetic Algorithms for hydroinformatic applications is presented. This framework employs object-oriented programming techniques to provide a flexible, extensible system for implementing single and multiple-objective algorithms. The framework developed has been deployed in a number of research projects as well as commercial undertakings and has been adopted to provide optimization facilities to three commercial products from two vendors.

1.2. Aims of Research

The research presented in this thesis builds on previous work in the Centre for Water Systems on evolution-based algorithms by introducing a framework for the development of such methodologies. This thesis presents novel approaches for simplifying the deployment of optimization techniques in hydroinformatic applications through the introduction of a componentized methodology that can be extended to implement new evolutionary algorithms with a minimal requirement for additional development work.

Relative to the implementation of the optimization algorithm itself, the operation of a hydraulic network solver incurs significantly greater computation overhead. To this end, novel approaches are proposed in the thesis to accommodate this issue from two perspectives. Firstly, novel approaches for improving the algorithmic efficiency of the evolutionary optimization algorithms themselves are explored. This includes investigations of new modifications to the representation of the genetic material employed and the operators that act on it in order to promote the efficient convergence of the population to an optimal solution. Other considerations include ensuring that the algorithms minimize wastage by avoiding the evaluation of solutions that have already been considered. The acceleration of stochastic optimization techniques is of particular importance, given the even greater runtimes common with such optimizations. The second approach undertaken is to improve the operational efficiency of the algorithms. This is achieved, herein, through the development of a methodology for massively parallelizing the evaluation of hydraulic network simulations by employing a cooperating network of computers.

1.3. Objectives

The following objectives have been formulated

- Evaluate the effectiveness and relative performance of alternative genetic representations for chromosomes in evolution algorithms with respect to runtime and solution quality considerations.
- Assess the potential for advanced caching and archiving techniques to reduce the runtime of evolution algorithms.
- Develop and implement a framework for distributed evaluation of hydraulic network simulation and determine its value for facilitating the massive parallelization and acceleration of evolution algorithms for the optimization of water distribution networks.

- Develop and implement a modelling architecture for representing connected hydraulic networks to assist in the above objectives.

1.4. Thesis Structure

This thesis is arranged in nine chapters with two supplementary appendices. Following this introduction and statement of the thesis aims and objectives, this thesis adopts the following structure:

The second chapter, the literature review, provides a background to the application of optimization techniques, with particular emphasis on evolutionary approaches, to the optimization of Water Distribution Systems.

Chapter Three introduces the concept of Evolution Algorithms (EAs) and relates the design, implementation and continuing development of a methodology for the application of EAs to hydroinformatic optimization problems.

Novel extensions to the classical implementations of these algorithms are presented in the Fourth Chapter with the aim of improving algorithm performance – both in terms of execution speed and quality of result.

Chapter Five presents a new methodology for the distribution of the computational workload associated with optimization applications that involve hydraulic network simulation between computers connected by a Local Area Network (LAN) or on multiprocessor/multicore computers.

Chapter Six demonstrates the applicability of the techniques introduced in the prior chapters through their application to a number of small-scale single-objective optimization problems from the literature.

This analysis is extended in Chapter Seven where the small-scale problems are refactored as multiobjective optimization problems. Having demonstrated the effectiveness of the novel techniques on small-scale networks, the analysis is concluded with the application of the techniques to a more computationally demanding network optimization problem.

Chapter Eight reapplies the methodologies presented to more complex problems – computationally and algorithmically – to demonstrate the wider applicability of the research to optimization of hydroinformatic problems.

The final chapter details the conclusions that can be drawn from this research and the proposed methodologies and suggest further avenues of research.

The first Appendix introduces a software component, OpenNet, used for the modelling of networks. This library implements a generic system for representing connected networks and for undertaking analysis upon them. A specialization for representing pressurized hydraulic networks is presented along with a generic, adaptable technique for representing networks using eXtensible Markup Language (XML) the specification of which can be found in the second appendix. The research is supported by the application of these tools to a number of practical applications. Those described in the first appendix include the translation of third-party hydraulic networks from one format to another and integrating hydraulic models with disparate GIS data sources.

Chapter 2. Optimization in Water Distribution Systems

2.1. Literature Review

2.1.1. Genetic Algorithms

Genetic Algorithms (GAs) are part of a group of stochastic optimization techniques called Evolutionary Optimization, inspired by Darwinian theories of natural selection. This class of algorithm is noted for its ability to tackle large, NP-hard (Templeman, 1982) optimization problems without any domain-specific configuration. Such NP-hard problems are those which are difficult to solve in polynomial time. Holland (1975) was the first to coin the term 'genetic algorithm' and identified the mathematical basis for the operation of the algorithms in terms of schema theory and the basis for the selection and recombination of genetic material, chromosomes, representing problem solutions. The implementation and techniques underpinning GAs are described in more detail in Chapter 3. Holland's GAs implement the archetypal chromosomal representation which use strings of binary digits (bits) to encode the genotype of a solution.

A number of workers in the field have investigated the extension of GAs to operate on multiple objectives simultaneously. Instead of allowing a population of individual chromosomes to converge to a single solution, a multiple objective algorithm maintains multiple trade-off solutions for two or more objectives. An early description of a conceptual multiple-objective algorithm by Goldberg (1989) was followed by functional algorithms including Non-dominated Sorted GA (NSGA) (Srinivas & Deb 1994) and multiple-objective GA (Fonseca & Fleming, 1993). Deb (2001) describes the substantially reworked NSGA-II algorithm, which forms the basis of the robust multiple-objective optimization developed by Kapelan *et al.* (2005) and subsequent developments described in Chapter 4.5.

2.1.1.1. Pump Optimization

One of the first applications of Genetic Algorithms to Water Distribution System (WDS) optimization is reported by Goldberg and Kuo, (1987). This work demonstrates the efficacy of applying the GA methodology to pipeline optimization problems. In this case, the optimization is applied to a 40 pump, serial pipeline and seeks to minimize the cost of pumping (in terms of power consumption) whilst meeting constraints of maximum discharge pressure and maximum and minimum suction pressure. The GA is implemented in terms of a simple binary string and employs single-point crossover and simple mutation (see 3.2.1.3).

The results are compared with the optimal solution determined using mixed-integer programming and the GA is found to have performed well, having achieved within 0.72% of the optimal solution on average whilst exploring a minute fraction of the search space of size 1.1×10^{12}.

2.1.1.2. Network Design and Rehabilitation

Murphy *et al.* (1993) present one of the earliest applications of a GA to a real-world WDS optimization problem. In this paper, they demonstrate the ability of a GA to produce an optimal design layout for a new housing development using discrete, commercially available pipe diameters. A theme returned to by Simpson *et al.* (1994) who review the alternatives for the optimization of water distribution systems, comparing "traditional" Linear Programming (LP) approaches with non-linear techniques and the, emergent GAs. Of particular interest is the ability of the GA to select discrete, rather than the continuous variable outputs of the LP technique – obviating the need to convert the solutions obtained by the algorithm into a commercially feasible installation.

Simpson *et al.* (1994) describe one of the first applications of GAs to a conventional supply system. Here the GA is applied to the combined network rehabilitation and design problem introduced by Gessler (1985). In this problem, a small network (1 reservoir, 1 tank and nine demand nodes) is to be upgraded through the provision of new pipes and the duplication or cleaning or others – minimising the cost of implementation whilst meeting a minimum pressure criterion at each node. A conventional binary string representation is employed with three bits being used to represent the options available for each pipe. In the case of the duplicated/cleaned pipes, this value represents either a decision to clean the pipe or the diameter of the replacement. For the new-build pipes, this value represents the diameter of the new pipe. The results obtained were compared to those derived from an exhaustive simulation of the possible solutions – thus, it is possible to compare the GAs performance against the global optimum for the problem. The GA as formulated using a single-point crossover and simple by gene mutation arrangement is shown to achieve the global optimum on the majority of its runs despite exploring between 0.1% and 0.15% of the solution space (16,777,216 possible solutions). Further to this comparison, a Non-Linear Programming (NLP) approach was also employed. This method uses gradient techniques to refine a single solution to a problem, producing continuous results for the decision variables. In this example, the decisions need to be made in terms of discrete, commercially available pipe diameters and, consequently, the solutions obtained from the NLP algorithm need to be

recast into discrete terms by rounding the continuous pipe diameters obtained to their nearest discrete equivalent. The NLP output is shown to be significantly inferior to that of the GA – indeed the continuous variable results obtained are inferior to the GA even prior to being rounded into discrete diameters.

Gupta *et al.* (1999) present an unconventional GA implementation for WDS design that does away with the normal chromosomal representation used by GAs. Instead of employing binary strings to represent discrete pipe diameters, this work substitutes the actual diameters into the chromosome instead – ostensibly to avoid the encode/decode cycle associated with the use of binary strings. It should be noted, however, that the overhead for maintenance of the binary string is trivial in comparison to the computational requirements of performing a hydraulic simulation and this improvement, of itself, is unlikely to be significant in substantially reducing the algorithm runtimes. The GA itself is heavily influenced by a number of heuristic modifications, including the initial stratification of the network into different diameter groups using expert judgement. The operation of the GA is steered through further iterative routines, which operate on a candidate solution to promote feasible solutions in the population. The results for six related case study scenarios are contrasted with those obtained from the authors' own WATDIS software which employs NLP and which is shown to be marginally inferior for all but one of the scenarios. It is noted, however, that both the GA and the NLP software require several trials to identify near-optimal solutions. In the case of the GA this is to accommodate different scenarios of initial stratification for the pipe diameter ranges whilst the progress of the NLP software is heavily dependent on the initial conditions as it has a tendency to identify local optima.

Savić & Walters(1997) introduce the GANET software and apply it to three previously published case studies in design and rehabilitation of WDS – New York Tunnels, Hanoi (Fujiwara & Kang, 1990) and Alperovits & Shamir's example network (1977). A comprehensive comparison of the results obtained in previous work, using various optimization techniques, is presented and illustrates the sensitivity of such analysis to the minor variation of some modelling parameters between different researchers. In particular, variation in the constants used to derive the Hazen-Williams coefficient (see Walski, 1984) is highlighted as being a cause for marked differences in the predicted hydraulic behaviour of the network. Such differences in behaviour are shown to have consequences for the steering of the optimization algorithms employed and the comparability of the results obtained: a solution to the New York Tunnels problem is given which at $37.13m betters the current

known optimal solution – achieved by relaxing the Hazen-Williams coefficient. The GANET tool, which combines an optimizing application coupled to a public domain hydraulic solver, EPANET (Rossman, 1993) which employs the Gradient technique (Todini and Pilati, 1987) for evaluating the hydraulic performance of networks.

The integration of WDS analysis with commercial optimizers is the focus of Lippai *et al.* (1999). They report the results of solving the New York Tunnels problem (Schaake & Lai, 1969) with WinPipes (an EPANET-derived hydraulic solver) coupled to four commercial optimization applications including, GA-based Evolver (Palisade Corp., 1998) and GENOCOP (Michalewicz, 1992). Interestingly, they report a solution for New York Tunnels, obtained with Evolver, of $38.13m – which would be the best result obtained to date were it feasible. Running the proposed solution through EPANET reveals an aggregate head deficit of 0.03psi. This may be related to the sensitivity to the model to small changes in the Hazen-Williams coefficient, as noted by Savić and Walters (1997) or through some rounding error.

The Messy Genetic Algorithm (MGA) of Goldberg *et al.* (1989) differs from a conventional GA in that the MGA operates with variable chromosome lengths, allowing the algorithm to progressively build up a solution as it runs – constraining the search space encountered. In this instance, integer-coded genes are employed. An enhancement to the MGA is described by Halhal *et al.* (1997) with particular application to the rehabilitation of WDS. They propose a modified algorithm termed Structured, Messy Genetic Algorithm (SMGA). The SMGA extends the approach of the MGA by employing an initial population, each member of which contains each single decision element. For example, in the small rehabilitation problem presented in this paper, there are 8 possible decisions to be made on each of 15 pipes which would lead to the SMGA having a starting population of $8 \times 15 = 120$ individuals, representing each possible decision using one intervention. Furthermore, the authors describe one of the first applications of multiobjective optimization to WDS problems. With this technique, Pareto-optimal ranking and fitness sharing Goldberg & Richardson (1987) is used to coordinate the retention of individuals in the population according to how well they fit the twin objectives of minimizing cost and maximizing benefit. In contrast to the other techniques employed, such a multiobjective optimization produces a Pareto-curve illustrating the trade off between the objectives rather than a single "solution" to the problem. The SMGA is shown to perform far better than a conventional GA for the presented problems, producing a better classification of individuals along the Pareto front

than the standard GA. This technique is later applied to the "Anytown" WDS benchmark system (Walski *et al.*, 1987) by Walters *et al.* (1999). This problem is a network reinforcement for increasing demand scenarios, which is constrained not only by infrastructure but also by pumping costs. The SMGA as presented by the authors is seen to improve the then best published result for this benchmark by between 4 and 5%.

Wu and Simpson (2001) seek to optimize the same network arrangement solved by Simpson *et al.* (1994) using a similar MGA. In addition, they demonstrate an improved performance, in terms of GA convergence to near-optimal solutions, for the MGA on the New York Tunnels problem over that achieved by the "Improved GA" of Dandy *et al.* (1996).

Dandy and Engelhart (2001) relate the use of GAs to optimize pipe-replacement schedules for single and multiple time-horizons. The algorithm described includes the use of a hybrid selection scheme in which Tournament selection (Goldberg *et al.*, 1991) is combined with the conventional Roulette-wheel technique (Holland, 1975). In addition, Uniform crossover (Syswerda, 1989) and the creep mutation of Dandy *et al.* (1996) were employed to operate on an integer-coded chromosome. The authors demonstrate that this approach employed on a problem with a large solution space ($\sim 1 \times 10^{100}$) produces good results by identifying pipes requiring replacement, within the required budget. The results were validated by comparison with a simplified asset model for which decisions were made on a case-by-case basis.

Kadu *et al.* (2008) present a modified Genetic Algorithm for undertaking optimal design of WDS employing techniques to reduce the optimization search space. A real-coded chromosome is employed, along with single-point, uniform and multi-parent crossover, non-uniform and neighbour mutation (amongst others) coupled with a Critical Path technique (Bhave, 1978) to reduce the search space -the different genetic operators being selected at random during the operation of the algorithm. The algorithm is demonstrated on a number of familiar networks from the literature including the Hanoi network where the algorithm is shown to match the best known result (Cunha & Sousa, 1999) and to better the result achieved for the stricter problem introduced by Savić & Walters (1997) where $\omega = 10.9031$. For performing the hydraulic analysis, the authors introduce GRA-NET a hydraulic solver which, like EPANET, is based on the Gradient Method (Todini & Pilati, 1987) but which allows the easy modification of the coefficient (ω) and exponents (α,β) used in the Hazen-Williams head loss equation:

$$h_f = \omega \cdot \frac{L}{C^{\alpha} \cdot D^{\beta}} \cdot Q^{\alpha} \qquad \text{i)}$$

where h_f is the head loss, L is the length of the pipe, Q is the flow through the pipe and C and D are the friction factor and diameter of the pipe respectively.

The sensitivity of GAs to the genetic representation used to represent the problem and its interaction with the recombination operators of crossover and mutation is demonstrated by Dandy *et al.* (1996). Specifically, they investigate the use of Gray coding to improve the performance of crossover in preserving schema in the chromosomes. A variable power-scaling fitness function is employed to direct the search as the optimization progresses. In addition, a novel mutation operator is introduced which permutes individual genes into adjacent values, rather than the traditional GA approach of randomizing gene values. As a case study, they apply these modifications to the New York Tunnels problem (Schaake & Lai, 1969) and report the best result obtained by a Genetic Algorithm of $38.80m for a fully feasible solution using discrete pipe diameter selections. The results obtained mark a considerable improvement in both computation performance and solution quality over the authors' previously work (Murphy *et al.* 1993).

2.1.1.3. Network Calibration

Since their introduction into the domain of hydroinformatics, GAs have been used for calibrating WDS – modifying system parameters, ordinarily pipe roughnesses in order to match model results with data obtained from the field (e.g. Savić & Walters, 1995). Vitkovsky and Simpson (1997) present a comprehensive analysis of the application of GAs to calibration, both for fitting roughness values to pipes and for transient calibration. The authors use real numbers encoded as binary strings as the basis for a conventional GA approach. The operation of an averaging crossover operator is improved by the occasional use of two-child average operators, which are seen to reduce the likelihood of a population of individuals converging prematurely.

de Schaetzen *et al.* (2000) utilise the GA in a different fashion for calibration: attempting to find the optimal arrangement and density of sampling points for the analysis. The optimization here is formulated in terms of maximizing an entropy function, employing an integer-based chromosome with each gene representing a potential sampling point location. The results are compared to those produced through expert judgement and are

found to be a useful tool for deriving a likely set of candidate sampling points. A similar approach is adopted by Meier and Barkdoll (2000) where the objective function is to realize the maximization of the length of pipes in the network that have non-negligible flows when the sampling is being performed for the proposed sampling point distribution. Kapelan *et al.* (2003b) further extend this analysis by formulating a multi-objective GA based on Fonseca & Fleming's implementation (1993) for considering calibration-sampling design.

2.1.1.4. Water Quality Optimization

Genetic Algorithms are used by Munavalli and Kumar (2003) to optimize the rate, timing and concentration of chlorine dosing in a WDS. Given predefined dosing locations, this methodology seeks to optimize chlorine dosing so as to minimise the maximum concentration found in the network whilst continuing to maintain the minimum level of chlorine residuals at all nodes in the network. Using hydraulic results from EPANET (Rossman, 2000) for extended period simulation of the network in question, the authors apply their quality model for chlorine decay. The GA employed uses a conventional binary string implementation but adds creep mutation, in which mutation permutes the variable by the smallest possible amount in a given direction, and a niching operator (multidimensional phenotypic sharing scheme - Goldberg 1989) to direct further the search.

2.1.1.5. Accommodating uncertainty in GAs

Kapelan *et al.* (2003a) report a technique for accommodating uncertainty in design constraints when optimizing using GAs. This approach embeds a stochastic optimization cycle within the operation of a conventional single or multiple-objective GA. The stochastic cycle evaluates samples for Probability Density Functions (PDFs) obtained for the stochastic variables (in this example, uncertain future demands) and aggregates statistics on the network performance over the lifetime of the chromosome – providing a measure of reliability of the network under the uncertain constraints.

Babayan *et al.* (2003) and Kapelan *et al.* (2004) present differing approaches to the optimization of WDS design/rehabilitation under conditions of uncertainty. The former approach employs the reformulation of the stochastic problem in deterministic terms. This is accomplished through some simplification of the problem and the use of numerical methods in the quantification of the uncertainty in hydraulic reliability. By comparison, the latter promotes a sampling-based technique for accommodating uncertainty that is

independent of the model under consideration. Both of these methodologies employ single objective GAs to perform the optimization.

2.1.2. Other Optimization Techniques

2.1.2.1. Linear Programming

One of the earliest efforts at computer-aided water distribution system (WDS) design was described by Alperovits & Shamir (1977) who developed a Linear Programming Gradient methodology for a least-cost implementation of a network. In this context, "least-cost" refers to the minimization of the implementation cost of the network in terms of the cost of the pipes to be installed. This methodology reduces the optimization to a series of sub-problems regarding the possible flow routes to each point in the network. The applicability of this approach is severely constrained by the complexity of the network involved and subsequent work has concentrated on reducing the computation complexity of optimizing such networks.

Morgan & Goulter (1985), Taher & Labadie (1996) formulate their design optimization as a Linear Programming problem for which a componentized software package was developed comprising a number of individual applications working in concert. These papers, and several others, suggest a common theme - the direct integration of the network optimization software with spatial information – in this instance, the spatial component is used to perform network analysis for pressure zone distribution, node demand allocation and least-cost routing.

2.1.2.2. Heuristic Approaches

Heuristic techniques are employed in situations where classical optimization techniques would otherwise struggle to achieve good results with acceptable runtimes. This is often achieved through the embedding into the optimization of some domain-specific knowledge, which can be used to steer the process more effectively. Makropoulos *et al.* (2003) employ a random search technique to resolve a spatial optimization problem obtained from the aggregation of water demand management scenarios developed through fuzzy inference rules. This work seeks to produce optimum strategies for maximizing water-saving whilst respecting investment constraints. The application of such a heuristic, as with many optimization strategies, is not guaranteed to produce a global-optimum and the authors validate the results obtained through a Monte-Carlo sampling approach. This technique is

expanded upon by Makropoulos & Butler (2005) where a heuristic approach is hybridised with a more conventional multiple-objective evolution algorithm for application to wider water sustainability issues.

2.1.2.3. Cellular Automata

Cellular Automata are a long established area of research in computer science having been first identified by John von Neumann and Stanisław Ulam whilst working at Los Alamos National Laboratory, New Mexico, U.S.A. (von Neumann, 1966). They have been successfully employed as an optimization technique for Water Distribution Systems by Keedwell & Khu (2006) in which the nodes and pipes of a network are configured to communicate with each other, according to predefined rules, in such a fashion that a self-optimizing behaviour emerges in the network. Keedwell and Khu (2006) employ three rules to direct this behaviour:

- If a demand node is pressure deficient then it requests connected pipes that *supply* it to be upsized.
- Conversely, if a demand node has a pressure surplus then it attempts to downsize its supply pipes.
- If a pipe receives an equal number of upsize and downsize requests it responds by upsizing.

The emergent behaviour that these rules embody makes no attempt to find a global optimum as it is driven purely by pressure differentials rather than other, conventional optimization factors such as implementation cost. Instead, it seeks a stable solution that it can achieve by making local changes to the network. This technique is of particular interest as it is computationally very efficient with respect to other optimization techniques and can be seen as a effective form of local-search heuristic. The authors suggest that it is well suited to the role of generating initial "seed" populations for other optimization techniques where the solution space for a given network problem is inordinately large.

2.1.2.4. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is introduced by Eberhart and Kennedy (1995). Like GAs, this technique operates on a population of individuals that is, initially, randomly generated. However, the mechanisms for improving the fitness of the population are quite different to those of evolutionary algorithms. Here, individuals in the population use

retained information about the best solution they themselves have encountered in the solution hyperspace and, combined with the "group knowledge" of the global best solution encountered, the individual solutions are perturbed in the directions of the local and global optima according to some random term. The Shuffled Frog Leaping Algorithm (SFLA) is related by Eusuff and Lansey (2003a,b), a memetic, meta-heuristic technique which combines a novel evolutionary technique with the PSO (Eberhart and Kennedy, 1995) to facilitate the local search element. Once more, the New York Tunnels (NYT) example is employed, with EPANET providing the hydraulic computation, with the authors contrasting their results with those obtained by several others in the field – including the seemingly infeasible result of Lippai *et al.* (1999) and those obtained by Savić & Walters (1997) through the variation of the Hazen-Williams coefficient. In Eusuff and Lansey (2003a), the SFLA algorithm produces a result for the NYT expansion problem of $35.27m. It is unclear from the paper as to how this result is obtained as the solution presented is infeasible when solved with EPANET (an aggregate head deficit of 7.27psi over three nodes) and remains infeasible even with the relaxed Hazen Williams constraint of Savić and Walters (-6.36psi over the same three nodes). Eusuff and Lansey (2003b), however, report modified results in which SFLA's best result is shown to be equal to that of Lippai *et al.* (1999) using a modified Hazen-Williams coefficient of 10.667.

2.1.2.5. Simulated Annealing

Simulated Annealing is an optimization technique which, in essence, applies the mutation operator familiar to the Genetic Algorithm to a single solution repeatedly. Initially, a high "temperature" allows the mutation to vary widely the values of the decision variables. As the "temperature" cools, i.e. during the progress of the optimization, the freedom of the mutation to vary the values is constrained – as an analogue with metallurgical annealing in which crystalline solids begin to appear during cooling.

Cunha & Sousa (1999) describe the application of the Simulated Annealing technique to WDS benchmark problems including one introduced by Alperovits & Shamir (1977) and the Hanoi network of Fujiwara & Kang (1990). The application is successful in finding low-cost solutions and has identified the lowest-cost solution for the Hanoi problem yet published of $6,056,370.68 – albeit with a relaxed Hazen Williams constraint.

2.1.2.6. Ant Colony Simulation

The Ant Colony Simulation (ACS) approach was introduced by Dorigo *et al.* (1996) and first applied to the domain of WDS optimization by Maier *et al.* (2003) who used this technique to tackle a number of benchmark rehabilitation problems. This work is notable, in particular, for finding the best-known solution, at the time of writing, for the New York Tunnels optimization problem introduced by Schaake & Lai (1969) and described in detail in Chapter 6.2. ACS operates as an analogue of the essentially random process of ants foraging for food in which individual ants lay pheromone trails as they explore. In the optimization technique, there is a higher probability of an ant following an existing pheromone trail that it encounters of a given threshold strength – resulting in a positive feedback mechanism which allows the "ants" to identify the most direct route to the food source.

Several variants of the approach have emerged. Following on from earlier work (Zecchin *et al.*, 2006), Zecchin *et al.* (2007) present a comparative study of the performance of five ACS algorithms applied to Water Distribution System benchmarks and find that the Elitist-Rank Ant System (Bullnheimer *et al.*, 1999) and the Max-Min Ant System (Stützle and Hoos, 2000) outperform the other types.

2.2. Summary

Evolution algorithms are no longer seen as an emergent technology in the field of Water Distribution System optimization. The traditional techniques founded on linear and non-linear programming have largely been usurped and new metaheuristic techniques have taken their place. These techniques, Genetic Algorithms, Ant Colony Optimization, Particle Swarm Optimization amongst their number, are being employed for an increasingly diverse range of applications in the field. The classical applications of network design and rehabilitation and model calibration have been supplemented with more complex and larger-scale problems. Hybrid optimization techniques are increasingly being employed to accommodate additional concepts to improve the reliability of optimized solutions – particularly by considering uncertainty in design criteria.

Chapter 3. Genetic Algorithms

3.1. Introduction

Evolution programs (Michalewicz, 1992), of which Genetic Algorithms (GAs) are probably the best-known types, are general, artificial evolution search methods based on natural selection and mechanisms of population genetics. They emulate the natural processes of evolution (Darwin, 1859)– being based on preferential survival and reproduction of the fittest members of the population, the maintenance of a population with diverse members, the inheritance of genetic information from parents and the occasional mutation of genes. These algorithms are best suited to solving combinatorial optimization problems that cannot be solved practicably using more conventional operational-research methods. Thus, they are often applied to large, complex problems that are non-linear with multiple local optima.

GA optimization is a powerful approach with a proven ability to identify near-optimal solutions (Savić & Walters, 1994; Halhal *et al.,* 1999). However, the correct operation of a GA optimization depends on careful configuration and parameter tuning - requiring appropriate skills and experience. Inappropriate penalty levels will distort the results away from the end user's perception of 'optimum.' Too high or low a rate of genetic interchange ('crossover' and 'mutation') result in degeneration to a random search or stagnation, causing a failure to converge on the *global* optimum. Because of the complexity involved in setting up a GA to operate effectively, this form of optimization is not well suited to 'trivial' problems, i.e. those for which the number of possible solutions is small.

Solving optimization problems related to water distribution networks is recognized as an NP-hard analysis that has conventionally been approached using a number of techniques including hill climbing, linear and dynamic programming. Evolution algorithms represent a proven, alternative strategy for approaching these problems.

The benefits of GAs stem from their ability to converge rapidly on an optimal or near-optimal solution, having analysed only a tiny fraction of the number of possible solutions available. For large problems such as those typically associated with networks, the exhaustive analysis of all options is unlikely ever to be feasible. In a water distribution network expansion and reinforcement problem (Atkinson *et al.*, 1998) analysed in the Centre for Water Systems, University of Exeter, using GAs, there were approximately 300 pipe links, each of which could adopt any of 14 diameters (including the option of not employing a pipe). This presents a problem size of 14^{300} or 6.89×10^{343}. Even if one billion design

evaluations (i.e. network simulation runs) could be performed in a second, the time needed to evaluate all possible schemes would be much longer than the age of the earth (estimated at 4.6 billion years). Given that, on current high-end PC platforms, design evaluations for multiple time-steps may take up to several seconds each, such iterative analyses are clearly redundant. In this instance, the GA optimization search was able to converge on the lowest cost solutions by carrying out several hundred thousand evaluations.

3.2. Methodology

The basic genetic algorithm is implemented by means of selecting a number of organisms from a population; recombining them in some fashion to produce a number of offspring; introducing some mutation factor; evaluating the resultant offspring with respect to their fitness as solutions for the problem at hand and finally reintroducing (or replacing) the organisms into the base population. The flowchart in Figure 3-1 illustrates the basic operation of a Genetic Algorithm through these repeated cycles of Selection, Recombination (crossover), Mutation and Replacement.

In nature, the evolution of biological organisms takes place as a result of the adaptive pressure exerted upon them by the environment in which they have to survive. Through the sexual competition between individuals, it happens that the strongest and most attractive individuals win the right to mate and to produce offspring.

The adaptation of organisms is ordinarily considered to take place through the production of offspring. Natural organisms, in general, produce numerous offspring, many of which will not survive. These offspring have varied traits inherited genetically from their parents. Natural selection acts in such a fashion that the offspring with the most useful traits for their environment are the most likely to survive and, hence, to perpetuate those traits.

As with its natural analogue, a GA operates on a population of individuals that can be seen as representations of potential solutions to a given problem. In nature, this problem is survival and procreation - in GAs it can be any problem for which there is a means to determine a solution's fitness (suitability). This means is commonly described as an individual's objective function.

3.2.1. Algorithm operation

Genetic algorithms search for schemata (Holland, 1975), which result in better fitness scores. Schemata can be considered building blocks An evaluation is implicitly processing a number

of schemata in parallel. Therefore, each time the genetic information from an individual organism is evaluated by the algorithm, instead of merely sampling a single point in the solution space, the algorithm is sampling many simultaneously. Holland finds that the relationship between population size (P) and the number of schema implicitly processed by each generation of that population is of the order of P^3.

It is also necessary to consider at what stage the operation of an algorithm should be terminated. Although the algorithm can be executed for a given number of iterations or run for a specific length of time, it is more useful to examine some measure of the genetic diversity of the population. This is normally done by some statistical analysis of the variance in the fitness of the organisms in the population - since if a population is composed of identical, or near-identical organisms, the prospects for advancement for the population are poor.

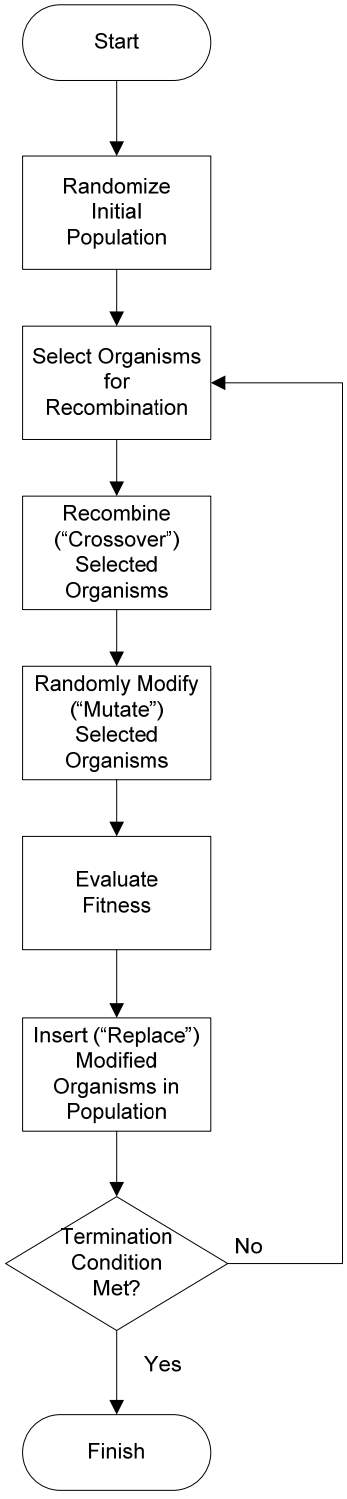

Figure 3-1: Flowchart illustrating basic Genetic Algorithm operation.

3.2.1.1. Algorithm types

There are three conventional methods for organising the operation of a basic single-objective genetic algorithm.

1. In a steady state algorithm, (Holland, 1975) a number of individuals (normally two) are selected for recombination. Once generated, their offspring are, in some fashion, incorporated into the existing population - provided they are of sufficient fitness.
2. The generational scheme (Goldberg, 1989) is markedly different in that the offspring borne of parents selected from the population are pooled in a new population. This process of selecting from one population and storing the offspring in another is ordinarily repeated until the size of the offspring's population is the same as that of the original population. At this time, the inhabitants of the original population are eliminated and the offspring's population becomes that from which selection takes place.
3. Adding elitism (Goldberg *et al.*, 1991) to the generation GA implementation involves preserving the best organisms from the original population and transferring them, unmodified, into the new population. The degree of elitism can be controlled by varying the number of organisms copied between the populations.

Extending GAs into the multiple objective domain was first undertaken by Fonseca & Fleming (1993) who formulated a Multiple Objective GA (MOGA) employing fitness ranking to produce a trade-off between competing objectives. Further multiple objective implementations have been proposed, such as the NSGA-II by Deb (2001) which has the advantage of being largely self-tuning. Both methodologies have been incorporated into the library described herein.

3.2.1.2. Selection

Although, conventionally, a GA does not have a concept of gender in its representation of organisms, it still possesses the notion of sexual selection in the initial selection phase of operation in that organisms are competing against each other for the perquisite of propagating themselves.

The process of selection determines, in some fashion, which of the individuals will have some or all of their genetic material passed on to the next generation. The selection scheme used by a GA seeks to give exponentially increasing selection to the fittest organisms in the population, thus differentiating a GA from a random search technique.

The "roulette-wheel" or proportional selection introduced by Holland (1975) and refined by Goldberg (1989) is amongst the most common means of performing selection. It is best visualised as a roulette wheel in which each slot on the wheel represents an organism in the population. The width of the slot is proportional to the fitness of the corresponding organism and thus the 'ball' is more likely to favour the fitter individuals in the population. Conventionally, the implementation of the roulette wheel relies on the objective function being configured for a maximization problem. It is a small matter to ensure that the function correctly configures the roulette wheel for the minimization problems that are more common in the water industry. A similar selection technique is proposed by Baker (1985) in which the basis of the proportional selection is the rank an organism holds, in terms of fitness, within a population.

Tournament selection (Goldberg *et al.*, 1991) can be thought of as a form of ranked selection as above. Instead of operating on the population as a whole, it functions by selecting at random a number of individuals from the population and then comparing their fitness. The fittest individual goes forward from the tournament to be one of the contributing individuals to the recombination process. The tournament is repeated until enough individuals have been selected to perform the recombination. One of the useful attributes of tournament selection is that the selection pressure can be tuned easily by modifying the size of the tournament: a smaller tournament promotes the likelihood that a weaker member from the population will be selected.

3.2.1.3. Recombination

A GA seeks to evolve fitter solutions through some means of selective recombination of parts of better, existing solutions. This mimics the "survival of the fittest" stratagem of natural evolution in which the "fitter" solutions are predisposed to be more successful at reproduction. This tends to lead to the removal of the less-fit individuals and an increase in the fitness of the population as a whole. In Genetic Algorithms, this recombinative operation is known as "crossover".

Crossover, like selection, is one of the key traits of a Genetic Algorithm. Ordinarily, it takes place on two organisms selected from the population and, conventionally, produces two offspring organisms - although this is not necessarily the case. Each child is likely to be different from its parents, unless its parents are identical, and yet they will each retain a number of characteristics from their parents. Given that both parents are likely to have high

fitness, since they prevailed in the selection process, there is a reasonable probability that the one or both of the children may prove to be more fit than either parent is.

The most commonly used crossover methodologies for standard GAs are single and two point crossovers in which one or two loci are selected on the parent gene and the genes between the end and the locus (single point) or between the loci (two point) are transposed to produce the two offspring. Syswerda (1989) introduced Uniform crossover in which the two children are produced by selecting at random, for each gene locus, from which parent the gene value should be copied. As with the other methodologies, the two resulting children retain all of the genetic information from the parents – it having been transferred in part to one child or the other.

3.2.1.4. Mutation

To ensure that the solutions in a population do not become stuck at a non-optimal solution, a randomization element is introduced. This stochastic alteration is known in GAs as "mutation". The purpose of the mutation component of the algorithm is to permit local search around a given solution. Various forms of mutation are commonly used, varying from those which permute individual bits of binary strings, randomize variables with their domain or adjacency operators that "nudge" gene values to adjacent values.

3.2.1.5. Replacement

After crossover, and potentially mutation, the final stage of the GA operation is Replacement. This operation applies primarily to Steady State GAs, since generational GAs populate wholly new populations, to form each successive generation - making the application of a replacement strategy irrelevant. Instead, the children produced in a Generational GA, following recombination, are automatically promoted to the new population. Such GAs tend to promote genetic diversity as a result.

The newly formed offspring have to be incorporated into the existing population in some fashion. There are many potential strategies, including substitution of parents, replacement of the weakest member of the population, etc. The choice of replacement strategy is dictated considerably by the nature of the algorithm being employed, be it steady state or generational and by the solution space itself.

Rather than explicitly control the replacement strategy, some GA implementations seek to place a measure of control on it by introducing a probability factor on the crossover

operation. This means that the parents pass through the crossover stage unaltered and, unless mutated, will find their way back into the population unaltered.

A number of mechanisms can be used to implement this replacement:

Replacement of Parents

This mechanism is, perhaps, closest to the biological analogue. Traditionally, organisms in GAs do not have a concept of age. However, by causing children to replace their parents this can be simulated to an extent. Ordinarily, this would only be done if they prove to be fitter than their parents are.

Replace Weakest

In this strategy, both children are added to the population and then the two weakest organisms are removed. This has the effect that if either or both of the children are weaker than the weakest existing member of the population then they will be eliminated. If they are stronger then an existing member is sacrificed, instead. This mechanism has the effect of rapidly excluding the weakest members of the population whilst converging on the very strongest members.

Replace First Weaker

Each child is compared with the existing members of the population until a weaker organism is encounters, whose place it then takes in the population. In the event that no weaker organisms are found (i.e. that the child is weaker than the weakest member) then that child becomes extinct. This promotes less rapid convergence, being less destructive than the Replace Weakest strategy with respect to the weaker members who may still retain valuable genetic information.

Replace By Rank

The 'Replace by Rank' mechanism operates by ranking the organisms within the population according to their fitness values. The newly produced children then replace the existing organism which has the rank that they would have, had they been part of the population. If there is no such rank in the population then they are eliminated.

3.2.2. Solution Representation

In order for a GA to function there must be some mechanism to allow the computation of the fitness of an individual for the problem at hand. The term genotype refers to the genetic

information stored in the *genome* of an individual *organism*. In order to make use of this information, it has to be decoded into the phenotype – the "real-world" interpretation of this information, if you will. The decoding process used depends on the representation used for the genotype.

3.2.2.1. Genotype Representation (Encoding)

Conventionally, Genetic Algorithms have used strings of binary digits to implement the genotype as in Figure 3-2. This figure represents four genes each of which is comprised of four binary digits (bits) which allow the representation of 16 (2^4) different values. The mechanism for encoding and decoding these values is covered in the following section.

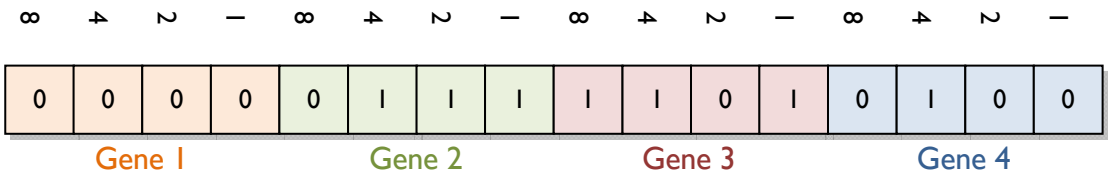

Figure 3-2: Example of a chromosome using binary strings

Two other commonly used forms of alternate generic encoding are real numbers and integer (or ordinal) values. Here, in contrast to the binary representation, the values required are stored directly in the chromosome as evidenced by Figure 3-3 which illustrates an integer-based chromosome encoding the save gene values (alleles) as in Figure 3-2.

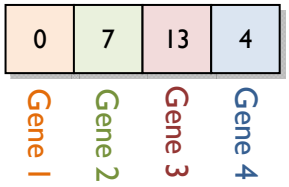

Figure 3-3: Example of a chromosome using integer values (same values as Figure 3-2)

Binary strings

The bit string encoding used for the chromosomes described above is one of the genetic representations (allelomorphs) that can be used. Figure 3-4 shows the structure of a conventionally encoded binary string with a value of 1,024 + 512 + 256 + 128 + 64 + 16 + 4 = 2004. The notations MSB and LSB indicate the Most and Least Significant Bit of the binary string respectively, referring to the influence that these bits have on the total value represented.

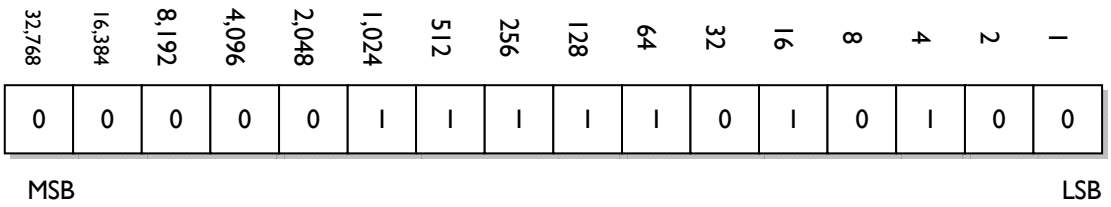

Figure 3-4: Conventional Binary String

Gray coding

Gray coding (Gray, 1953) is a means of recoding the binary string such that successive values of the phenotype are guaranteed to be represented by binary strings that differ in only a single bit position. There are many such Gray codes that can be developed for a given bit length - the type of coding implemented in this library is "binary-reflected" as this is quite simple to implement in an efficient manner. In this technique, with each increasing integral value, the next binary string representation is derived by flipping the least-significant bit that will produce a binary string that has yet to be encountered.

Value	0	1	2	3	4	5	6	7
Normal	0000	0001	0010	0011	0100	0101	0110	0111
Gray	0000	0001	0011	0010	0110	0111	0101	0100

Value	8	9	10	11	12	13	14	15
Normal	1000	1001	1010	1011	1100	1101	1110	1111
Gray	1100	1101	1111	1110	1010	1011	1001	1000

Table 3-1: Comparison of conventional binary strings and Gray-coded binary strings for 4-bit values

Table 3-1 illustrates the "adjacency property" (Goldberg, 1989) in which adjacent values of Gray-coded binary strings differ only by a single bit value. This has been shown (Michalewicz, 1992) to improve the performance of the mutation operation as it allows a more effective exploration of a local-search space. For example, to progress from a value of 7 to 8 with a conventional representation would require the flipping of four bits – clearly an unlikely mutation. However, using the Gray-coded representation such a change – as with any other in the sequence – would result from the modification of a single bit. With higher-order binary string length, this issue becomes more apparent.

Real Number Encoding

This encoding strategy uses chromosomes whose genes are not represented by binary digits - rather they use real numbers. The principal advantage of such a representation is that the decoding stage to obtain the phenotype is unnecessary since the genes are already in the form that they are required. This has a clear performance advantage for the algorithm, reinforced

by the fact that floating-point genes are far easier to manipulate than their bit-string equivalents and occupy significantly less memory space. Janikow & Michalewicz (1991) found experimentally that such real number encodings vastly outperformed their binary equivalents both in computation performance but also in their flexibility for the implementation of operators that function as vectors in the solution space itself.

Real encodings have significant benefit to the engineering design optimization field as there is typically a large number of parameters to describe the design options that can be chosen, as well as being more intuitive in the first place. As the size of the solution space is generally extremely large, the use of bit-strings for representing the solutions becomes prohibitive in terms of performance.

The mechanisms of crossover and mutation that are normally associated with the bit-string representation are appropriate for use with real number encoding. However, the representation allows new forms of crossover to be considered, such as averaging the value in each parent, weighted averages, weighting each child to a particular parent.

In a similar fashion, the mutation operations can be optimized for use with real numbers. Rather than simply replacing a gene with a newly generated random version, it is possible to add or subtract from the existing value of the gene or average the value with a random number. For the purposes of Water Distribution System design, however, real encodings are of lesser value – given that the decision variables are typically installation pipe diameters, tank sizing and the like: these things being more commonly represented as discrete values to reflect the reality of the "off-the-shelf" sizes of these elements. However, they find some utility in applications such as network calibration in which friction factors of individual pipes are modified to produce a match between observed and modelled pressures and flows.

Ordinal/Integral Value Encoding

Where a problem can be described in terms of a discrete number of choices, as is often the case in design optimization an ordinal encoding can be appropriate. Within an organism's objective function, the decoding of these ordinals often takes place using a look-up table.

Ordinal encoding is also well suited to combinatorial optimization problems such as the oft-cited Travelling Salesman Problem (Cormen *et al.,* 2001). In this problem, a salesman is required to visit a number of towns and return home without travelling to any town more than once. The aim of the optimization is to minimize the distance travelled by the salesman.

Whilst this problem can be encoded using a conventional bit-string arrangement, it is more intuitive to represent it as a chromosome of ordinal numbers, where each ordinal represents an individual town. When using this representation, however, it is necessary to use crossover and mutation operators that are aware of this form of encoding. The use of the conventional crossover and mutation operators are highly likely to result in the creation of invalid (illegitimate?) children. An ordinal chromosome must not contain duplicates and must have every ordinal represented at some locus of the chromosome.

3.2.2.2. Decoding

The first step is to decode the chromosome. The genome of the organism which is comprised, in this example, of bit-strings need have no implicit relationship to the values they relate to with respect to the problem - it is simply the organism's genetic material. The phenotype of a genome describes what the values of the genes *actually* mean and represents, therefore, the parameters used in determining the fitness of an organism. Thus decoding is the process of mapping the genome (or genotype) onto the phenotype.

3.2.2.3. Evaluation

Once the chromosome has been decoded, the calculation part of the objective function is responsible for plugging the decoded phenotype values into some form of function to map these parameters to a positive number, the fitness. The mapping function need not be a simple mathematical function, indeed in the case of hydraulic network optimization, the function in question is a complete hydraulic solver package, which produces a series of values for pressures and flows for given parts of the network that are further used to assess the fitness of an organism. In order for acceptable performance it is important that the calculation part of the objective function must be as swift as possible given that this function will be executed many tens of thousands of times during the runtime of an algorithm. Indeed, it is this speed that determines whether many problems are suitable candidates for solution through GAs.

3.3. Implementation

The Centre for Water Systems at the University of Exeter had previously developed genetic algorithm applications using a public-domain library, 'libGA' (Corcoran, 1993). This library was written in the C language (Kernighan & Ritchie, 1988) and was restricted to single-objective, genetic algorithm applications only. The author of this library had made a

concerted effort to ensure that the library was fully extensible through a complicated series of references to procedural variables. Such a structure allowed for the straightforward interchange of a number of different components - however, no mechanism was in place to promote code-reuse between components or to provide for extensibility of the library.

Prior to exposure to libGA, a number of design criteria had been identified as being desirable for creating a generic framework for implementing GA applications. Among these, which were not available with the libGA library, were:

- heterozygous chromosomes - where individual genes code for different phenotypical variables and the chromosomes themselves may have variable structure or length between the individuals in a population.
- heterogeneous chromosomes to support the above where genetic representation may vary according to that which is most appropriate for modelling the phenotype involved.
- parallelization of the execution of GAs.

These requirements and the absence of object-orientation from this C-based library led to its rejection as a basis for ongoing development.

At the beginning of the development of the GA software presented in this thesis, applications being developed in the Centre for Water Systems at the University of Exeter, were using the Delphi programming environment (Borland International, 1997), which is based on an object-oriented version of the Pascal language. In order to incorporate Corcoran's library into a controlling program written in Delphi, it would have been necessary to encapsulate the GA library within a Dynamic Link Library (DLL) written in C adding significantly to the complexity of employing the library. At the time, it was understood that this would effectively preclude the use of the library on any other platform other than Microsoft Windows operating systems running on Intel processors, as this was the only platform for which Delphi produced applications. However, several years later, Borland began producing versions of Delphi that would compile to Intel-based Linux as well as Microsoft's .NET platform. Integrating a C++ (Stroustrup, 1997) library such as GAlib would have involved the creation and maintenance of a "wrapper" to interface with the object-oriented C++ routines.

After many years of development, however, a decision was taken to improve the compatibility and portability of the library and all of the legacy Pascal code was replaced with

ANSI-compliant C++ code using the functionality provided by the C++ Standard Template Library (Stepanov & Lee, 1994) and the Boost extension library (www.boost.org). This led to a "forking" of the codebase in which other developers continued to extend the Pascal implementation – notably Engelhardt *et al.* (2002) who added Fonseca and Fleming's (1993) MOGA multi-objective optimization capability to the basic GA – testament to the flexibility of the original design was that this was accomplished with the minimum of structural changes to the library.

3.3.1. Algorithm Modularity

The processes that constitute the genetic algorithm are themselves represented by individual classes, known as Extensions, derived from a single parent. This class is primarily tasked with providing hooks that can be accessed by the user interface to query the status and to configure a given extension.

The basic genetic algorithm class contains placeholders for five Extension-derived objects. These represent Selection, Crossover, Mutation and Replacement. This and the other basic class relationships can be seen illustrated in Figure 3-5.

A virtual method *configure* is introduced which may be implemented by descendant classes if required. This would ordinarily be used to present interactive interface elements to the user to configure the operation of this extension. The user interface may query whether configuration is possible by examining the public *Configurable* property, which returns false by default. Other public properties allow the interface to retrieve information about the type of extension, author, version number and a description of the extension's characteristics for display in the user interface.

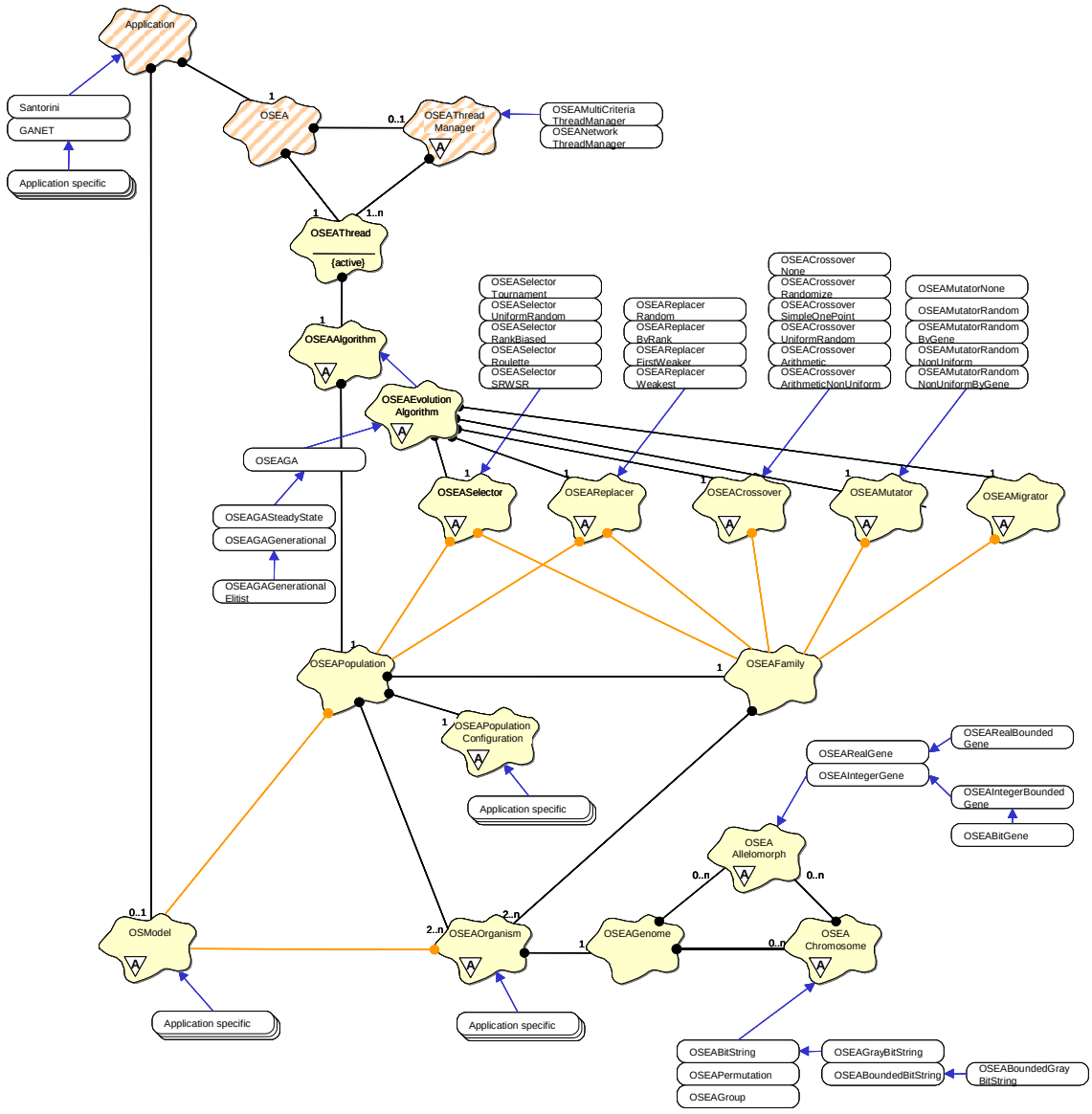

Figure 3-5: Outline class diagram for GA library implementation (Pascal version)

Selection

The abstract selection class introduces an abstract, virtual method *select* which must be overridden in all descendents of this class - failure to do so will result in a compilation error. This function is called by the algorithm implementation as the first step in the GA. The derived selection objects operate by querying the current crossover object as to the number of parent organisms required for its operation. This number of parents is then selected, according to the implementation of the selection process and references to the parents are stored in the GA's Family member – a repository in which parents and their offspring are stored during the recombination and evaluation process and prior to replacement into the population.

Crossover

The Crossover class implements the core functionality required for performing the crossover operation. A virtual method *cross* is introduced which performs the crossover on the parent organisms found in the GA's Family member. This method should be overridden in descendent classes, although this is not required - by default the children produced by the cross method are identical copies of their parents. The *cross* method is the second method called by the GA algorithm class during its execution cycle.

The base crossover class also exposes two properties for the use of other Extension classes. These are used to indicate how many parent organisms are required for the crossover and how many children are produced as a result. As these attributes are implemented using properties, it is possible to construct a descendent crossover class that dynamically varies the number of parents/children it manipulates. The mechanism of these variations is hidden from the other extensions, which depend on this information.

In addition, a number of protected tools are provided for descendent crossover classes. These are intended to provide a level of abstraction from the type of chromosome representation being used (indirected or expansive map). Although the genome class itself implements this abstraction, some operations, particularly those that iterate over the genome structure, are more efficient with one representation than the other. Consequently, implementing a further level of abstraction in the crossover class, which is mostly concerned with iteration, allows the derivation of *cross* methods, which have variant implementations depending on which representation is being used. Among these methods are functions to determine a valid crossover point and most importantly, to exchange the genes between the parent's genomes over a given range.

Mutation

Support for mutation is represented by the Mutator class. This class introduces a single virtual method, mutate, which performs the implemented mutation operation on the child organisms present in the GA algorithm class's Family member. By default, this method does nothing, leaving the children unmodified. The mutate method is the third called by the GA algorithm class during its execution cycle. Ordinarily, however, this execution is conditional - derived mutate methods should reference the GA algorithm class's MutationProbability property and determine, using a random number, whether to proceed with the mutation. Notwithstanding this, this behaviour has not been made obligatory,

A further method actsOn is provided for internal use. This method takes a gene or a chromosome as a parameter and returns true or false depending on whether the implemented mutation method is able to mutate the referred object.

Replacement

The abstract replacement class, Replacer, introduces a single abstract, virtual method replace which according to some scheme, inserts the children present in the GA algorithm class's Family member into the population referenced by the GA. In the case of the Generational GA implementation, the replacement technique is not specified as child organisms are always incorporated in the new population. The replace function of this class is normally the last to be called by the execution cycle of the algorithm.

3.3.2. Genetic Representation

3.3.2.1. Chromosome

This class is the immediate ancestor for all genetic elements that group genes together. It may be used on its own, simply to group related genes, or in the case of derivatives, to add additional functionality. Its most important descendents are those concerned with implementing bit strings and ordinal combinatorial optimization strings.

The *gene()* property introduced in this class allows the retrieval of a gene from a given locus in the chromosome. The implementation of this property is recursive, thus if gene *n* is to be found in a nested chromosome it will be correctly returned.

Also prototyped at this level is the "repair" functionality, implemented virtually. Repair functions are applied to a chromosome, after recombination, in order to ensure that the decision variable values are valid. Whilst having no effect at this point in the hierarchy, this method is used in derivative chromosome types, particularly those for permutation optimization in which duplicates within the chromosome are not permitted and need to be removed where generated during recombination. By devolving this to this level, it is possible to specify repair mechanisms for small parts of the genome, rather than operating across the entire genome.

3.3.2.2. Genome

The genome class is a specialized derivation of the chromosome class. It differs from the latter in that it transparently maintains the expansive map of the chromosome if that

representation is being used. It also introduces a length property that can be queried to determine the total length of the chromosome in genes.

3.3.3. Third Party Extensions

Since the initial development of the Genetic Algorithm library, a number of researchers have exploited the extensibility afforded by the library to add significant new functionality.

To bring the GA library to a wider audience of developers within the Centre for Water Systems, the library was encapsulated as a custom C++Builder Component by Edward Keedwell. Offering a "plug-in" optimization component for use in the C++Builder environment, this development makes the production of a simple optimization application a matter of using the Builder user interface to drag a component onto a window and then to configure the parameters of the optimization using the interactive dialog boxes provided. Additional components are provided to provide reporting functionality including a specialized charting component for graphing of the progress of the algorithm.

In this form, the library has been used in a variety of research and commercial projects as well as by undergraduate and postgraduate projects.

3.4. Conclusions

A library for the development of evolution algorithms is presented. This library makes extensive use of object-oriented programming techniques to implement an extensible, open architecture for genetic algorithms and related optimization techniques. The modular nature allowing for the extension of all components of the algorithm is described and is demonstrated through the implementation of specific algorithm implementations. The final state of the library is illustrated in Figure 3-7 for single objective algorithms and Figure 3-8 for multiple objective algorithms.

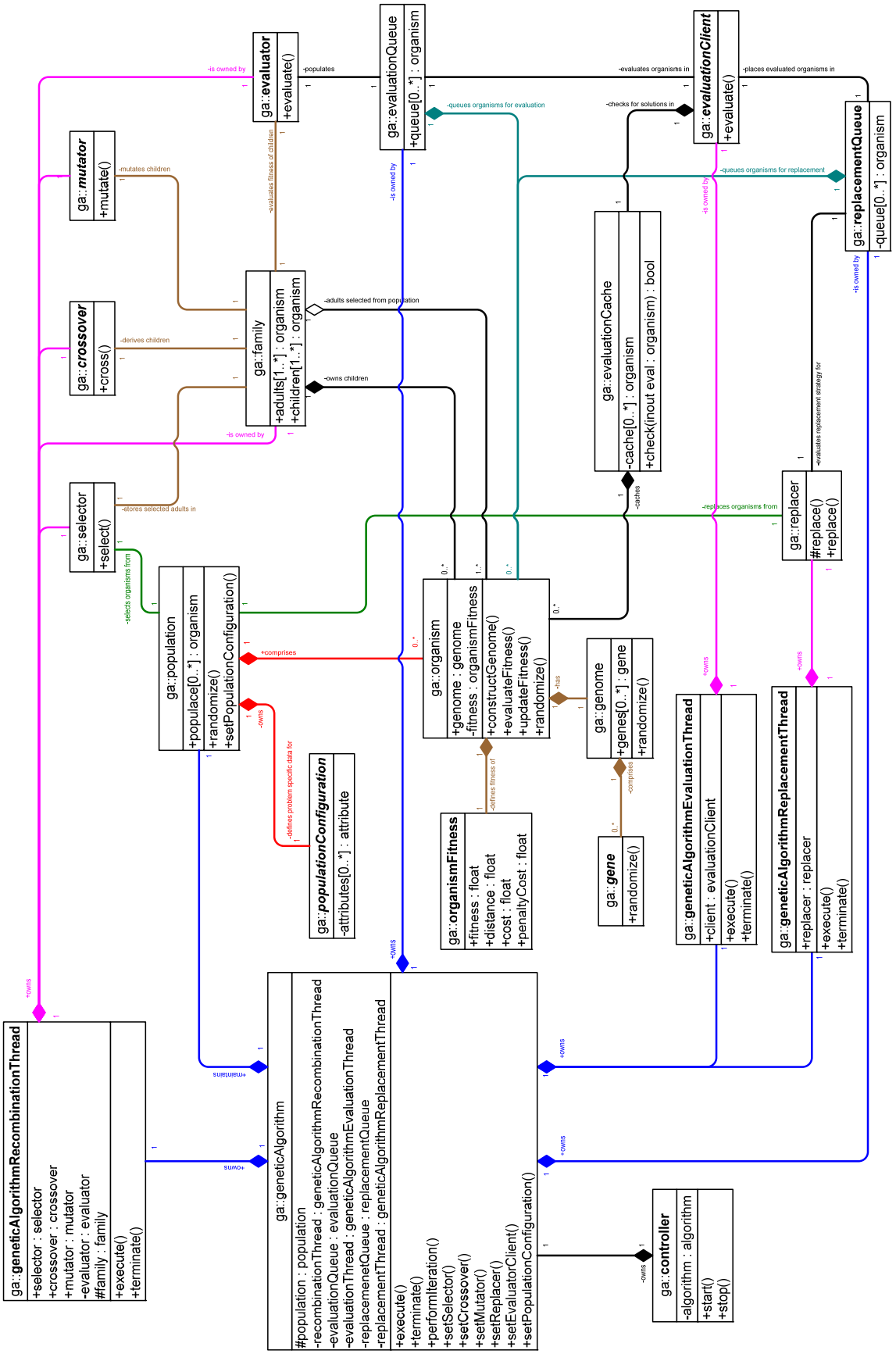

Figure 3-7: GA methodology: final design (single objective)

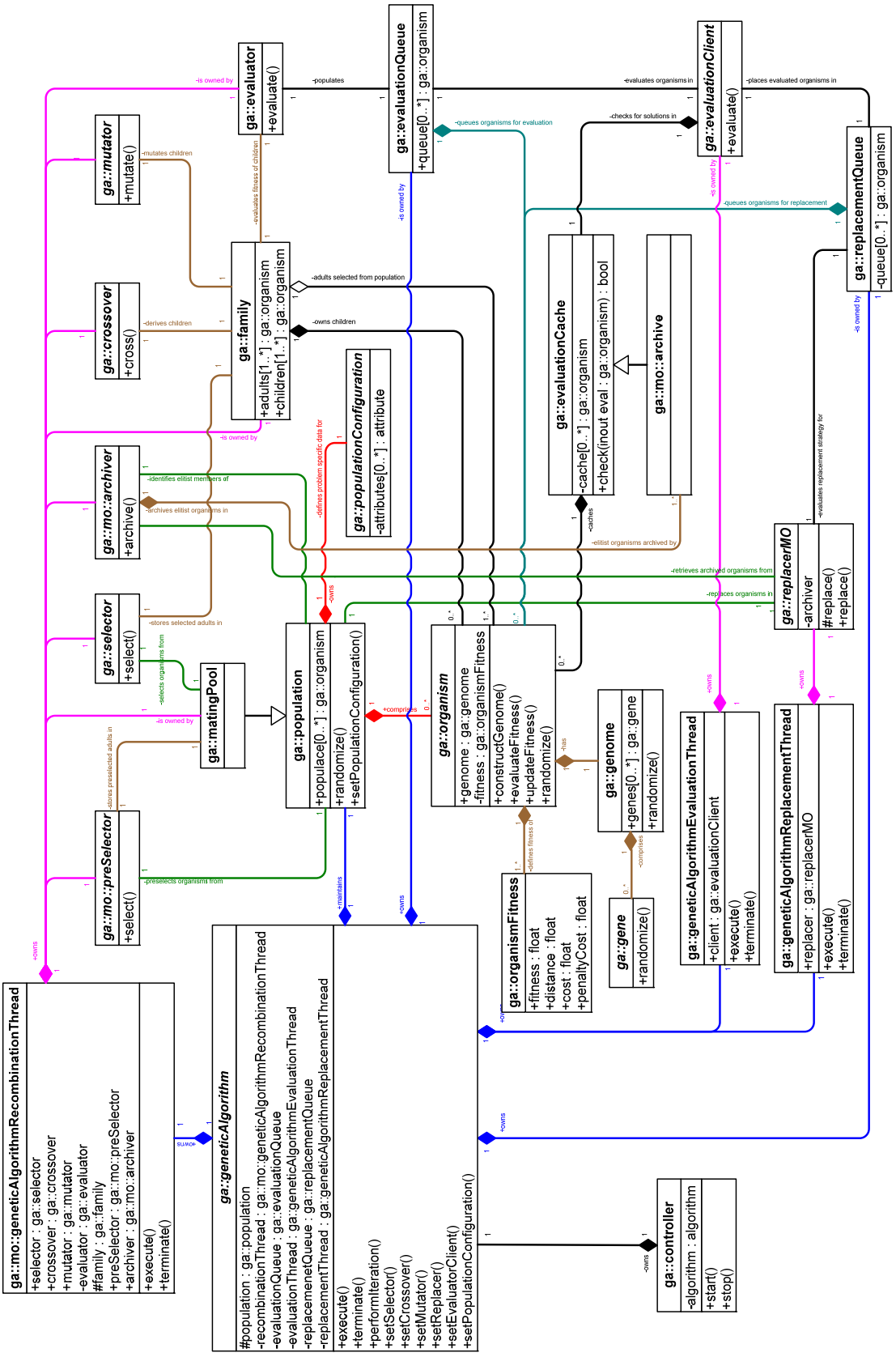

Figure 3-8: GA methodology: final design (multiple objective)

The fitness for purpose of this library and its constituent methodologies is demonstrated through its deployment in a number of research applications covering a number of hydroinformatic subject areas, e.g. Fullerton *et al.* (2002) (storm water flow modelling and optimization), Savić *et al.* (2000) (optimal design of expansion to a large-scale hydraulic network) and Engelhardt *et al.* (2002) (whole-life-costing for water distribution network management). In addition, the software has been used in a number of generic, commercial software applications. GAnet (Morley *et al.*, 2000) and GAcal (Walters *et al.*, 1998) were sold by Ewan Associates as bespoke software applications for design/reinforcement and calibration of water networks, respectively. More recently, the library has provided the optimization functionality in SEAMS' WiLCO software (Engelhardt & Skipworth, 2005) used for the optimization of asset rehabilitation strategies to minimize whole-life-costs in water distribution and sewer networks.

Chapter 4. Extending the GA methodology

4.1. Introduction

Evolution algorithms are particularly sensitive to unexpected performance impediments due to the tightly iterative nature of their design. Thus, efficient implementation of the algorithmic structures themselves is essential. Even small inefficiencies can be magnified when looped repetitively. Several novel techniques have been investigated to improve the computational performance of generic genetic algorithms. The representations of binary strings have been explored in this thesis and a novel representation combining attributes of the integer representation is proposed as a contribution. In addition, techniques for caching decoded values and entire solutions are introduced along with mechanisms for directing the mutation of individuals during an optimization. Where appropriate, fragments of C++ or pseudo-code have been incorporated to help illustrate the implementation of the techniques discussed.

4.2. Binary String Implementation

4.2.1. Introduction

Classically, binary strings – strings of bits (binary digits) – have been used in Genetic Algorithms to represent the chromosomes of the population (Goldberg, 1989; Michalewicz, 1992). Whilst the binary representation is simple to manipulate and understand, it can suffer in terms of computational efficiency because it is both a larger structure to manipulate and typically requires encoding and decoding into the specific domain required by the evaluation function of the GA.

A variety of representations is examined by the Author for implementing the genes (genotypes) and their relative efficiency in computational terms established. A novel representation that combines characteristics from the binary string and integer representations is introduced in this thesis and contrasted with the established representations.

4.2.1.1. Genotype Representations

This section describes the basic binary string representations utilised in the thesis. Thereafter, the remainder of the chapter describes novel techniques that have been devised to improve the performance of such representations.

Binary strings

Binary strings are the conventional representation for Genetic Algorithms. The process for encoding a bit string is as follows:

```
void binary string::setValue(int a_Value)
{
 for (int loop=0; loop < m_BitLength; ++loop)
 {
  m_Bit[loop]= a_Value & 1;
  a_Value >>= 1;
 }
}
```

Figure 4-1: C++ code fragment for encoding a binary string

The process is performed in reverse to decode the bit string. It should be clear from the description of the encode process that performing the encode (Figure 4-1) or decode (Figure 4-2) cycle entails significant expenditure of processor if the process is repeated often - as is the case in a genetic algorithm. Consequently, these routines are ideal for optimization in assembly language.

```
int binary string::value()
{
 int result= 0;
 int multiplier= 0;
 for (int loop= 0; loop < m_BitLength; ++loop)
 {
  result+= multiplier * m_Bit[loop];
  multiplier <<= 1;
 }
}
```

Figure 4-2: C++ code fragment for decoding binary string

Gray coding of binary strings

To facilitate the (transparent) use of Gray-coded binary strings in the GA library a derived class of the standard binary string is provided. This class overrides the *getValue* and *setValue* methods of the binary string class to perform the Gray encoding and decoding without the end-user or other constituents of the algorithm being aware of the change. The *setValue* function has been optimized by the Author for C++ implementation and relies on a transformation of the incoming value to perform the Gray encoding.

```
bool grayBinary string::setValue(int a_Value)
{
 int newValue= a_Value ^ (a_Value >> 1);
 binary string::setValue(newValue);
}
```

Figure 4-3: C++ code fragment for encoding a Gray-coded binary string

The encoding shown in Figure 4-3 permutes the incoming phenotypic value *a_Value* and produces a modified value by performing the XOR operation (see Table 4-1)on *a_Value* against itself when integer divided by 2.

Value A	Value B	Result
0	0	1
1	0	1
0	1	1
1	1	0

Table 4-1: Binary eXclusive OR (XOR) operation

The C++ '>>' operation above notates a bitwise right shift of 1 binary place which has the effect of integer dividing by two – any remainder is discarded implicitly. This process is illustrated in the following where the example string (Figure 4-4) has a value of 1,024 + 512 + 256 + 128 + 64 + 16 + 4 + 1 = 2005.

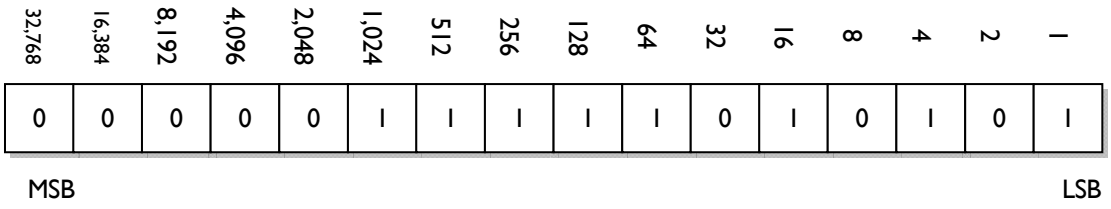

Figure 4-4: Binary string prior to right shift

Figure 4-5 shows the same string following a right shift of one place in which each bit has moved one place to the right. The end-most bit was shifted out of the string and has been discarded. The resulting value is 512 + 256+ 128+ 64 + 32 + 8 + 2 = 1002 –the integer division by two of the original.

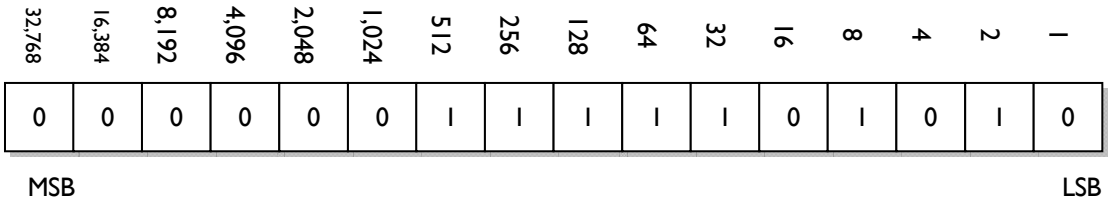

Figure 4-5: Binary string following right shift

The C++ '^' operator performs the XOR. Once the modified value is calculated, it is passed to the inherited *setValue* method to be encoded in the binary string in the normal fashion. Referring to Table 3-1 it can be seen that when using the binary-reflected means of Gray coding, a value of 10 (nominally 1010_2) should be encoded as 1111_2 equivalent to a

decimal value of 15. Following the operation outlined in Figure 4-3 gives 10 XOR 5 = 15, as required.

Decoding a Gray-coded value is, however, less straightforward than encoding and requires some iteration:

```
int grayBinary string::getValue()
{
 int result= binary string::getValue();
 int mask= result >> 1;
 while (mask > 0)
 {
  result ^= mask;
  mask >> 1;
 }
 return result;
}
```

Figure 4-6: C++ code fragment for decoding a Gray-coded binary string

One of the properties of the binary-reflected Gray encoding is that the most-significant bit remains the same between the encoded and decoded representations - thus the decode routine can start on the second-most-significant bit. A dry run through this code is shown in Table 4-2 illustrating the reverse of the example above in which a Gray-coded genotype of 15 (1111_2) is converted back to the phenotype of 10 (1010_2).

Program step	Result		Mask	
int result= binary string::getValue();	15	1111_2	undefined	**undefined**
int mask= result >> 1;	15	1111_2	7	**0111_2**
while (mask>0)	15	1111_2	7	**0111_2**
result ^= mask;	8	1000_2	7	**0111_2**
mask >> 1;	8	1000_2	3	**0011_2**
result ^= mask;	11	1011_2	3	**0011_2**
mask >> 1;	11	1011_2	1	**0001_2**
result ^= mask;	10	1010_2	1	**0001_2**
mask >> 1;	10	1010_2	0	**0000_2**
return result;	**10**	**1010_2**	**0**	**0000_2**

Table 4-2: Example dry run for decoding a Gray-coded binary string

As with the conventionally encoded genotype, storing the value of the binary string – post Gray-decoding – would be expected to render a performance improvement.

A number of different implementations for representing binary strings have been investigated by the Author to assess their relative benefits with respect to performance and memory requirements. The details of implementation are specifically tailored to C++

representations. However, many of the principles involved remain relevant when considered across operating system platforms as a whole.

4.2.2. Conventional Representations

1 bit per bit

In terms of memory efficiency, it is obvious that the most economical means of representing a string of bits is to represent each bit as a single bit. However, modern operating systems do not permit the access of single bits in their own right – restricting the developer to the byte as the smallest unit of addressable memory. Thus, it is necessary to employ "tricks" to achieve such representations by using classes to pack and unpack bits from a byte-wise representation. Two such representations have been evaluated. However, since much of the analysis of these representations is specific to C++, discussion of the differences in their implementation is not presented here.

1 byte and 2 bytes per bit

In C++, the one byte per bit representation is correctly implemented using the *unsigned char* type, which, at first sight, seems rather counterintuitive, given that this is the datatype used to store a textual character. However, this is also the correct type for an integral variable that can take on the values in the range 0 to 255. These two types were included in the analysis for completeness.

1 word per bit

The most straightforward representation that can be used is one in which an entire processor word (i.e. 32 bits under most current operating systems) is used to represent a single bit in the string. Whilst this arrangement is the most expensive in terms of memory requirement, it should theoretically perform quickest because it is, by definition, using the processors native unit of memory access.

STL implementation

The Standard Template Library (STL) is an integral part of the C++ language and provides a number of simple containers, algorithms and other data structures to accelerate development in C++. Unusually, for a library, the STL is a *specification* of how an implementation should behave rather than an implementation in its own right. Compiler developers are free to choose whether to use an existing STL implementation or to provide their own. The result

of this is that highly optimized STL implementations are available for most compilers and platforms.

Using the STL container, *vector<bool>,* it is possible to implement a bit-string using 1 bit per pixel. *vector<bool>* achieves this by applying bitwise operators to set individual bits in a byte and is atypical because this means of access violates several of the principles of STL containers. The use of bitwise operators involves a small, but tangible, processing overhead to each read and write access of the binary string.

Boost implementation

"Boost" is a third-party library that extends the basic STL containers, adds vital platform-independent support for common development paradigms and can be considered a repository for candidate code for the STL itself. Boost has come about primarily as a result of the lengthy times involved in ratifying changes to the STL proper.

As well as *vector<bool>* the STL has a container called *bitset*, which allows bitwise operations to be applied to the binary string without needing to decode it first and has useful functions to directly translate binary strings to integers and vice-versa. Unfortunately, the implementation of *bitset* requires its size to be known at compile time - rendering it unsuitable for generic GA applications with variable chromosome lengths. Boost, however, includes a *dynamic_bitset* container, which overcomes this limitation.

4.2.3. Hybridized integer gene

Binary string gene representations in genetic algorithms are traditionally viewed as being inefficient in terms of performance because of the additional processing required to encode and decode values as well as overheads incurred in the storage and recombination of large data structures, as described in 3.2.2.1.

Since integer values are mapped directly to binary digits by the processor itself, it makes little sense to recreate the same representation for our own purposes. Instead, by using modified crossover and mutation operators which effectively act on the binary representation of the gene value, it is possible to manipulate integer values *as* binary strings with all the attendant advantages of speed in manipulating integer values as well of those of schema preservation from binary strings – thus producing a hybrid between the two techniques. The sole disadvantage of this representation is that the maximum length of the gene is limited to the length of the integer representation in the operating system/processor –

nominally 32 bits. Clearly, though, as this still allows each gene to take on over 4.2 billion values, this should not be regarded as a significant constraint. This approach, as shown below is very close in performance to a pure integer-based approach, but maintains the crossover performance of a binary string. In addition, many of the operations undertaken translate directly to the processor's own microcode acting on its registers maximizing the performance benefit accrued.

4.2.3.1. Crossover representation

The crossover and mutation operations take the form of bitwise operators, which have the advantage of being directly supported by the floating-point processor in hardware. Binary strings are represented with a sequence of bits, each one corresponding to a power of 2. The value of a binary string is computed by summing the power of 2 represented by each bit that is set. Conventionally, binary strings are represented right-to-left from the Least Significant Bit to the Most Significant Bit to conform to our own decimal numbering system. However, in hardware, the strings are normally oriented in reverse.

In the case of crossover – as in all of the representations surveyed here – whole genes are copied *en masse*. Partially recombined genes are dealt with as follows:

Given two parent genes [*a*] and [*b*] of values 8,221 and 392:

0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 4-7: Parent gene *a* (value= 8,221)

0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0

Figure 4-8: Parent gene *b* (value= 392)

Performing a crossover within the resulting gene will result in child genes as those seen in Figure 4-9. This particular crossover is the result of a crossover at the eighth locus in the gene.

0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	1

Figure 4-9: Expected child outputs from crossover

These recombined chromosomes give post-crossover values of 8,328 and 285 respectively. Crossover operators on binary strings conventionally swap each bit, as

appropriate. Using bitwise operators on integer variables has exactly the same outcome – but with far better performance characteristics.

First, it is necessary to create a mask, which will be used to identify which genes of the chromosome are to be swapped. The mask simply set to zeros for one side of the crossover and ones for the other:

0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 4-10: Crossover mask *c* (value= 255)

Construction of the mask is simplified by retaining an array in memory containing the powers of 2 from 1 to 32. Using this array it is straightforward to create a mask based on the start locus of the crossover *(startLocus)* and the number of genes to crossover *(count)*:

```
int mask= 0;
int endLocus= startLocus + count;
for (loop= startLocus; loop < endPoint; ++loop)
 mask= mask | power2_32[loop];
```

Figure 4-11: C++ code to generate mask for crossover

The C++ '|' operator represents the OR operation. This mask [*c*] is combined with each of the existing chromosomes [*a*] and [*b*] using the Boolean AND (C++ '&') operation to give us two halves of the new chromosomes:

```
d = a & c;
e = b & c;
```

0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 4-12: Masked parent *d* – least significant byte (value= 29)

0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0

Figure 4-13: Masked parent *e* - least significant byte (value= 136)

In order to isolate the other halves of the chromosomes for crossover, it is straightforward to subtract these newly created halves from the originals:

```
f = a – d;
g = b – e;
```

0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-14: Masked parent *f* - most significant byte (value= 8,192)

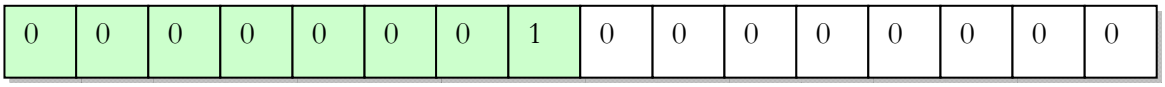

Figure 4-15: Masked parent *g* - most significant byte (value= 256)

The final stage of the crossover is straightforward being merely to combine the opposing halves of the chromosomal fragments through the Boolean OR operation:

```
h = e | f;
i = d | g;
```

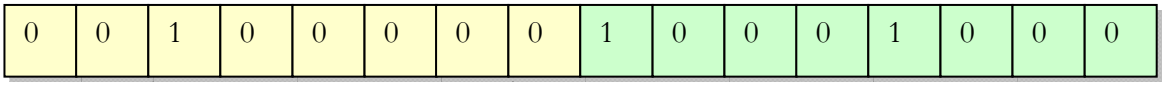

Figure 4-16: Child *h* (value= 8,328)

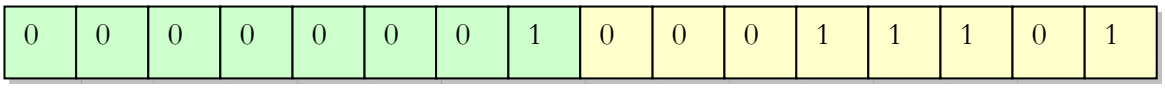

Figure 4-17: Child *i* (value= 285)

As can be seen, the outputs [*h*] and [*i*] are the binary strings expected from Figure 4-9 with values of 8,328 and 285 respectively.

4.2.3.2. Mutation representation

Application of mutation is uncomplicated using the array of powers of 2 used in creating the crossover mask. Having determined a locus for the mutation to take place *(locus)*, the XOR operation is performed with the value of the gene against the power of 2 represented by that locus – resulting in it being removed from the gene if it is already set or adding it if it is absent.

4.2.4. Experimentation

To determine the relative performance of the binary string representations a simple GA was devised that reflects the normal process of decoding, encoding and recombination that a binary string would undergo. The performance of the representation was then evaluated over a series of five runs, each of one million evaluations. This performance is measured in terms of evaluations per second.

The organism representation was 25 binary string genes, each of which was 25 bits long – a total genome length of 625 bits. To minimize the impact of an objective function on the performance it was assigned a simple task to sum the values of each gene and to return that sum as the fitness of the individual organism. The memory requirements and performance results from the experimentation are presented in Table 4-3.

Binary string representation	Memory requirement † *(bytes)*	Performance *(evaluations/second)*
1 bit/bit (STL)	100	7,568
1 bit/bit (Boost)	100	8,594
1 byte/bit	625	9,714
2 bytes/bit	1,250	9,581
1 word/bit	2,500	9,263
Hybridized integer	100	11,083
Pure integer	100	13,339

Table 4-3: Binary string implementations: relative performance

† *Per individual. Each representation has additional memory overheads associated with the implementation of the containing class – however, these overheads are largely constant and, for clarity, are not considered here.*

4.2.5. Conclusions

Two clear conclusions may be drawn from the above results. First is the surprisingly good performance of the 1 byte/bit representation relative to the 1 word/bit which was expected to have the highest performance. This suggests that, despite the processor intrinsically using "words" in its manipulations, there is greater overhead in manipulating binary strings of this size than the overhead on the processor of having to perform manipulations on individual bytes.

The second observation is the massive performance and memory advantage enjoyed by the hybridized integer representation. This is due to it not having encode/decode cycles at all – rather the value is directly stored in an integer variable and is only considered as a binary string when used in crossover and mutation.

Despite being fastest, the integer representation would not yield as good results for a given run-time as, using the conventional GA operators, it is only able to recombine chromosomes *between* genes and as such requires a higher mutation rate to operate effectively. This is because the only means by which the values of the individual genes may change is through mutation. Running such a GA without specialised operators or an enhanced mutation rate often results in a population that quickly stagnates, converging to a local optimum owing to a lack of genetic diversity.

4.3. Binary String Caching

4.3.1. Introduction

Aside from memory requirements, binary string use in GAs is compromised because of the additional overheads in reading and writing values with them – although these overheads may ultimately prove to be insignificant compared to the runtime of the evaluation function. One mechanism for improving this performance is to cache the decoded value of a binary string and then to keep track of any changes to the binary string. During a GA run, only binary strings that are directly affected by a crossover or mutation operator will have their values changed. This means that, particularly in the case of binary strings that represent real numbers, a significant amount of binary string processing and recalculation can be avoided.

4.3.2. Implementation

Along with the bit-wise data itself, each binary string will preserve an integral variable that represents the decoded value of the string. In addition, a single Boolean variable is used to determine whether the binary string has been modified or not. When returning the value of the string, this variable is tested and if the string is found to be unmodified – i.e. the representation has not changed – the pre-decoded value is returned. When setting the string to a specific value, for instance during randomization, it is also sensible to store this value in the pre-decoded value to accelerate decoding. All that remains is for the crossover and mutation operators to be adapted to ensure that they mark the modified flag of any binary string whose content they modify.

4.3.3. Experimentation

To analyse the relative performance of caching binary strings two simple genetic algorithms were developed each involving 25 binary strings each comprising 25 bits. The evaluation function for both GAs involves extracting an integral value from the binary string, before transforming it to a certain range. A summation of all of the values in the string is the value returned by the objective function, thus:

$$F = \sum_{i=1}^{25} V_i + C$$

ii)

Where F is the fitness of the individual, V is the value encoded by the gene, i, and C is a constant which is used to shift the value out of the range $0..n$ to $C..n+C$.

Performance figures are also given for performing the same operations on a Gray-coded binary string. Gray-coded strings have an additional performance overhead associated with their encoding and decoding and, as such, are likely to benefit from caching.

Baseline figures for the pure integer implementation are also given – although it should be noted that this representation would not attain similar algorithmic performance without the provision of customized operators.

Number Type	Encoding	Representation	Memory (bytes)	Uncached (eval./sec.)	Cached (eval./sec.)	Improvement %
Binary String Integer (25 bits)	Normal	1 bit/bit *(STL)*	100	7,568	8,418	11.23%
		1 bit/bit *(Boost)*	100	8,594	9,206	7.12%
		2 bytes/bit	1,250	9,581	9,787	2.15%
		4 bytes/bit	2,500	9,263	9,498	2.54%
	Gray	1 bit/bit *(STL)*	100	6,287	6,852	8.99%
		1 bit/bit *(Boost)*	100	7,018	7,404	5.50%
		1 byte/bit	625	7,581	7,582	0.01%
		2 bytes/bit	1,250	7,530	7,492	-0.50%
		4 bytes/bit	2,500	7,259	7,194	-0.90%
Pure Integer		native	100	13,339	13,339	n/a
Hybridized Integer	Normal	native	100	11,083	11,093	n/a
	Gray	native	100	8,491	10,087	18.80%

Table 4-4: Comparison of cached/uncached performance for binary string representations

4.3.4. Conclusions

This per-string caching produces a marginal benefit for the fastest of the conventional binary string representations – and has a more significant impact on the slower implementations – as might be expected. The Gray-coded representations also show significant improvements – though the effect of the caching on the faster routines is quite odd: impacting detrimentally on performance. It is surmised that – having isolated environmental factors – this decrease in performance can be attributed largely to the additional overheads imposed by encoding/decoding Gray-coded binary strings relative to the complexity of the evaluation function, which, in this instance, is trivial. Problems that are more complex could expect to see a more significant benefit from using the binary string caching.

Once again, the hybridized integer representations win out – as with the pure integer representation, caching is redundant as the value is already stored natively as an integer variable. However, the caching can be used in the Gray-coded variant to accelerate its

operations and, as can be seen from Table 4-4 above, Gray-coded binary strings can now be used with minimal performance impact as the hybridized integer version is shown to be faster than all of the conventional, normally encoded representations. Despite the Integer representation being fastest, this representation would not yield as good results for a given run-time as it only recombines chromosomes between genes and as such requires a higher mutation rate to operate effectively.

4.4. Solution Caching

During the lifetime of a genetic algorithm – and in particular towards the end of a run – the algorithm may revisit solutions that it has already evaluated. In a classical GA there is no mechanism to prevent this duplicated effort. By maintaining a cache of previously visited solutions, it is possible to avoid repeated calls to the objective function. This is clearly an important consideration when such a call may have significant processing requirements. Figure 4-18 shows how this technique is integrated into a classical GA. Despite being an apparently obvious strategy to employ within Genetic Algorithms, there is a puzzling lack of literature on the subject. Kratica (1999) and Povinelli & Feng (1999) investigate similar caching approaches utilising a simple hash table for storing the most-recently accessed, cached objective function results and finds that the approach is viable in reducing GA runtime performance. It may be that the objective functions employed with GAs hereto may lack the computational complexity to merit the widespread application of the technique; certainly for most hydroinformatics applications this is not the case. This section proposes two forms of cache for general use with GAs :

- A simple, cache based on the common binary-tree data structure – both in a simple and "tiered" approach in which older, unused cached items percolate down through tiers of increasingly larger (and therefore slower) caches.
- A novel cache utilising the "Judy Tree" algorithm, commonly used in large scale. intensive hardware data access caching applications (e.g. server hard drives) which offers unparalleled access performance.

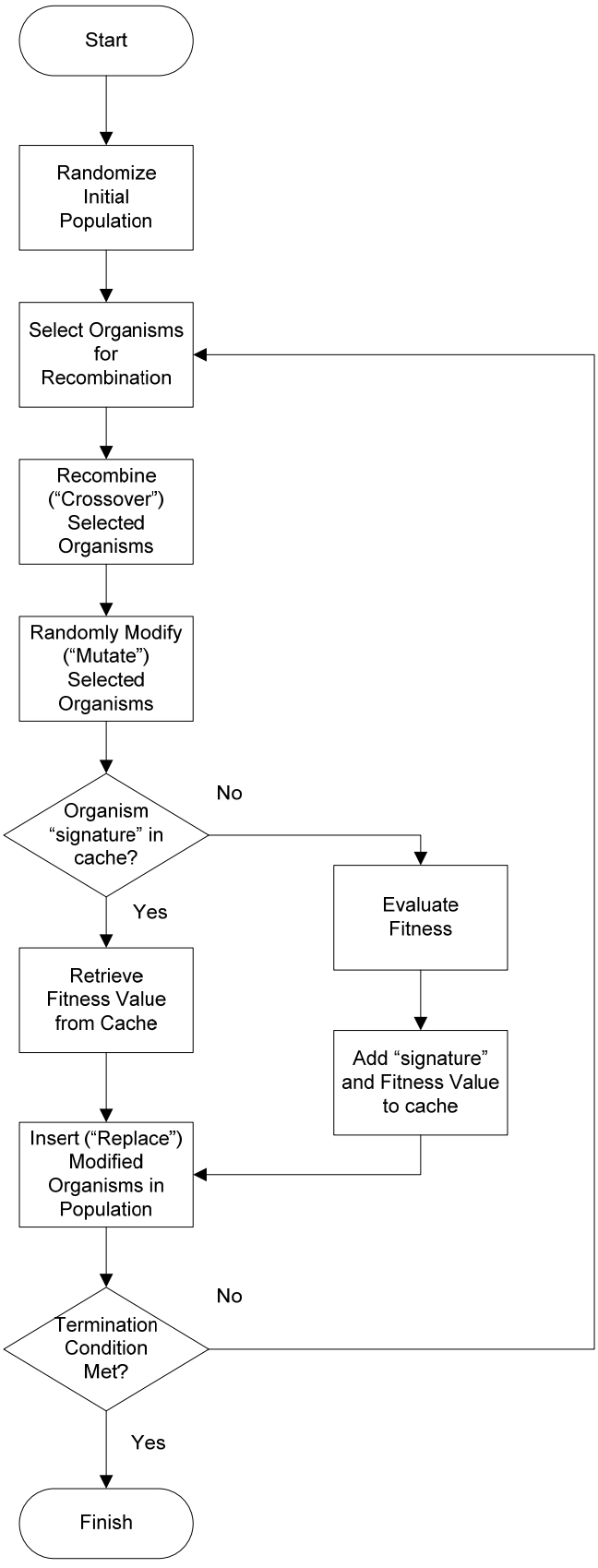

Figure 4-18: Flowchart illustrating the role of caching in a simple GA

4.4.1. Red-Black Binary tree cache

The basic caching structure is a variant of a conventional binary tree, which is ordered according to a representation of the genome and contains fitness information for that evaluated genome. Trees offer an efficient data structure for storing ordered data. Binary trees are among the most commonly used variants and have been specialised for various tasks (Knuth, 1997a). Figure 4-19 illustrates a binary tree with seven data members comprising a "value" and two pointers to other data members. Conventionally these pointers are used to point to "smaller" and "larger" data members. In addition, each node possesses a data record of some type – in the context of caching for evolution algorithms, this record will contain information about the fitness of a solution.

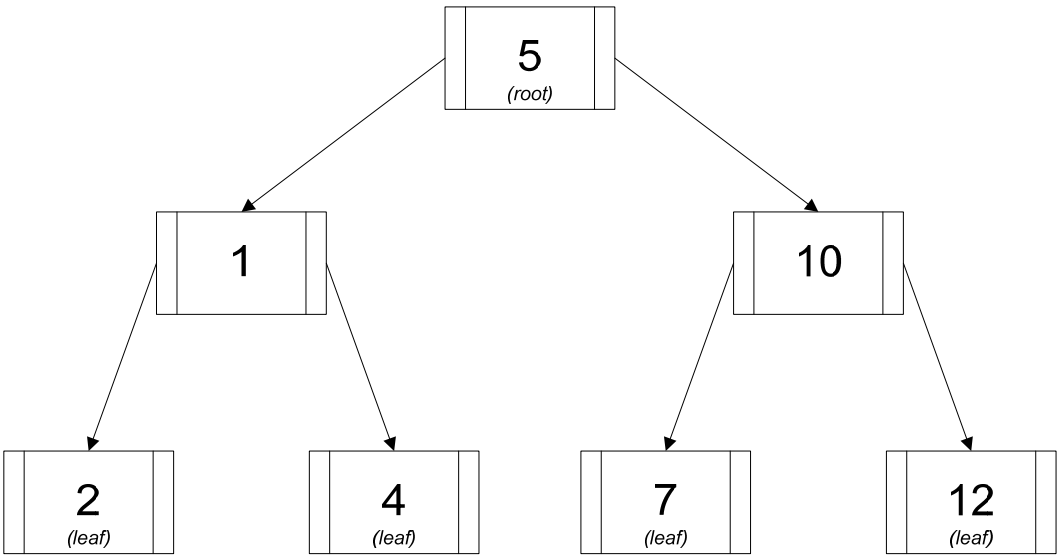

Figure 4-19: Traditional binary tree representation

The tree is assembled as data is added to it and, as such, the resulting form of the tree is highly dependent on the order in which values are inserted. Searching the tree for a given value begins at the root node of the tree (value "5" in Figure 4-19) and then proceeds to traverse the "smaller" or "larger" branches of the tree depending on the value sought and the value of the current node of the tree. The search continues recursively until either the value is located or there is no branch to traverse – in which case the value sought is not present in the tree. In the above example, Figure 4-19, a search for number "7" proceeds through the nodes of the tree in the sequence "5" – "10" –"7" comparing the target value with the node value at each stage and determining which branch to take through the tree. Similarly, a search for the number "3" would proceed through nodes "5" – "1" – "4" whereupon the

search would conclude that "3" is not present in the tree owing to the fact that there is no node attached to the "less-than" branch of the "4" node.

The tree shown in Figure 4-19 represents a "perfect" binary tree – one in which all of the nodes have exactly two branches and the depth of the tree (in this case 3) is constant. This is the most efficient tree structure for searching as, in the example, the maximum number of comparisons that would have to be made to locate a value in the tree is 3. Figure 4-20, however, shows an alternative scenario for a similar tree in which the data was inserted in a different order resulting in a lop-sided, unbalanced tree. Despite having the same number of data members, a search of this particular tree may require as many as 5 comparisons.

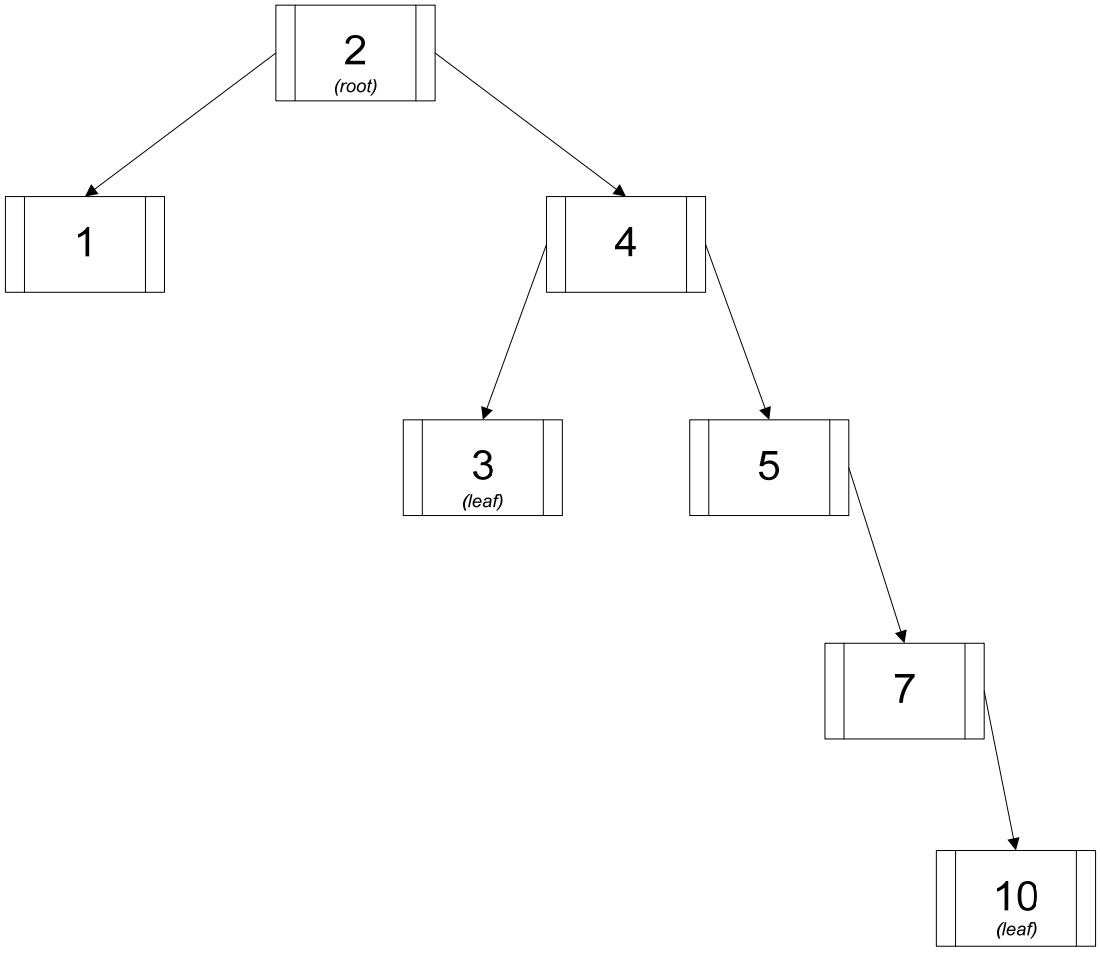

Figure 4-20: Unbalanced binary tree

Once a traditional binary tree has been populated with all the data it is possible to run a balancing algorithm to rearrange the tree in a more optimal fashion. However, this is an expensive operation to run if the data in the tree is constantly changing – as is the case in a caching application. Pfaff (2004) compared 20 different representations for Binary Search

Trees (BSTs) and found that the most efficient variant when considering the input of random or near-random data was the Red-Black binary tree (Cormen *et al.*, 2001). Thus, the first implementation considered in this thesis employs the C++ Standard Template Library (STL) container *map* (Josuttis, 1996) which, in most implementations of the STL, is an implemented with a red-black binary tree. This algorithm practises a common form of self-organization. Self-organization is a vital characteristic for caching algorithms: searches of binary trees are most efficient when the tree is balanced such that the average depth of tree that needs to be searched to locate a record is minimized. Figure 4-21 illustrates an example Red-Black binary tree. Along with the perquisites of the binary tree structure above, Red-black binary trees have additional constraints in order to be valid:

- Each node has two children, each coloured either Red or Black.
- Every leaf node (those at the extremities of the network) is coloured Black.
- Both children of Red nodes are coloured Black (a consequence of this is that there cannot be two consecutive Red nodes in a path from the root to a leaf).
- Every path from the tree root to a leaf contains the same number of Black nodes (known as the "Black Height" of the tree).

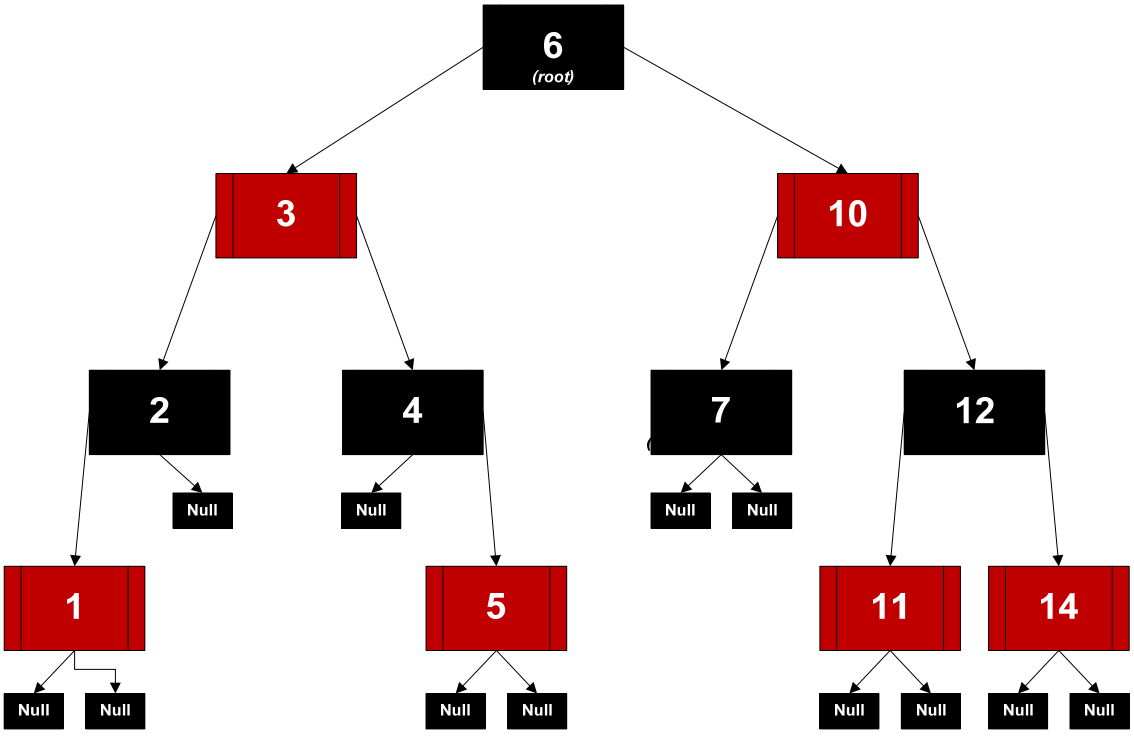

Figure 4-21: Example Red-Black binary tree

When a new node is added to a Red-Black binary tree, its nominal insertion point is identified. If this insertion violates any of the conditions of the tree structure then a recursive series of reorganisations are undertaken to restructure the tree dynamically in order to conform to the constraints. Thus, Red-Black binary tree searches will normally come close to the O (log *n*) search performance for a perfect binary tree – albeit with additional overhead associated with the reorganisation as data is inserted into the structure.

For the purposes of caching, the search key is some representation of the decision variables – thus it is necessary to search the cache to determine whether a particular combination of decision variables has been encountered before and, if so, to return the result of the evaluation function without having to recalculate it.

In addition to the tree data structure itself, a first-in, first-out (FIFO) queue, implemented as an STL *deque* is maintained in parallel, which allows the cache to identify which of the entries in the binary tree is oldest and to remove it when the maximum size of the cache is exceeded. The core cache functionality is represented thus:

```
if (findInCache(organism->genome()))
 return cachedFitness(organism->genome)
else
 {
  organism->evaluateFitness();
  addToCache(organism->genome(),organism->fitness());
 }
```

Figure 4-22: Pseudo-code for cache search logic

4.4.1.1. Multi-tier cache

The principal constraint on the performance of the Red-Black binary string cache is the number of comparisons that have to be undertaken in order to locate – or otherwise – a cached record in the tree. By creating a multi-layered or "tiered" cache, which has variable size layers, the most recently found solutions can be stored in smaller caches which are searched first whilst older solutions may be found by searching the larger, lower tiers. Figure 4-23 presents a schematic of the cache arrangement employed.

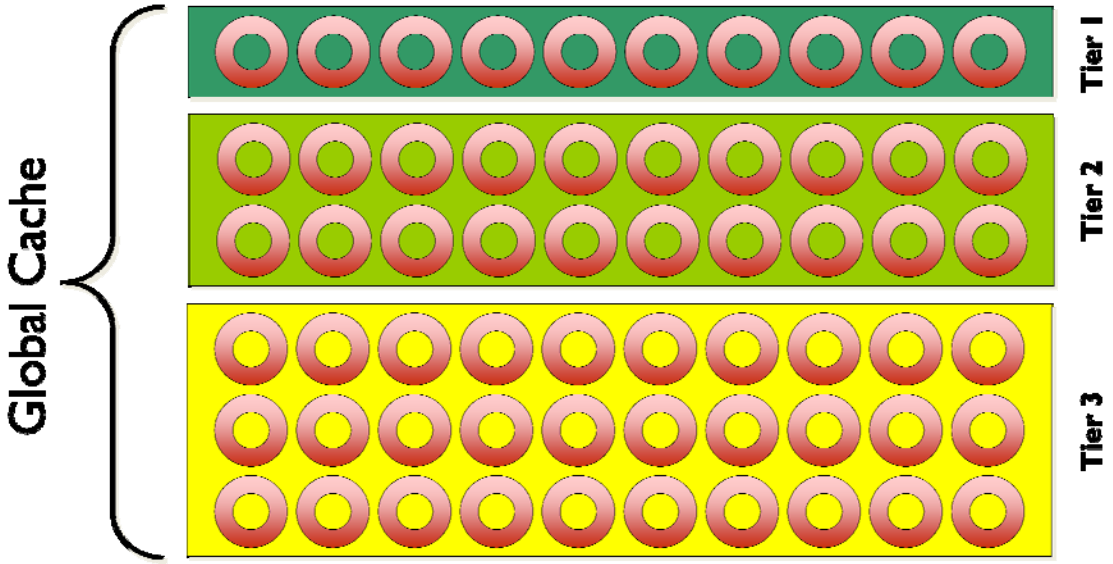

Figure 4-23: Tiered Cache

Instead of the oldest solution being deleted when the maximum cache size is reached, it is instead demoted to a lower, larger tier. Similarly, when a solution is located in one of the lower tiers it is promoted up a tier so that, gradually, the most commonly encountered solutions percolate up through the tier structure whilst the rarely encountered solutions move down and ultimately are removed from the cache altogether when newer solutions are added to the cache.

4.4.2. Judy Tree Cache

Whilst these Red-Black binary structures examined thus far are memory efficient in terms of the amount of memory required to retain each solution, they experience increasingly detrimental performance as the size of the cache increases – requiring a large number of memory access to determine whether a solution exists in the cache and, if so, to retrieve its fitness value. Two related data structures, the Digital Tree (also known as a Trie – pronounced "try"- Knuth, 1997b) and a derivative, the Judy Tree (also known as a Judy Array) offer the potential to significantly improve the performance of genetic algorithm caching as well as being suitable for archiving solutions – dependent on available memory.

The Digital Tree structure can be thought of as a *n*-way tree. The Digital Tree in Figure 4-24 shows a 2-way representation for storing each of the eight permutations of two letters (A and B) in a sentence three characters long (i.e. AAA, AAB, ABA, ABB, BAA, BAB, BBA and BBB). Unlike the binary tree representation of the same dataset (see Figure 4-26) it can be seen that the individual nodes do not contain the search key itself, rather one

component of it. In this fashion, only the leaf nodes of the tree will contain an associated data as the intermediate and root nodes represent incomplete "sentences" or paths to the search key.

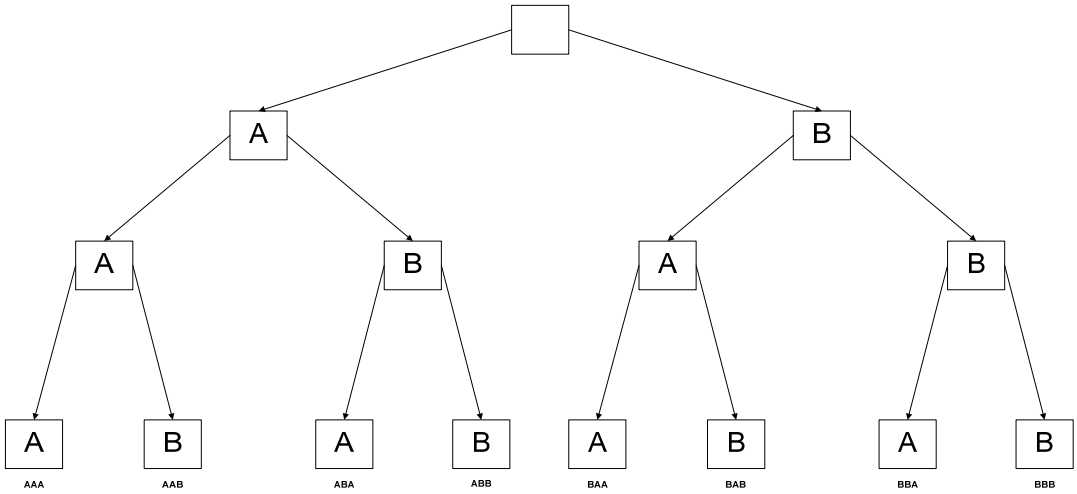

Figure 4-24: Two-way Digital Tree (trie)

Aside from the rationalisation of the manner in which the key is stored in the Digital Tree, it further differs from the conventional binary tree by not being restricted to two way operation. Figure 4-25 shows a Three-way digital tree populated with each of the permutations of three letters (A, B and C) in a sentence two characters long (i.e. AA, AB, AC, BA, BB, BC, CA, CB and CC).

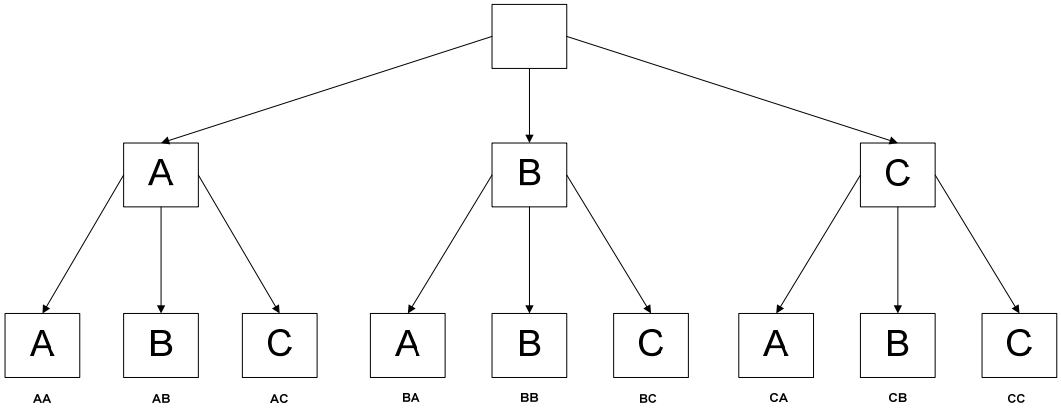

Figure 4-25: Three-way Digital Tree (trie)

Judy Trees are an implementation of a type of multi-way tree developed at Hewlett Packard's UNIX Software Enablement Laboratory (Hewlett Packard, 2001) in the early 1980s. The name "Judy" not only encompasses the data structure implementation itself but a range of optimizations intended to maximize the benefits of CPU caching to the search process. Until 2001, Judy was retained as an internal, commercial secret but has now been

released as a public library (Silverstein, 2002) for maintaining large, efficient in-memory structures and is a particularly appropriate representation for caching applications in genetic algorithms.

Judy implements a structure similar to a Digital Tree in that it ordinarily decodes one or more 8-bit digits in its key (i.e. a 256 way digital tree). However, unlike a Digital Tree where each node must have the same number of children, each node need not be a 256-way vector to store the possible values. Judy has the ability to change dynamically the representation employed by each of the nodes depending on the number of elements that it contains. This changing representation means that Judy may elect to represent part of the cache in one of three ways:

- as a simple array containing pointers to child nodes, termed *"uncompressed"* which is useful if that part of the tree is nearly or fully populated (as in Figure 4-27).
- as a *"bitmap"* which contains 256 bits, each representing whether the corresponding child is populated or empty, interspersed with 8 pointers to 8 ordered lists of up to 32 next-level pointers each which are created as they are required by the contents of the bitmap. This is the second-most memory efficient arrangement.
- as a *"branch"* which contains a count of how many children are populated, enumerates them and lists pointers to them. This is the most memory-efficient arrangement and is used when the population is sparse.

In this fashion, the Judy Tree achieves much better memory usage than a conventional digital tree. This makes the representation ideal for storing complex data, which are sparse in nature such as GA solutions in which only a minute proportion of the search space is actively explored (Goldberg, 1989).

In comparison to binary trees, which have (at best) a search lookup time of O (log n), Digital Trees achieve O ($\log_m n$) – where m is the number of significant digits in the search key. Insertion into a digital tree is also rapid compared to a binary tree, as at most m comparisons will be needed to insert an element into the tree. The Judy tree maintains the search performance of the Digital Tree, being at most O ($\log_{256} n$). In addition, there is no need for either a Digital Tree or Judy Tree to undertake any form of balancing which is an expensive operation for a binary tree and its related forms.

The Judy data structure is characterised as being opaque, in that the end user need not know anything about the storage mechanisms that are being used in the algorithm – nor have to be concerned about initialising the data structures. They are highly scalable in terms of memory consumption as memory use scales with the number of items in the array and does not require the pre-allocation of large data structures as would be the case with a conventional array structure. One of their principal advantages for caching applications is that they are very efficient in terms of performance through the implicit compression of the search keys (which also reduces its memory requirement).

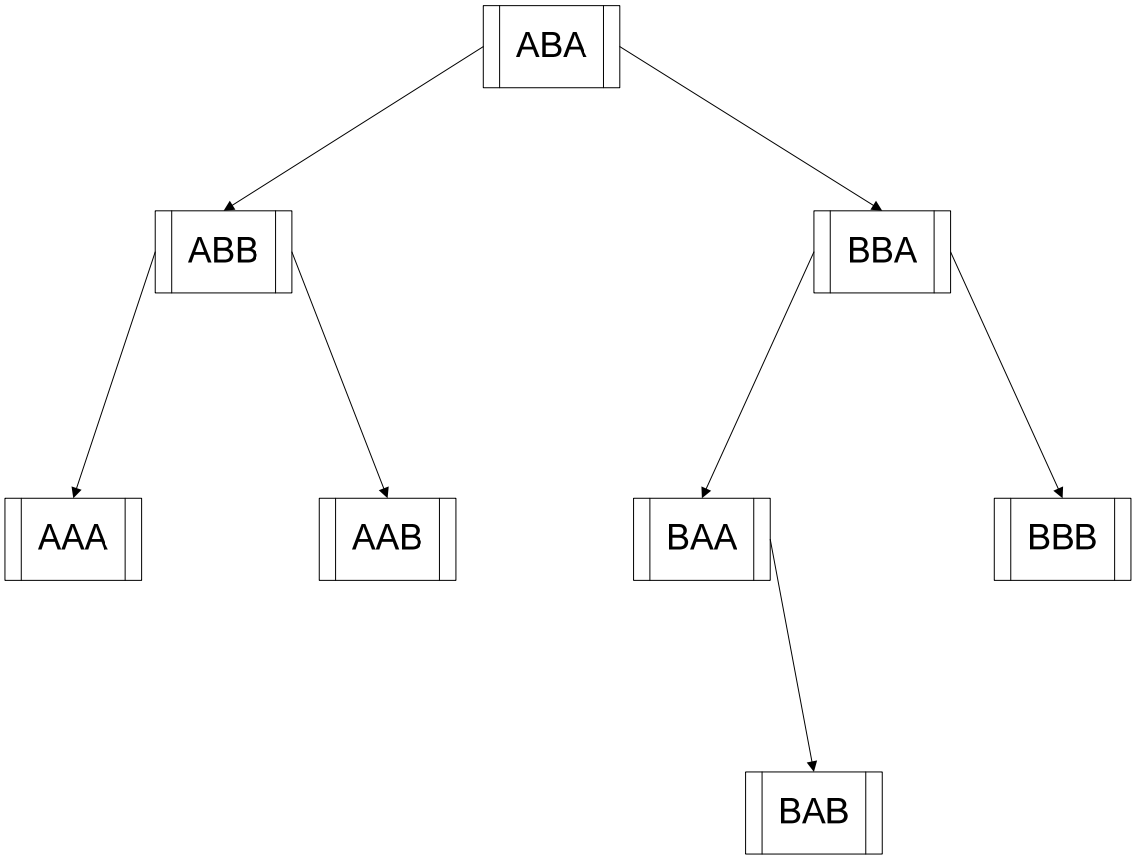

Figure 4-26: Example Binary Tree representation

Figure 4-26 shows a binary tree that has been populated with the eight permutations of two letters (A and B) in a three character sentence. It can be seen that to store these search key values in the tree incurs a cost of $3 \times 8 = 24$ characters. The equivalent Judy Tree representation is shown in Figure 4-27 where each node in the tree is itself a one-dimensional array. The Judy Tree algorithm decomposes the search key into its constituent characters and only stores at each node the minimum information to represent the path. In this fashion this representation requires only $2 \times 7 = 14$ characters to store the same data.

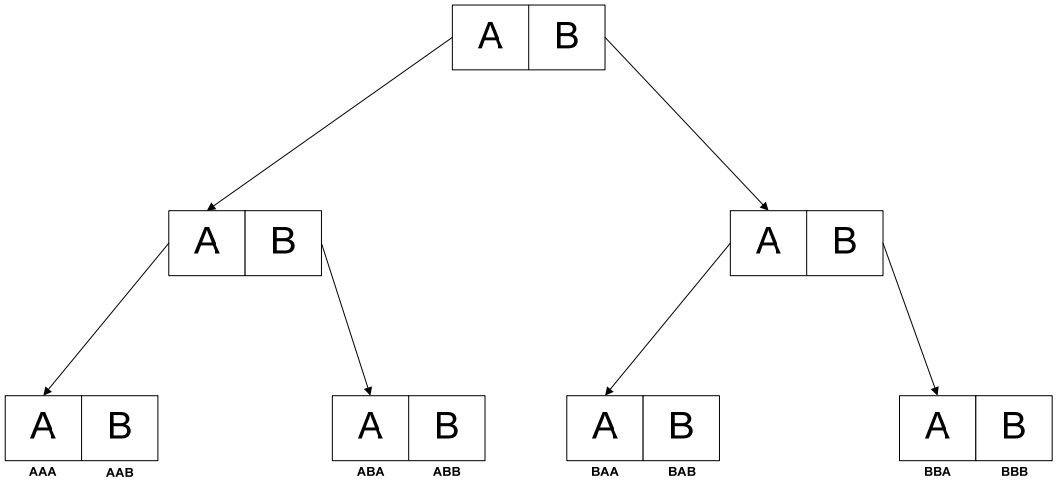

Figure 4-27: Example Judy Tree representation demonstrating implicit compression of search key.

Whilst such an improvement may not seem particularly significant, when considering much longer strings as search terms, such as the chromosomes indexed by the caching, it can be seen that the Judy representation will offer a significant improvement in the memory footprint of the cache. It should also be noted that even for this very simple example, the maximum number of comparisons that a Judy Tree search could require to find a given value is three – one less than that of the binary tree in this example. Furthermore, each of these comparisons for the Judy Tree (and Digital Tree) is of only one character whereas for the binary tree, the entire search key is compared with the value in each node traversed – a significant difference if long chromosome keys are being cached.

4.4.2.1. Example

As Judy Trees offer significant advantages over Digital Trees without any disadvantages, their application to GA caching will be considered here using as an example appropriate to a hydroinformatic, genetic algorithm application: the representation of the chromosome applied to the New York Tunnels optimization problem (Schaake & Lai, 1969). This problem, described in detail in Chapter 6.1.2, is a hydraulic reinforcement problem in which 21 pipes may be duplicated with one of 15 commercially available pipe diameters in order to meet certain pressure requirements across the network. Accordingly, New York Tunnels employs a chromosome of 21 genes – each of which can represent 15 different pipe diameters as well as a sixteenth, "do nothing" option – Figure 4-28 represents such a chromosome:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
15	2	6	12	7	9	6	0	12	0	10	0	1	0	4	2	0	8	11	4	9

Figure 4-28: Example New York Tunnels chromosome

A digital tree is characterized by the fact that the key of an element in the tree is decoded one byte (or one digit) at a time. In this way, a tree structure is developed that has the same depth as the number of decision variables in the chromosome – thus the maximum number of comparisons that must be made to determine whether an particular chromosome for this problem has been encoded into a digital or Judy Tree is 21. Figure 4-29 illustrates a portion of a digital tree structure for storing the representation of the chromosome in Figure 4-28 (*ptr* is used as an abbreviation for "pointer").

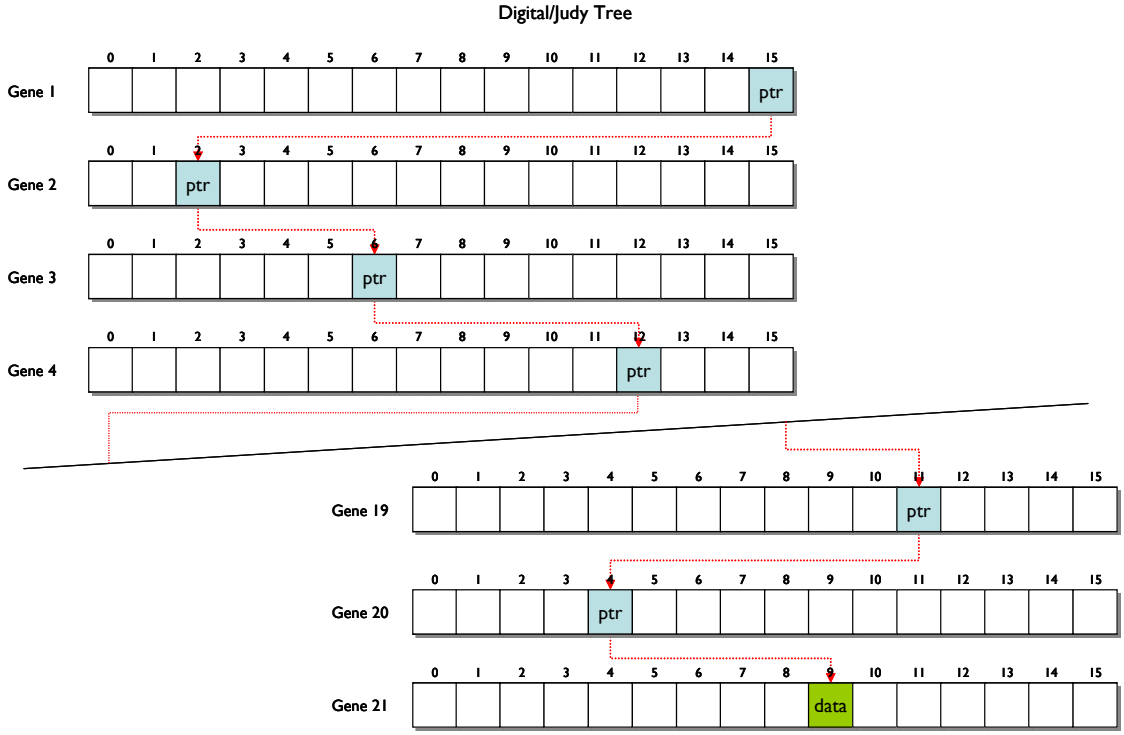

Figure 4-29: Digital/Judy tree implementation for New York Tunnels chromosome

In effect, this view is a hyper-dimensional slice through the Judy Tree data structure. If fully populated, the different levels of the Judy Tree would have vast numbers of nodes according to the relationship:

$$n = 16^{l}$$

where n is the number of nodes on level l. Given that a Judy Tree structure like this would have 21 levels, one for each of the pipes in the optimization, this is an unrealistic amount of data to be handling. However, considering an evolutionary optimization that has encountered 100,000 unique solutions in optimizing this problem, we can compare the

memory footprint and the maximum number of comparisons that may be required between the Judy Tree and the Red-Black binary tree – as seen in Table 4-5.

Representation	Memory Requirement	Mean Search Comparisons
Judy Tree	127 kilobytes (see below)	21
Red-Black Binary Tree	$100{,}000 \times 21\ bytes = 2\ Megabytes$	$\log_2 16^{21} = 84$

Table 4-5: Comparison of Red-Black Binary Tree and Judy Tree cache requirements for New York Tunnels Problem (100,000 solutions)

Estimating the actual memory requirement for the Judy Tree for a given optimization *a priori* is difficult, given that it is dependent on the compression that can be achieved by encoding the chromosome's gene values in the tree. The worst case would occur where there is as little genetic commonality between the chromosomes as possible. In this problem, this would result in a memory requirement of around 127 kilobytes – still a significant saving over that required by the conventional binary string.

The final data pointer is used to point to the data to be associated with the cache record. In this application, this record will contain information about the fitness of the individual so that it need not be evaluated again. However, it may also contain statistical information for stochastic optimization routines (see the following Chapter 4.5 for an example).

4.4.3. Experimentation

To validate the performance of the solution caching a series of experiments were undertaken on a benchmark problem called the Generalized Assignment Problem (Chu & Beasley, 1997). This class of problem involves a number of workers, "agents" each of whom has a finite workload limit and an associated cost per unit work. The optimization seeks to allocate "jobs" to the agent of varying workload on a least-cost basis. This optimization was undertaken using a simple, steady-state GA – albeit without the heuristic extensions to the Genetic Algorithm that Chu & Beasley (1997) employed to good effect.

Relative to evolution algorithm applications in hydroinformatics, the computational workload of the GAP algorithm is very small – although the solution space is comparable. To simulate a more complex objective function, the processor was made to "sleep" for a number of milliseconds (termed "ballast") in addition to the computation required to compute the fitness. The results obtained from this experimentation are related in Table 4-6.

Cache Size

In the experiments below, several cache sizes are used. Here 3 caches are used increasing in magnitude each time. Therefore "10-100-1000" represents a top-tier cache of 10 solutions, level 2 cache of 100 solutions and level 3 cache of 1000 solutions.

Algorithm Comparison

To compare the solution caching with that of a non-cached run, two measures are used, run-time and evaluations saved. A run-time analysis should be more accurate in that it takes into account all of the computation required to maintain the cache, algorithm and to update the display – which can be a significant impact on runtimes. However, evaluations saved gives an important benchmark so an expected saving can be computed for different objective function evaluation times. In the following experiments, the cached individuals were compared with a *theoretical-best-case* solution. This is computed as the number of objective function evaluations divided by the ballast, to give the minimum number of seconds it would take to evaluate N solutions. This is the best case because it does not allow for any other computational load (incurred by the algorithm, the caching, or Operating System). Therefore the benchmark seen here for comparison is the strictest possible.

Results

Table 4-6 provides a comparison between runs of the 200 job/20 agent GAP problem with different cache sizes against theoretical best case given the size of the ballast employed:

Cache status	Cache sizes	Runtime (seconds)	% Runtime	50ms Ballast		
				% Runtime Saved	# Evals Saved	% Evals Saved
OFF	n/a	5,000	100.00	n/a	n/a	n/a
ON	10-100-1000	4,610	92.80	7.20	22,947	22.95
ON	20-200-2000	4,640	92.20	7.80	23,664	23.67
ON	40-400-4000	4,606	92.12	7.88	23,895	23.90

Table 4-6: Comparison of cached and best-case theoretical performance for the 200 job/20 agent GAP problem using the tiered Red-Black Binary Tree cache

Table 4-6, above, shows that - over 100,000 evaluations with a 50ms ballast - a run-time saving of almost 8% is possible against the best-case. However, the performance benefits are actually to be far higher than this as the computational disparity between the objective function and the other computation load on the system (Operating System, maintenance of the caching data structures), is not that marked. The more noteworthy figure

is that almost a quarter of objective function calls were saved during a modest run on a large problem.

Cache status	Cache sizes	Runtime (seconds)	% Runtime	125ms Ballast		
				% Runtime Saved	# Evals Saved	% Evals Saved
OFF	n/a	62,500	100.00	n/a	n/a	n/a
ON	40-400-4000	27,426	43.89	56.12	310,616	62.12

Table 4-7: Long term 500,000 evaluation comparison of cached and best-case run-times and evaluations

Table 4-7, above, shows that over a longer optimization period and a greater ballast, the caching run-time performance is close to that of the number of evaluations saved. This is because with 125ms ballast, every objective function call saved is much more crucial to the overall performance of the algorithm. Also, the caching becomes more effective as the optimization continues and the population begins to converge.

Further Results

In performing the initial experimentation above, it was noted that the caching algorithm was highly sensitive to the mutation regime employed by the algorithm and that that employed in the above experiments was unduly amenable to the caching strategy – i.e. that there was a low mutation rate which promoted the operation of the cache. To that end, the experiments were repeated with a wide variation in mutation rates. In the revised experiments, four caching strategies were adopted and compared:

- None, i.e. caching disabled.
- Tiered, the Red-Black Binary Tree cache with three tiers of 40, 400 and 4000 individuals each.
- Huge, the Red-Black Binary Tree cache with a single tier that is allowed to grow until constrained by the available memory of the system.
- Judy, the Digital Tree derived cache which is unconstrained except for available memory.

Small GAP problem (20 Agents, 100 Jobs)

Two types of mutator were used: the first is expressed as a probability that mutation will take place applied to each of the 100 genes of the chromosome in turn. The mutation rates so examined were: 0% (i.e. no mutation), 0.2%, 0.5%, 0.75%, 1%, 1.5% and 2%. The second

class of mutator is expressed as a probability that exactly one gene will be mutated. The mutation rates for this class are 100% (i.e. exactly one gene mutated every iteration), 90% and 80%. By way of a simple performance metric for a single objective optimization, Figure 4-30 shows the median values obtained over 100 runs of the optimization with each mutation rate. As can be seen from the graph, the best performing mutation regime for this problem is that of the 100% in which *exactly* one gene is mutated, per chromosome, in each iteration.

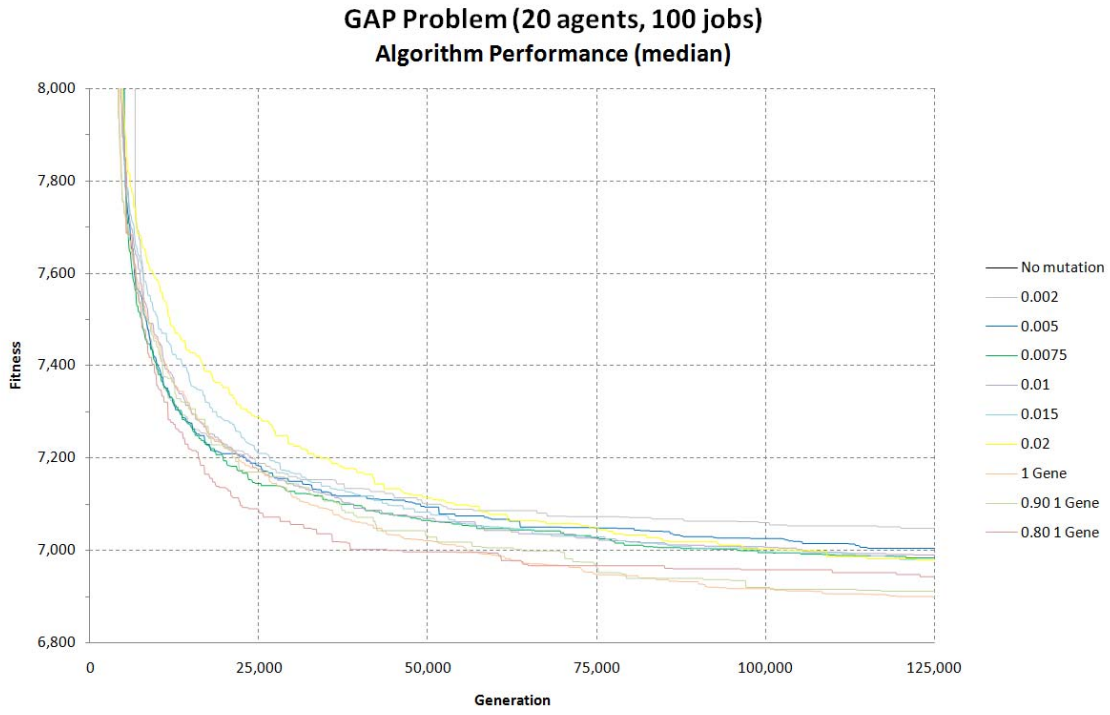

Figure 4-30: Algorithmic performance (median) for small GAP problem (20 Agent/100 Job) with variable mutation rates

Figure 4-31 graphs the effect of the different caching strategies on the algorithm runtimes with respect to the different mutation regimes. The most striking result is that of the tiered cache which can be seen to be performing worse than the completely un-cached algorithm. However, it should be remarked that here the algorithm is not having to accommodate any "ballast" to simulate a more complex objective function. Accordingly, for a more computationally intensive hydroinformatic optimization, the cache hits afforded by the tiered representation will likely improve the overall runtime performance. The other two caching strategies, the Judy Cache and the Huge Red-Black Binary Tree appear well matched for all mutation rates on this problem with the "Huge" representation slightly outperforming the Judy Tree (see Table 4-8).

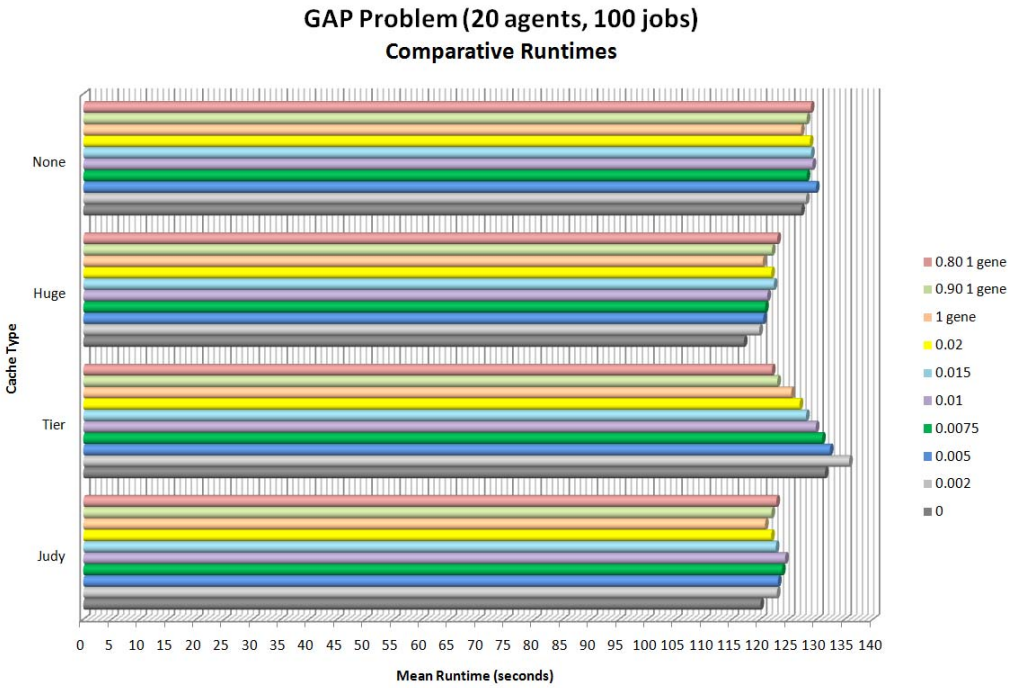

Figure 4-31: Comparative Runtimes for small GAP problem (20 Agent/100 Job) with variable mutation rates and four caching strategies

Mutation rate	Judy		Tier		Huge		None
	Runtime (seconds)	% of uncached	Runtime (seconds)	% of uncached	Runtime (seconds)	% of uncached	Runtime (seconds)
0%	120.08	94.32%	131.52	103.31%	117.14	92.01%	127.31
0.2%	123.00	95.97%	135.81	105.96%	119.88	93.53%	128.17
0.5%	123.19	94.81%	132.39	101.89%	120.58	92.80%	129.93
0.75%	123.91	96.60%	131.00	102.13%	120.91	94.26%	128.28
1.0%	124.46	96.24%	129.90	100.45%	121.30	93.80%	129.32
1.5%	122.80	95.15%	128.17	99.31%	122.45	94.88%	129.07
2.0%	121.98	94.68%	127.02	98.59%	122.00	94.70%	128.83
1 gene	120.90	94.99%	125.55	98.64%	120.50	94.68%	127.28
0.90 1 gene	122.04	95.13%	123.08	95.94%	122.12	95.19%	128.29
0.80 1 gene	122.91	95.26%	122.12	94.65%	123.08	95.40%	129.02

Table 4-8: Runtime results for caching of small GAP 20 Agent/100 Job problem with variable mutation rates

Large GAP problem (20 Agents, 200 Jobs)

As with the smaller GAP problem, two types of mutation were employed in this analysis. As before one mutator is expressed as a probability that exactly one gene will be mutated. The mutation rates for this type of mutator remain at 100% (i.e. exactly one gene mutated every iteration), 90% and 80%. The other mutator class has modified probabilities as this larger

problem has a chromosome comprising 200 genes – twice the size of the other. As a result, the probabilities of this mutator acting on each gene need to be diminished so as to retain the same approximate overall mutation rate for the algorithm. The mutation rates employed for this class of mutator are: 0%, 0.1%, 0.25%, 0.375%, 0.5%, 0.75% and 1%.

Once more the median performance of the varying mutation rates was assessed over 100 runs of each algorithm. The results are presented in

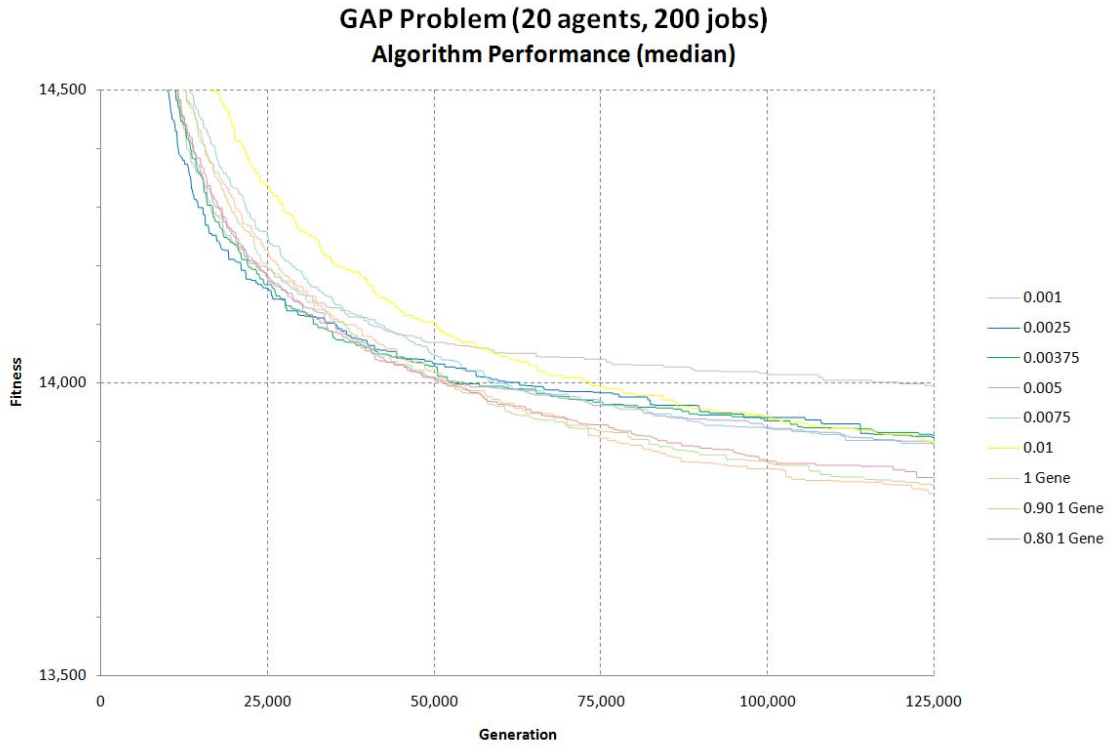

Figure 4-32: Algorithmic performance (median) for large GAP problem (20 Agent/200 Job) with variable mutation rates

In Figure 4-32, it can be seen that the mutator that changes *exactly* one gene in each iteration performs the best, once more, closely followed by the two other mutators of this type. In contrast to the smaller problem, however, the performance metrics for the large problem with respect to the caching are more varied. The results (Figure 4-33 and Table 4-9) demonstrate that the Red-Black Binary Trees do not perform as well on the larger problem size – indeed in a number of cases both the Huge and Tiered strategies which use this representation are slower than the uncached algorithm. The Judy cache continues to perform well, however, allowing for runtime savings of between 1.6 and 8.3% depending on the mutation operator employed.

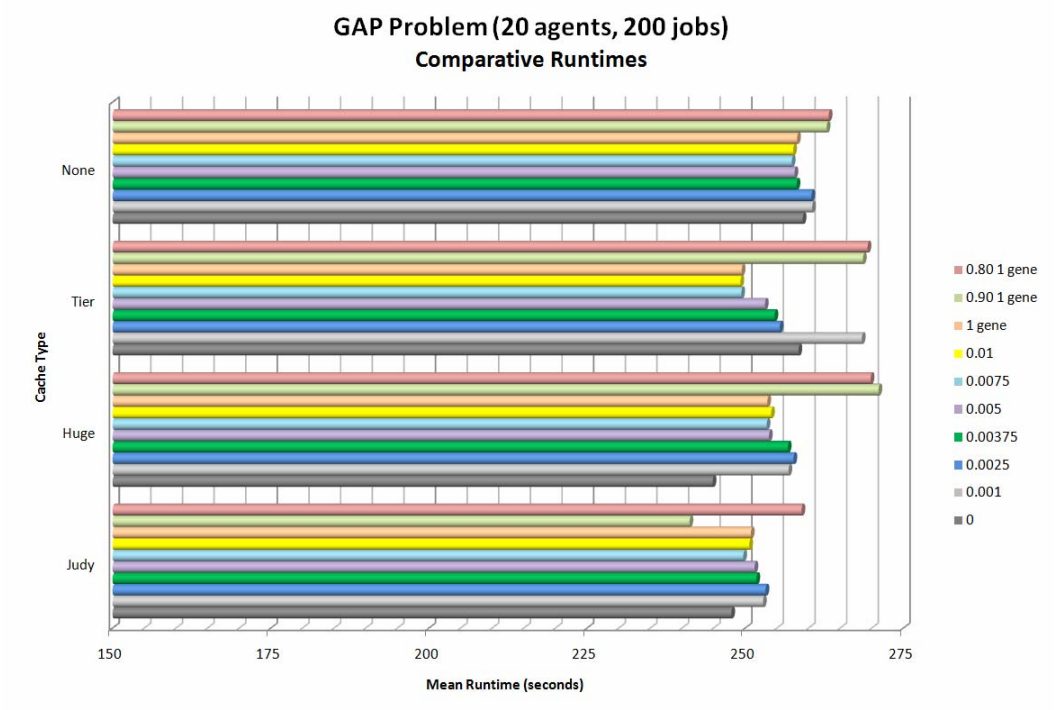

Figure 4-33: Comparative Runtimes for large GAP problem (20 Agent/200 Job) with variable mutation rates and four caching strategies

Mutation rate	Judy		Tier		Huge		None
	Runtime (seconds)	% of uncached	Runtime (seconds)	% of uncached	Runtime (seconds)	% of uncached	Runtime (seconds)
0%	247.82	95.65%	244.87	94.51%	258.44	99.75%	259.10
0.1%	252.83	97.02%	256.87	98.57%	268.45	103.01%	260.60
0.25%	253.23	97.22%	257.65	98.91%	255.47	98.08%	260.48
0.375%	251.79	97.54%	256.75	99.46%	254.65	98.65%	258.14
0.5%	251.49	97.55%	253.78	98.44%	253.13	98.18%	257.81
0.75%	249.72	97.02%	253.41	98.46%	249.40	96.90%	257.38
1.0%	250.66	97.31%	254.11	98.64%	249.26	96.76%	257.60
1 gene	250.91	97.17%	253.51	98.18%	249.46	96.61%	258.22
90% 1 gene	241.22	91.75%	271.10	103.12%	268.63	102.18%	262.9
0.80% 1 gene	258.93	98.37%	269.85	102.52%	269.36	102.34%	263.22

Table 4-9: Runtime results for caching of large GAP 20 Agent/200 Job problem with variable mutation rates

4.4.4. Conclusions

Solution caching is shown to be an effective technique for reducing the runtimes of GA applications, becoming more effective, with respective to the potential performance improvement, with the increasing complexity of the objective function and with the duration of the optimization undertaken. Because cache performance will vary from machine to

machine with respect to that of the objective function, the caching routines should, through its own internal timing, determine the complexity of the objective function and determine whether caching is an efficient strategy to employ – without interaction from the end-user. A comparison between the efficiency of the two caching regimes with respect to benchmark hydroinformatic problems is undertaken in Chapter 6.2.6 Given the relative simplicity of the objective functions employed herein, even for the larger-scale GAP problem, performance savings on more computationally intensive optimizations can be expected to be considerably greater than those demonstrated in this experimentation. Experiments demonstrate that the tiered cache is shown to be inefficient relative to a larger Red-Black binary tree arrangement whilst the Judy Tree outperforms the other representations, both in terms of performance and memory requirements, when considering the indexing of more complex chromosomes.

4.5. Non-Repeating GA (NRGA)

The caching techniques introduced above have also been used to improve the algorithmic performance of the stochastic evolutionary optimization technique, rNSGA-II introduced by Kapelan *et al.* (2005). Here, uncertain variables within the optimization are accommodated through sampling techniques and statistical aggregation of the fitness results. For example, in the model by Kapelan *et al.* (2005), each organism is evaluated a number of times for a sample of stochastic inputs and then the aggregated fitness is obtained by applying statistical analysis (e.g. mean, standard deviation, etc.). In this technique, due to the sampling procedure employed, a given set of decision variables will produce a different result every time the objective function is evaluated. Consequently, the form of caching employed above is of little utility. However, it has been observed that for a variety of problems employing the rNSGA-II, up to 4% of new solutions (dependent on population size) generated by the algorithm have already been encountered by the algorithm in a prior iteration. Moreover, it is common in runs of the rNSGA-II to find that the population contains many duplicates of a particular solution – each with its own statistical record. Reducing or preventing these duplicates would both improve the quality of the stochastic modelling by improving the accuracy of the statistical record (and therefore fitness values) as well as better maintaining the genetic diversity of the population. Figure 4-34 illustrates the operation of the "non-repeating cache" in the context of a single-objective GA. The operation of the GA, be it a single or multiple objective, continues largely as normal with the exception that, in the event of an organism being removed from the population then the

statistical data that it has accrued during its lifetime in the population is stored, along with its genetic signature, in the cache.

The use of the cache allows the algorithm to determine, in the first instance, whether one of the newly created individual generated by the algorithm is present in the existing population. If it is then the new individual is rejected and the selection and recombination process begun afresh in order to prevent the duplicate individual entering the population. If the new individual is not in the population but **is** located in the cache then it represents a solution that has been identified before but has subsequently been ejected from the population. In this event, the statistics for this organism are recovered from the cache and the evaluation process proceeds as normal – the organism having effectively been resurrected to the condition in which it was when it was removed from the population.

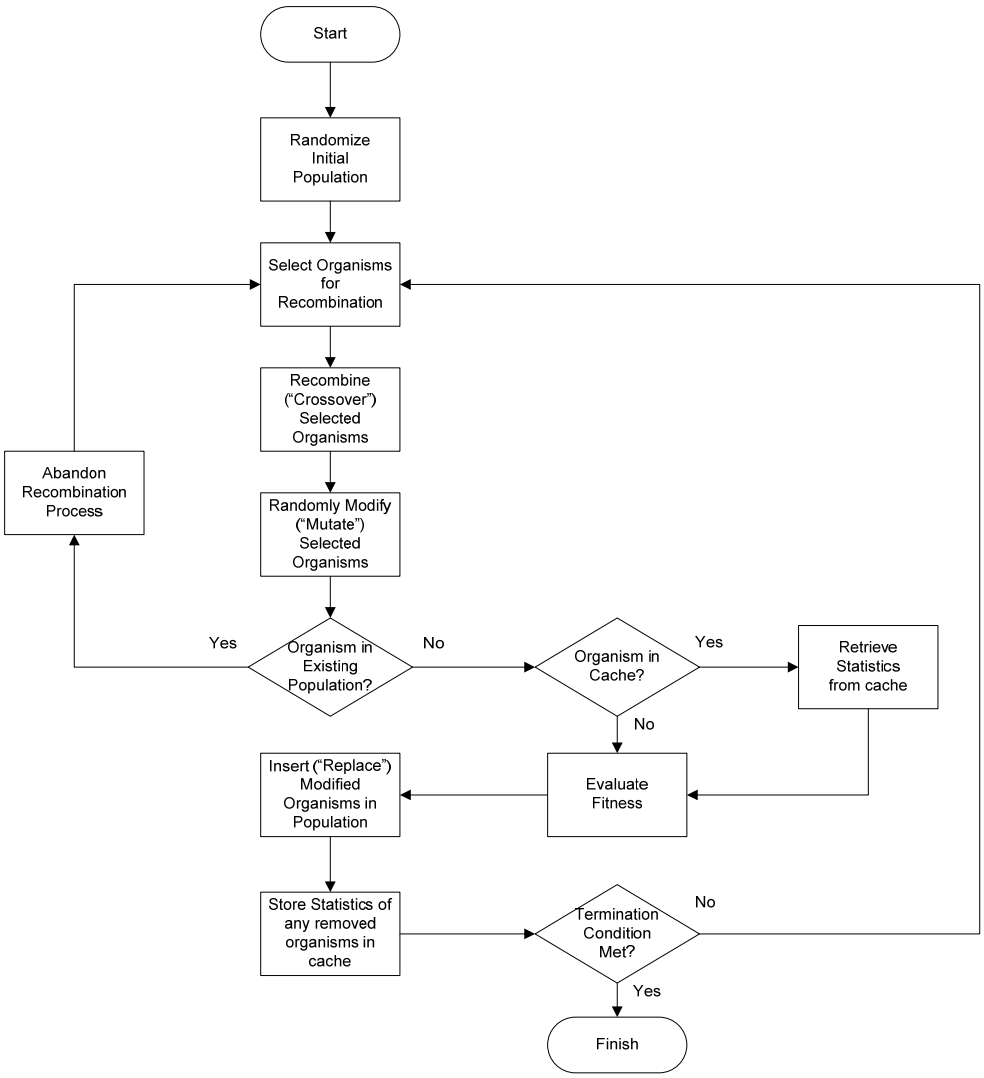

Figure 4-34: Outline flowchart for non-repeating GA

4.6. Adaptive Differential Mutation

4.6.1. Introduction

With all real world problems, and especially those with large numbers of decision variables, certain decision variables will be more important than others will. In fact, in many problems, the majority of the fitness of the solution depends on a small number of the decision variables. A case in point being the New York Tunnels problem referenced elsewhere in this chapter where a very small number of pipes (less than 25%) are critical in the determination of a feasible, low-cost solution. It is hypothesised that knowledge obtained during the evolution could be utilised to update the mutation probabilities of individual decision variables.

In this methodology, each decision variable (gene) is associated with an individual, independent mutation probability. If the modification of a set of variables is useful when the problem changes, the mutation probabilities of these variables will be increased. Throughout the operation of the genetic algorithm, the *sensitivity* of the fitness to each of the decision variables is determined by noting the change in fitness when the value of that variable changes as a result of mutation. Even though the changes in the chromosome are not made in isolation – given that many mutations may occur in a single iteration - over a large number of evaluations, the sensitivity of the objective value to each gene can be evaluated.

In addition, by recording the direction in which a mutation moved a gene value and its relationship to the changes in fitness value, it is possible to determine the direction in which future mutations should preferentially take place. Clearly, however, this trend analysis will only return sensible results if there is an underlying scalar relationship between the value of the gene and the physical property that it represents in the problem solution. In the case of the New York Tunnels example (described fully in Chapter 6.2), the gene value maps into a list of pipe sizes which varies from 0 inches (or pipe closed) to 204 inches – thus fulfilling this scalar relationship prerequisite.

In concert, these measurements can be used to indicate which genes have proven to represent the more significant variables in the algorithm run thus far and can therefore be used to drive the mutation probabilities accordingly. Consequently, a variable which is considered to have more influence on the result will be mutated more often. Similarly, if it appears to positively affect fitness when permuted in a particular direction, then mutation in this direction is accorded a higher probability of occurring. The implementation of this

mutation strategy is limited to the steady-state, single objective class of GA. Whilst there is no technical reason why this may not be extended into other forms of GA, such as the generational form, this implementation is unable to consider multiple objectives.

4.6.1.1. Sensitivity and Trend Score Implementation

Implementing the differential mutation is a two stage process: data on the performance of the algorithm has to be collected before the differential mutation can be used. This is achieved by adding routines to the recombination cycle to monitor the sensitivity of the fitness of solutions to individual gene changes. During the "learning cycle" – the first *n* iterations of the algorithm when the default mutation operator is being used – this sensitivity data is collected. Thereafter, that data is used in the differential mutation and is also updated with the results of the differential mutation.

Analysis to determine some measure of the fitness of the solution to the value of each gene was undertaken using a global repository of scores on a per gene basis:

```
struct geneticRecord
{
 double positiveScore;
 double negativeScore;
 int trendUpCount;
 int trendDownCount;
 int resultCount;
}
```

Figure 4-35: C++ structure for recording gene mutation trend score data

Of these fields, *positiveScore* represents the number of times that the gene has contributed to an improvement of the solution, *negativeScore* to deterioration in the solution. Similarly, *trendUpCount* and *trendDownCount* note, respectively, whether an increase or decrease in the variable value – relative to the best organism yet found - has been responsible for an improvement. *resultCount* is simply the number of times this gene has been changed.

During crossover and mutation, it is necessary to demark which genes have been affected by crossover or mutation from one of their parents.

After evaluating the fitness of children, the global scores for each of their genes are updated:

```
// baseScore reflects the difference between the newly created
// children and the best organism yet encountered
double baseScore= organism->fitness – bestOrganism->fitness;
for (geneLoop= 0; geneLoop < genomeSize; ++geneLoop)
{
```

```
 switch (organism->trend->gene(loop))
 {
  case down:
   if (baseScore > 0.0)
   {
    geneHistory[geneLoop]->addPositiveScore(baseScore);
    geneHistory[geneLoop]->trendDown();
   }
   else
    if (baseScore < 0.0)
     geneHistory[geneLoop]->addNegativeScore(baseScore);
   break;
  case up:
   if (baseScore > 0.0)
   {
    geneHistory[geneLoop]->addPositiveScore(baseScore);
    geneHistory[geneLoop]->trendUp();
   }
   else
    if (baseScore < 0.0)
     geneHistory[geneLoop]->addNegativeScore(baseScore);
   break;
 }
}
```

Figure 4-36: C++ code for trend scoring for differential mutation

4.6.2. Differential Mutation Implementation

The mutation operator itself is coded thus:

```
if (random()<0.5)  // produces a random number where ℝ ∈ (0,1)
{
 int numMutations= genomeSize * mutationRate;
 for (int loop=0; loop < numMutations; ++loop)
 {
  double spin= random()*totalScore;
  double runningTotal= 0.0;
  int currentGene= 0;
  while (runningTotal < spin)
  {
   runningTotal+= geneScore[currentGene];
   ++currentGene;
  }
  --currentGene;
  mutateGene(currentGene);
 }
}
else
 defaultMutator->mutate();
```

Figure 4-37: C++ code for mutation operator

The above code snippet shows that the differential mutation operator is only applied with 50% probability; otherwise the default mutation operator is used. This is important as it allows other genes, which have not been identified in the sensitivity analysis, some

opportunity to be mutated. To facilitate this form of mutation, it was necessary to update the gene classes to enable them to "increment" and "decrement" their values accordingly.

The *mutateGene* function referenced above is where the trend analysis is applied to the mutation, if required:

```
void mutateGene(int index)
{
 int trendUpCount= geneHistory[index]->trendUpCount();
 int trendDownCount= geneHistory[index]->trendDownCount();
 bool test;

 if (trendUpCount > trendDownCount)
 {
  if (trendUpCount > 0)
   test= random() < (trendDownCount/trendUpCount);
  else
   test= random() < 0.25;
  if (test)
   gene(index)->decrement();
  else
   gene(index)->increment();
 }
 else
  if (trendDownCount > trendUpCount)
  {
   if (trendDownCount > 0)
    test= random() < (trendUpCount/trendDownCount);
   else
    test= random() < 0.25;
   if (test)
    gene(index)->increment();
   else
    gene(index)->decrement();
  }
  else
   gene(index)->mutate();
}
```

Figure 4-38: C++ code for differential mutation operator

4.6.3. Cellular Automaton Mutation Implementation

Cellular Automata (CA) are a long established area of research in computer science (von Neumann, 1963) which are characterised by a population of "cells" which are able to communicate with their neighbours according to predefined rules as a response to stimulus received from their neighbours. The most famous CA is John Conway's Game of Life (Gardner, 1970), a CA which takes place on a two dimensional cellular grid. The grid is initialised with cells in an initial state, "alive" or "dead" and the simulation is then allowed to proceed through discrete timesteps for each of which the predefined rules are applied to every cell in the network. Conway's rules are very simple:

- A live cell with one or fewer live neighbours dies of loneliness.
- A live cell with four live neighbours dies of overcrowding.
- A live cell with two or three neighbours survives unchanged to the next timestep.
- A dead cell with three live neighbours is reborn.

Through these rules, intricate patterns are played out as time passes. Some initial configurations produce stable configurations whilst others die out over time. The "CA" Mutation operator is an adjunct to the differential mutation and is almost identical in operation to its sibling except it is applied to *each* gene in a chromosome in turn – rather than to a random selection – reflecting the global, simultaneous application of the rules in a true CA.

```
for (int loop= 0; loop < genomeSize; ++loop)
  mutateGene(loop);
```

This mutation option is activated as an option in the code such that there is a probability of 50% normal mutation, 25% differential mutation and 25% "CA" differential mutation.

4.6.4. Conclusions

Differential mutation with trend analysis support has the potential to improve consistently the performance of the GA subject to some relationship being maintained between the values of the decision variables and some real-world property that influences the solution. Clearly, further analysis into the scalability of this technique, along with a means to identify the appropriate point to start differential mutation, would be desirable – particularly to determine a trade-off between the overhead of maintaining the history of gene behaviour versus the algorithmic performance advantage that might be expected from employing it. If used in a mixed genome where there are many types of data, it would be sensible to highlight the genes that could be used most effectively with this type of mutator, rather than to waste evaluations modifying variables whose influence is difficult or nonsensical to track trend data for. Experiments in Chapter 6.4.7 investigate the application of this technique to a number of hydroinformatic problems.

4.7. Conclusions

The enhanced methodologies presented demonstrate several novel techniques for improving the performance of evolution algorithms. Exploiting caching at various levels within the algorithm is shown to have the potential to improve dramatically performance. This use of caching includes retaining values for binary strings as well as retaining objective function results in a solution cache. It is shown that the use of such caches has the potential to improve GA performance by ensuring that processor effort is not expended on solutions that have been encountered previously during the optimization. Further quantification of this with respect to hydroinformatics applications will be presented in Chapter 6. Furthermore, novel modifications to the mutation operator are expounded demonstrating the potential to improve GA performance by concentrating mutation operations on the genes that are determined to have the greatest impact on a solution, i.e. solutions are sensitive to changes in particular genes. To improve upon the computational performance of binary string representations, a hybridised-integer gene is presented which offers the representational benefits of the archetypal GA binary string representation combined with the performance benefits offered by integer and real-encoded genes.

Chapter 5. Distributed Evaluation for EPANET: deEPANET

5.1. Introduction

Optimization applications for hydraulic networks, particularly those involving evolutionary algorithms, are characterized by long runtimes owing to the need to evaluate large numbers of hydraulic solutions. Where conditions of uncertainty are to be considered, this issue becomes exacerbated as multiple hydraulic simulations must be performed for each solution under consideration. Distributed Evaluation for EPANET (deEPANET) is implemented as an extension to the updated release of the popular hydraulic solver toolkit (Rossman, 2000) and adds the functionality to distribute hydraulic networks for concurrent evaluation.

In order to reduce the computational runtime of GAs applied to hydroinformatic problems. Balla and Lingireddy (2000), introduce a distributed computation implementation, PCNet. This approach splits the computational load of the individual objective function evaluations across computers coupled over a Microsoft Windows-based local area network. A specimen application is described for the calibration of WDS using this approach and the performance improvement obtained from the distribution is seen to vary near-linearly with the number of computers employed. The implementation, as presented, does not however implement an automated scheme for balancing the load between the client computers attached to the network. Instead, the load balance is calculated *a priori* using the aggregate performance of the individual computers on an optimization problem. Such an approach does not allow for differences in network hardware, nor does it allow for dynamic changes in performance caused by network congestion or other loads on the client systems.

The distributed evaluation methodology presented herein permits the balancing of computational load using a simple queuing construct, which can contend with dynamic variation in computation performance and ensures that the throughput of all cooperating computers is maximized. The efficacy of this approach – even for relatively simple networks – is demonstrated with a case study on the water distribution network of a small Italian town, which is analyzed under conditions of uncertain demand. The results presented demonstrate that significant improvement in optimization performance can be realized through harnessing multiple computers in parallel in this manner. In addition, non-trivial improvements in performance are demonstrated even for single machines where advanced processor architectures with implicit parallelization are available.

5.1.1. Parallelization of Optimization

One of the key design goals for the GA library - introduced in Chapter 3 -since the beginning of its development, has been to offer the potential for accelerating optimization through the provision of parallelization techniques. Modern operating systems offer the opportunity for an application to control multiple, parallel processes simultaneously. On a conventional, single processor computer, these parallel processes merely have the appearance of running in parallel, being time-sliced automatically (i.e. divided and run consecutively) by the operating system. However, on modern Intel processors featuring HyperThreading, dual-core processors or genuine multi-processor computers there is an element of genuine parallelization involved.

Operating Systems such as those based on Linux or Microsoft Windows NT implement Symmetric Multiprocessing (SMP), which balances processor load across multiprocessors whether real or virtual. In the case of a HyperThreading processor, the potential performance gains through multithreading are minimal – relating principally to the more efficient operation of the host operating system. For multiple-core or multiple-processor systems, however, there is a tangible performance improvement, which could theoretically approach a linear improvement in speed as resources are added.

In early versions of the GA library, each Genetic Algorithm was optimized for parallelization, meaning that if one were to run *more* than one GA at a time then there would be some performance benefit (c.f. Thurley *et al.,* 1999). On single-processor platforms it allowed parallel operations to be performed which can be used to initiate several short-optimization runs (i.e. batch-run mode) for processing overnight. This is referred to as algorithmic parallelization. This implementation can be considered as generally successful although there were a number of lingering problems that proved difficult to rectify - particularly relating to the stability of inter-thread communication. Data transfer between threads requires careful management to ensure that thread processes are synchronized at the time or that threads do not attempt to access the same data structures at the same time.

Subsequent versions of the CWS library incorporate this same basic algorithmic parallelization although information about the progress of the algorithm is published in the form of custom Windows messages in order to reduce the need to synchronize threads. A number of memory-management issues were identified within the thread handling code, which were resolved.

For the most part, this algorithmic parallelization is retained purely for organizational purposes although it has the potential to be used in the construction of nested GAs for which there is no direct support or management provided by the library at this time.

The most recent evolution of the library incorporates the single thread per algorithm approach and additionally targets the execution of the GA objective function in a more piecemeal fashion.

On each execution of an objective function, a thread is created which handles that execution and is automatically terminated on the completion of that function. This behaviour is limited to creating a number of threads up to the number of processors within the system. The number of Objective Threads created is limited in this way as multiple threads of this type on a single processor offer no advantages over sequentially processing them save for allowing the Operating System to reallocate dynamically the processor responsible for the thread.

In breaking down the evaluation operations to this extent, the library gains the advantage of implicitly accelerating, on multi-processor machines, the processing of single-algorithm applications that are by far the most common.

The provision of multithreading support for Genetic Algorithms, whilst providing substantial performance advantages on multiprocessor platforms, is not without its pitfalls. Foremost among these regards the routines detailed to evaluate the objective function. In the above examples, the objective functions are simple mathematical routines that are easily coded directly into the organism. Other optimization applications, including those of water distribution system design, use an external solver application to evaluate the hydraulic state of the network. If such a system is to be used with multithreading then there are two techniques that can be used to accommodate the external solver:

A mutex (mutual exclusion) can be applied to the external solver, which allows use by a single thread at a given time - thus negating many of the advantages of the multithreaded approach, particularly given that the solver is likely to be handling the most computationally demanding part of the optimization.

Multiple instances of the solver application may be created with the same network data. For efficiency, the solvers should only be instantiated once and not with the creation of each new thread.

Whilst the first option is wholly undesirable, the second in many applications may be unobtainable. Many external applications may refuse to start multiple instances, in other cases there may be unforeseen interaction between instances - something that developers rarely examine. Therefore, in this thesis, a tool is presented, deEPANET, which embeds the hydraulic solver into a server-side application of which many instances may be instantiated on an individual machine or distributed amongst a network of cooperating machines.

5.2. Implementation

Underpinning deEPANET is the OpenNet library (Morley *et al.*, 2000- see Appendix A) which is employed as a solver-independent hydraulic model within the software environment. In this fashion, the mechanisms employed by deEPANET for distributing and solving hydraulic networks are made entirely opaque to the developer of an application – modifications made to the OpenNet representation are transparently reflected in the underlying EPANET model. A detailed description of OpenNet and its capabilities can be found in Appendix A.

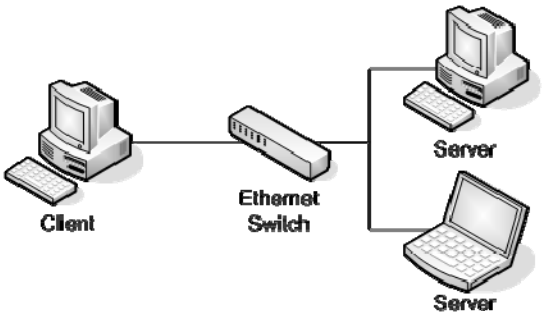

Figure 5-1: Typical PC network configuration for deploying deEPANET

deEPANET uses a client-server methodology to distribute solutions to remote computers or to a local computer using a loopback network address. The standard Internet networking protocols of TCP/IP (Transmission Control Protocol/Internet Protocol) and UDP (User Datagram Protocol) are used to implement a client-server protocol for distributing EPANET networks, partial networks and solver results across a local area network as illustrated in Figure 5-1.

The deEPANET server application can function either as a standalone, conventional Windows application or as a Windows Service. The Windows application provides an interface where the progress of optimizations may be monitored by the user as well as providing full control over the configuration of the application. deEPANET is only available

to remote users when the application is running, allowing the owner of the computing resource to determine whether and when deEPANET is permitted to run. By contrast, the Windows Service form runs in the absence of a conventional user interface. The deEPANET service is started automatically when Windows is started and, as such, does not require a user to have logged into the system. The deEPANET service can be configured through the Windows Control Panel where a simple interface is exposed which allows the configuration of all the options that are available in the conventional Windows application. In addition an option is available to restrict the times at which the deEPANET service will accept solution requests from remote computers.

Upon initial connection to a deEPANET server, the client software will normally upload the base EPANET input file that will be operated upon. It is necessary to transfer this information as an input file as the EPANET Toolkit DLL does not contain the necessary functions to configure programmatically a network, i.e. to dynamically modify the network topology necessary for running a simulation.

Having uploaded a base network to the server, a client may request that the network be solved, i.e. to perform a hydraulic simulation run. The solve process is broken down into three stages: Configuring the hydraulic network; performing the hydraulic analysis and returning the results.

The configuration of the hydraulic network is done in a sparse fashion so as to limit the amount of network traffic generated. A list of network element identifiers plus the parameters to be changed, and the new settings, are passed from the client to the server – rather than sending *en masse* all of the network specification.

In a similar manner, a list of the results required from the analysis is also passed from the client to the server. In this fashion, the server need only return the absolute minimum of data that the client is interested in. This is particularly important for large networks where an optimization may only be focusing on a few critical nodes for analysis or, as in the case of the example case study in 5.4, where there are a large number of returned data values because of the resolution of the analysis undertaken.

One novel feature of deEPANET is that it maintains a queue of solutions waiting to be performed. The solutions in this queue are distributed to servers on a first-come first-served basis. This ensures that faster computers are not hamstrung by slower servers on the network as they are able to make more frequent requests of the queue in contrast to the methodology of Balla and Lingireddy (2000) who determine which computers will be given a

higher workload *a priori*. This approach does have a limitation in that it requires that an ample supply of solutions to be queued for optimal performance. Conventional steady-state Genetic Algorithm (GA) applications are less well suited for this as they generally produce a pair of solutions for evaluation at a given time. However, generational GAs are more appropriate for this as they can queue *population_size* individuals for evaluation at a time. The problem presented in the case study related in Chapter 5.4 involves stochastic sampling coupled with a genetic algorithm, which can queue *population_size* × *sample_size* individuals for evaluation at a time.

5.2.1. Robust networking

Any distributed computing application has to accommodate the possibility that the failure of a remote computer or network component may interrupt the flow of data. deEPANET accommodates this on the client-side by analyzing the frequency of data returns from the connected servers. If a server fails to return a result within three times the average time that it has previously calculated results in then that server is asked to cancel its operation and the solution is tasked to another server – whilst the original server is given a virtual "black mark". After exceeding a user-defined number of "black marks", a server may be disconnected from the application.

For the ease of developing new applications using the deEPANET library, functions were added to allow for the automated search of the LAN for available servers. The search facility is limited to a LAN as it will not be able to traverse a switch and so the deEPANET client also offers the facility to add manually an IP address to send a message to. Accordingly, the deEPANET server implements a UDP listener to allow it to respond to this broadcast message; the response taking the form of a reciprocal TCP connection over the same port. The client and server negotiate the port numbers that are used in the event that either side is using any ports in the default range. The combination of the UDP listener and TCP connection is used for both the automatic and manual discovery of servers as this verifies that any manual server additions are valid and can be reached through any intermediate firewalls.

5.2.2. Advanced processor architectures

deEPANET seeks to exploit emerging technologies such as HyperThreading™ (a mechanism for optimizing instruction pipelining for multiple threads on a single processor core) and multiple-core processors to further accelerate performance. Through its

multithreaded structure and its novel ability to manage concurrent instances of the hydraulic solver, deEPANET is able to improve the performance of EPANET on standalone (non-networked) hardware with features such as HyperThreading™ and multiprocessor/multicore systems. Since both of these technologies are becoming increasingly common it makes sense to target such systems for use with optimization applications.

The ability to employ multiple processors in standard PC hardware has existed for many years – requiring multithreaded software to maximize its benefit. Yet the use of multiprocessor machines outside of the server environment has been somewhat restricted, requiring expensive motherboards and, naturally, two costly processor chips. The emergence of "multi-core" processors in which two or more processor chips are incorporated on the same silicon die has seen this situation change. Not only are these chips considerably cheaper than buying equivalent processors separately and there is no requirement for an expensive multiple socket motherboard but their presence is relatively transparent to software and hardware alike.

5.2.3. Cross platform characteristics

To provide deEPANET with cross-platform capability, the library is coded in portable C++ (with the exception of the optional user interface) and uses a portable, open source TCP/IP library to provide its network connectivity. The only difficulty with using deEPANET on Linux or other operating systems is likely to be the need to recompile the EPANET library itself for that platform. Because of the inherent platform-neutral nature of TCP/IP, using deEPANET instances on different platforms in conjunction with each other would not pose any problems.

5.3. Application

The computers for the test environment were selected to form a representative cross-section of the type of computers that might be found in a normal networked environment. Represented in the test environment are high-end and mid-range processors from both the dominant PC processor manufacturers, Intel and AMD.

Computer	Processor	Memory	Network
A	1.8 GHz AMD Athlon 64	1 GB	Gigabit
B	1.6 GHz Intel Pentium 4†	512 MB	Fast
C	3.0 GHz Intel Pentium 4HT	1 GB	Fast
D	2.2 GHz AMD Athlon 64x2	1 GB	Gigabit

†The processor in computer B is a 3.0 GHz Intel Pentium 4 that has been de-rated to operate at 1.6 GHz.

Table 5-1: Hardware specifications of test environment computers.

The computers are connected via a Gigabit Ethernet switch to allow the two Gigabit equipped computers to be connected at the highest speed (1,000 Mb/s). The switch employed can be forced to operate in a "Fast Ethernet only" mode (100 Mb/s) which allowed the influence of network speed on deEPANET's performance to be investigated.

5.4. Case Study Network

The case study relates to the water distribution network of the small Italian town of Piedemonte San Germano (Tricarico *et al.,* 2005) and comprises 45 pipes serving 33 demand nodes, arranged so as to form 12 loops, gravity-fed from a single reservoir. A full description of the network topology and characteristics may be found in Chapter 6.3.4.

Whilst the scale of this network is relatively trivial, the demand conditions for this system are extreme in that the network model contains demand data obtained from the real-world network for 24 hours at 1-second intervals and resampled to 1 minute intervals. Tricarico *et al.* (2006) analyzed this data using the robust Non-Dominated Sorted Genetic Algorithm II (rNSGA-II) developed by Kapelan *et al.* (2005).

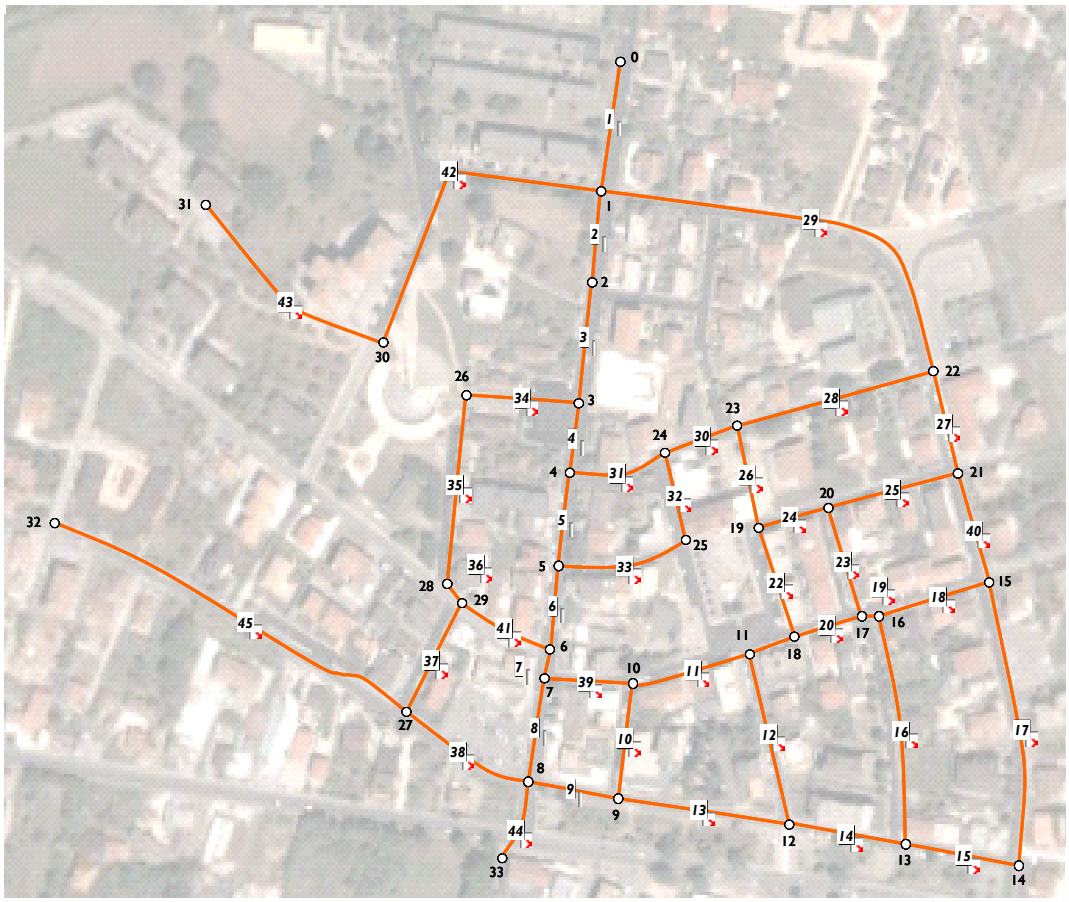

Figure 5-2: Topology of Piedemonte San Germano Case Study Network

A key characteristic of the rNSGA is its use of stochastic sampling techniques to obtain a measure of the robustness of a given solution under conditions of uncertainty – in this instance the network demands. This approach requires significantly more simulation evaluations than ordinarily required by a conventional, deterministic GA optimization with between 5 and 50 *additional* hydraulic simulations being required for each solution evolved by the algorithm, as well as up to 100,000 additional simulations performed on the best solutions as a post-processing exercise. The use of such a sampling technique makes this type of algorithm ideal for use with deEPANET given that many solutions may be queued for evaluation for each iteration of the algorithm.

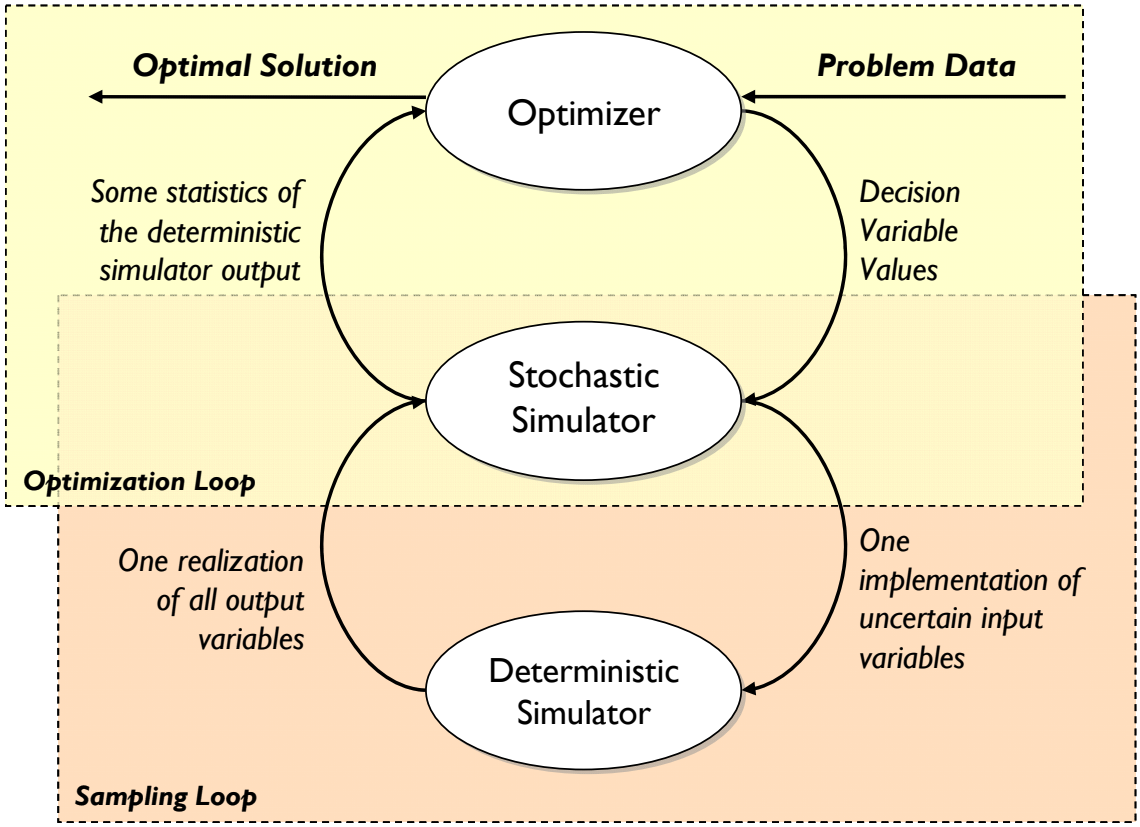

Figure 5-3: Logical Structure of Stochastic Optimization Software (after Kapelan, 2005)

To prevent the performance improvement being masked by any additional overheads, a simulation exercise was devised that performs no processing other than generating a network to be solved and then dispatching that network for solution and the interpreting the results returned.

5.5. Results

In order to demonstrate the potential performance gains from deploying deEPANET it is necessary to evaluate first the baseline performance of the computers in the test environment.

Computer	Performance (evaluations per second)		
	Single thread	Two threads	Three threads
A	11.97	11.75	11.24
B	3.45	3.42	3.32
C	7.07	9.25	9.09
D	13.95	25.07	25.18

Table 5-2: Baseline performance on Piedemonte San Germano simulation exercise.

Table 5-2 and Figure 5-4 show the results obtained from averaging ten runs of the simulation exercise outlined above. The simulation exercise was repeated using two and three instances of deEPANET running on the same machine to determine whether there was any benefit of doing so.

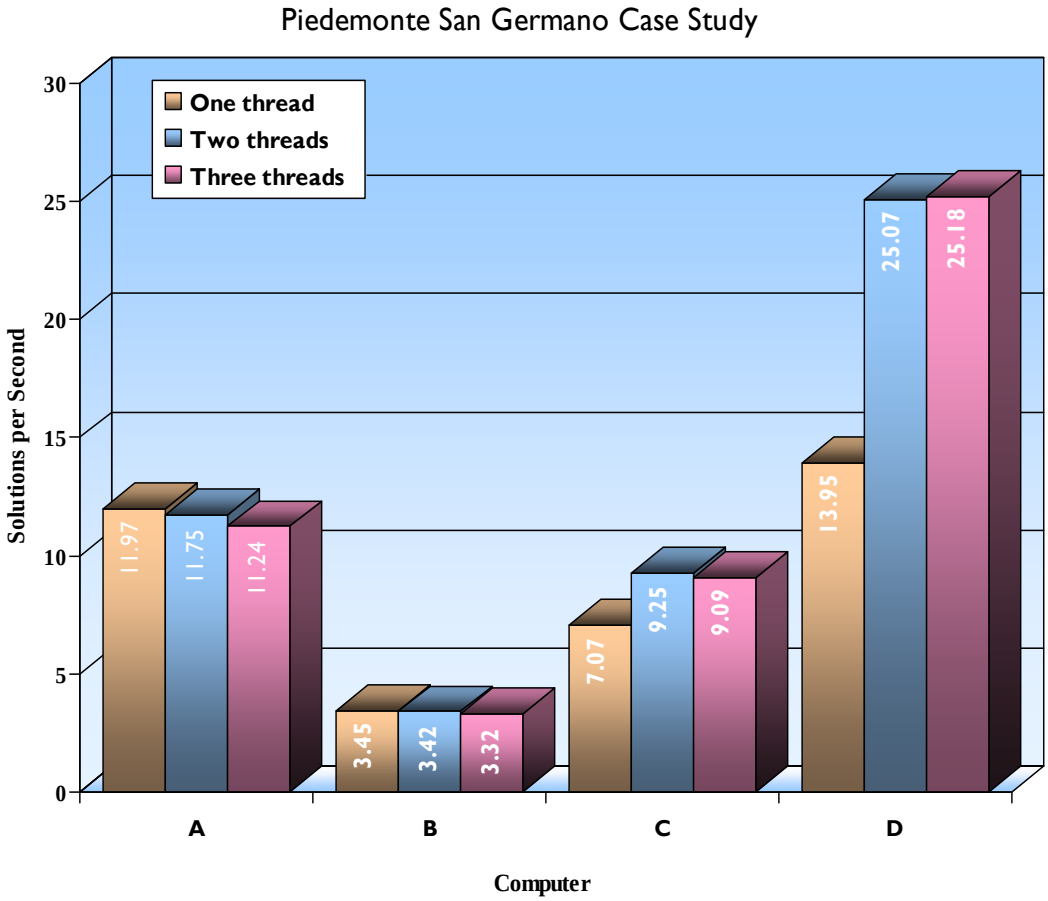

Figure 5-4: Baseline performance on Piedemonte San Germano simulation exercise.

As was expected, the availability of a second thread on the dual core machine *D* allowed it to almost double its throughput of solutions; indeed, the performance of this machine continued to improve with up to six instances being used – the reasons for which will be discussed below. What was more surprising, however, was the performance of computer *C* that demonstrated a performance improvement > 30% with the addition of the second instance – rather than the degradation which might be expected and which was shown by the other single-core processors.

To determine whether Intel's HyperThreading technology was likely to be responsible for this performance gain, the processor in computer *B* was returned to its native 3.0GHz performance and retested – the processors in computers *B* and *C* are identical save

for the latter's support for HyperThreading. Under these conditions, computer *B* achieved a baseline score of 7.09 solutions per second with a single thread and degraded thereafter with additional instances, in line with its prior performance confirming that the HyperThreading support on the processor was indeed assisting in the running of two instances of the deEPANET client.

The baseline results demonstrate that, even without employing collaborating computers across a network, deEPANET can realize significant performance improvements on standalone machines with multiple processors – be they physical or virtual.

To evaluate the performance of the software it was necessary to nominate a single computer to act as the client for which the other computers would serve results. Tests demonstrated that it was most effective for the quickest machine to act as the client in order to ensure the responsiveness necessary for distributing and collating the network solutions in a timely fashion. To this end, computer *D*, the dual-core AMD machine was selected for this purpose. In addition to serving the other computers, computer *D* also continued to act as a server in its own right, returning results to its own client.

Computer	Baseline score	Distributed score *(single threads)*	Distributed score *(dual threads)*	
A	11.97	11.81	11.82	
B	3.45	3.69	3.54	
C	7.07	6.17	*Thrd. #1* 4.30 *Thrd. #2:* 4.35	8.65 *(combined)*
D	13.95	11.10	*Thrd. #1* 11.52 *Thrd. #2:* 11.52	23.04 *(combined)*
Totals	**36.44**	**32.77**		**47.05**

Table 5-3: Results obtained from running single threads on each of the computers and dual threads on the multiprocessor computers.

The results in the *single thread* column of Table 5-3 illustrate a drop in performance of around 10% when compared to the baseline score representing the overhead of transmitting the data across the network. Much of the drop in performance is attributable to computer *D,* which is now responsible for distributing solutions to all of the other computers. It should be noted that a direct comparison between the baseline results and the distributed results is not entirely valid as the baseline results would represent individual optimization runs operating on separate computers whereas the results from the distributed arrangement contribute to a single optimization run – potentially a more useful scenario when considering multi-objective optimization algorithms.

It has been shown that running two threads on the HyperThreading and Dual-Core machines greatly improves their performance and the results obtained by adding a second thread to these machines is shown in the *dual threads* column of Table 5-3. As can be seen, the addition of the second threads on the multiprocessor machine demonstrates much the same performance improvement that it did in the standalone case.

An analysis of processor utilization at this point revealed that each of the computers was failing to reach 100% by some margin. This is due to the inherent latency in sending and receiving network messages that causes a thread to wait whilst this information is processed. Accordingly, an additional thread was added to each of the machines in order to allow this "wasted" time to be directed to undertaking the network analysis. The results from this addition are shown in Table 5-4. A further test was made by restricting the network switch that connects the computers to a maximum operating speed of 100 Mb/s – the effect of this change is also related in the following table.

Computer	Baseline score	Distributed Score (Gigabit Ethernet)		Distributed Score (Fast Ethernet)	
A	11.97	*Thrd. #1:* 6.24	12.46 *(combined)*	*Thrd. #1:* 4.57	9.08 *(combined)*
		Thrd. #2: 6.22		*Thrd. #2:* 4.51	
B	3.45	*Thrd. #1:* 1.91	3.83 *(combined)*	*Thrd. #1:* 1.65	3.29 *(combined)*
		Thrd. #2: 1.92		*Thrd. #2:* 1.64	
C	7.07	*Thrd. #1:* 2.79	9.33 *(combined)*	*Thrd. #1:* 2.83	9.23 *(combined)*
		Thrd. #2: 3.81		*Thrd. #2:* 2.82	
		Thrd. #3: 2.73		*Thrd. #3:* 3.58	
D	13.95	*Thrd. #1:* 6.39	22.51 *(combined)*	*Thrd. #1:* 6.66	22.95 *(combined)*
		Thrd. #2: 9.85		*Thrd. #2:* 6.34	
		Thrd. #3: 6.27		*Thrd. #3:* 9.95	
Totals	**36.44**		**48.13**		**44.55**

Table 5-4: Results utilizing one thread per processor (virtual or physical) plus one supplementary thread.

Further tests have shown that further increasing the number of threads on each machine does not yield further improvements in performance. Preliminary results suggest that this technique remains scalable and a network comprised of four computers with a similar configuration to computer *A* in collaboration with the server *D* have achieved in excess of 75 solutions per second on the example problem. Identifying, the extent of this scalability is difficult – being dependent both on the configuration of the problem, the network topology and the performance of the individual computers involved.

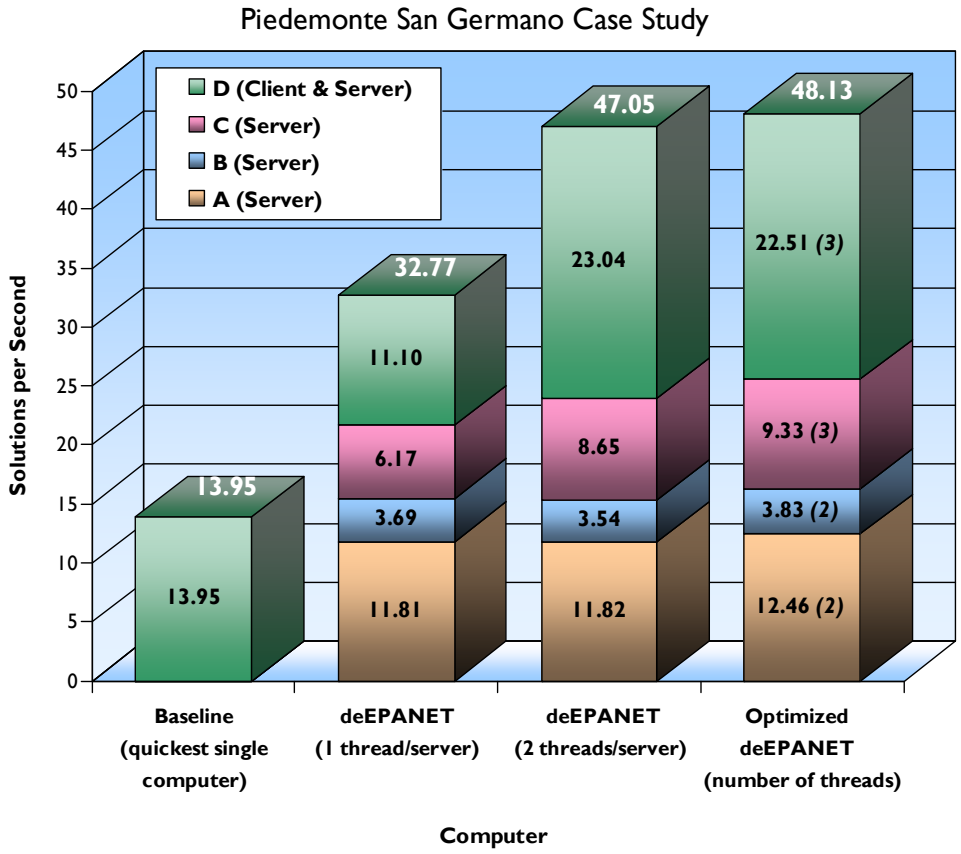

Figure 5-5: Results utilizing one thread per processor (virtual or physical) plus one supplementary thread.

5.6. Distributing Stochastic Computation

In attempting to determine the level of scalability that the application of deEPANET enjoys, the system was applied to the network topology shown in Figure 5-6. In addition to the computers used for the case study above, four additional computers have been introduced to the network. Two of these, *F* and *G* are machines similar in specification to *A* and are connected to the server *D* via a Gigabit Ethernet connection.

E is a laptop connected by an IEEE 802.11b wireless connection that has a peak speed of 11mbps (just over 1% of the bandwidth available to the Gigabit connection). The last addition is *H* which is to be found attached to the Internet via a residential ADSL (Asymmetric Digital Subscriber Line) connection around 200 metres from the campus of the University of Exeter. Despite its apparent proximity, however, the gateway between the ADSL provider and the University's SuperJANET network is in London.

Preliminary results from running deEPANET on this network were encouraging in that the additional computers attached to the Gigabit network continued the scaling trends seen previously. However, what was unexpected was the relatively poor performance of the

wireless connection and that of the WAN (Wide Area Network) connection off campus. Whilst both connections are acceptably fast for data transfer, the latency associated with both wireless networks and WAN connections meant that the transport times for small packets of data were compromised.

In an effort to reduce the implications of this – and to improve the practicability of deEPANET – a decision was made to offload the computation of the stochastic variables to the remote servers. In the original configuration of deEPANET, the client computer would generate a queue of networks to be solved by varying the stochastic variables according to a Probability Distribution Function (PDF) specific to each input variable.

A typical configuration would mean that for each solution generated by the genetic algorithm, the stochastic loop (see Figure 5-3) would run 20-50 times, producing that number of individual, slightly-differing networks to be solved hydraulically – in a distributed fashion through deEPANET. Devolving the stochastic computation to the servers entails passing the description of each PDF for each stochastic variable, in this instance the nodal demands, rather than the generated values obtained from the PDF. The server computers are then in a position to generate the stochastic variables directly from the PDF themselves. This has a twin-effect on the performance of the algorithm, as can be seen from Table 5-5:

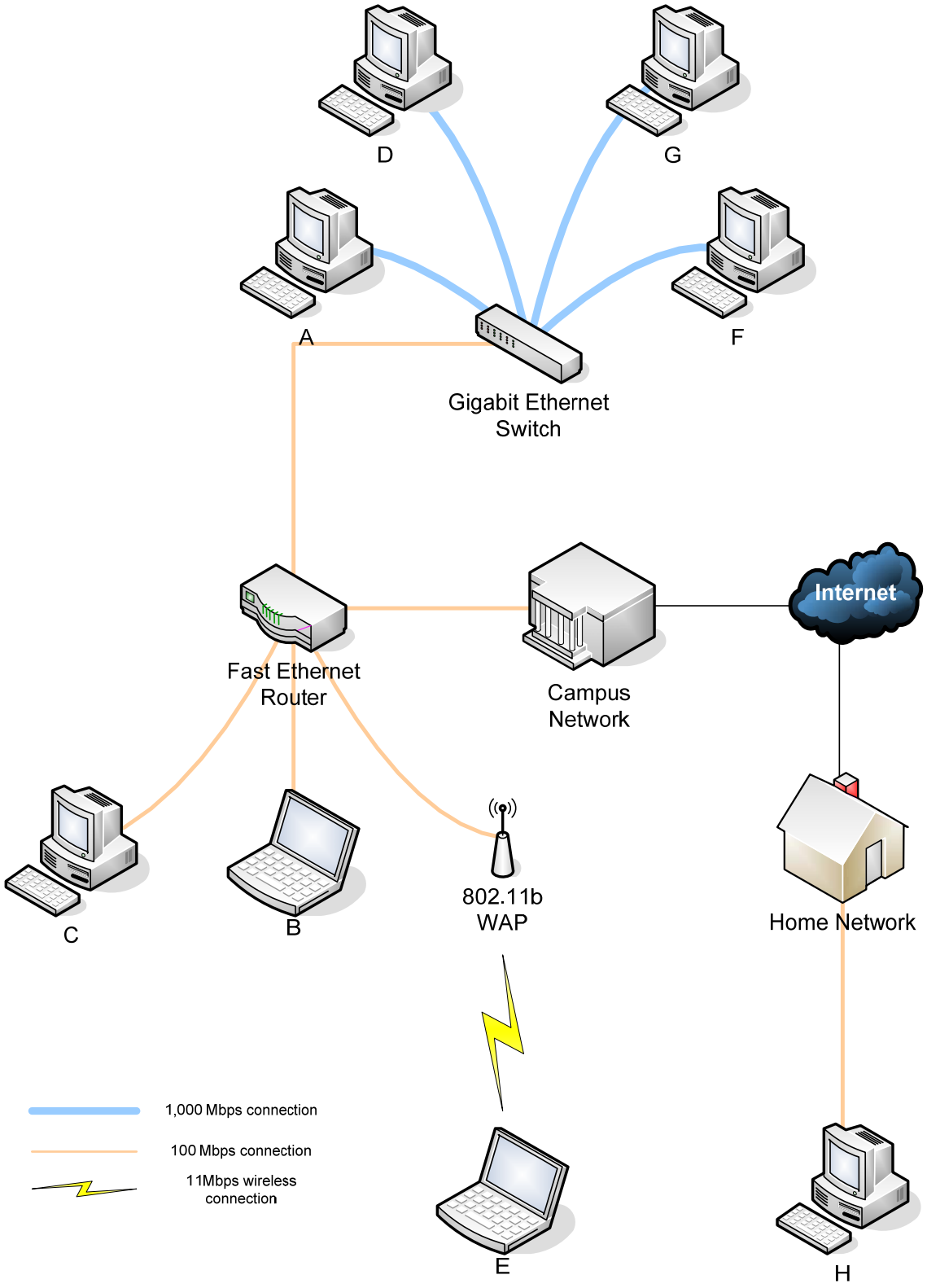

Figure 5-6: Extended test network for deEPANET simulations

The *amount* of data transferred across the network is thus greatly reduced. Each network solution only requires the transfer of the pipe configuration and the PDF

specification – rather than one set of pipe configurations and nodal demands for each sample. A saving proportionate to the number of samples can be achieved in terms of a reduction in network traffic. In terms of the results returned there are two techniques that can be applied. The server can either collect all of the results from the individual hydraulic evaluations and return them to the client, as before, or aggregate the statistics for a given solution – significantly reducing the amount of data transferred and the load on the client.

The data transfers are also *consolidated* into two transfers that occur at the beginning of the evaluation of a solution and when all of the samples have been hydraulically evaluated. This has the effect of reducing the effect of the latency on the algorithm as most of time that is wasted in deEPANET is involved in facilitating a network connection rather than actually transferring the data.

	Data Transfer *(bytes per network solution)*			**Number of Data Transfers** *(per network solution)*	**Performance** *(sample solutions per second)*	
	Outbound	Inbound	Total		using four LAN computers	using one WAN computer
Standard[1]	15,200	6,200	21,400	100	48.13	0.18
Devolved[2]	550	6,200	6,750	2	48.58	7.02
Devolved with Aggregated Results[3]	550	32	582	2	49.54	7.75

[1] sending 45 pipe diameters and 31 nodal demands and receiving 31 nodal pressures – for each of 50 samples.

[2] sending 45 pipe diameters and 31 PDF descriptions for each network solution and receiving 31 nodal pressures for each of 50 samples.

[3] sending 45 pipe diameters and 31 PDF descriptions and receiving 8 statistics for each network solution.

Table 5-5: Comparison of data transfer and performance for standard and devolved stochastic configurations (for the Piedemonte San Germano case study as before – assuming 50 stochastic samples)

Table 5-5 clearly illustrates the type of performance improvement that can be achieved when devolving the stochastic sample generation to the server machines – with the attendant benefits of reducing and consolidating the network traffic necessary. From the table it can be noted that there is a small improvement in the performance of the four, LAN connected computers. Given that this network is not badly affected by latency in the first instance, it can be assumed that this improvement can be principally be attributed to the reduction in the amount of data processing that has to take place on both sides of the network connection. The result for the WAN connected computer is more significant, however. Under the original implementation, 100 packets (not IP packets) of data are sent across the network, averaging 214 bytes in size. Under the devolved and devolved/aggregated implementations this reduces to 2 packets (550 bytes sent and 6,200

bytes received) and 2 packets (550 bytes sent and 32 bytes received), respectively. Clearly, reducing the number of packets sent has a dramatic effect on the network overheads as can be seen from the marked performance improvement witnessed for the WAN connected computer – some 43 times quicker. It should also be noted that reducing the throughput required of the network is likely to result in improved scalability of the entire solution.

One issue that arises with reducing the granularity of the elements distributed across the network is that it is less straightforward to balance the performance of servers. Before, a server that outperformed its peers undertook a larger share of the workload – which has the net effect of all of the servers returning their final contribution to an individual network solution at approximately the same time. This effect may be mitigated by selecting smaller numbers of stochastic samples where appropriate. For this reason, deEPANET is configurable as to which technique is to be applied depending on the topology of the network that is available for use.

5.7. Conclusions

The results presented demonstrate that deEPANET can significantly shorten runtimes for optimization algorithms by distributing evaluations to computers across a network and by exploiting multi-threading techniques on standalone computers equipped with virtual or physical multiprocessors.

Chapter 6. Single Objective Optimization Problems

6.1. Introduction

In order to demonstrate the applicability of the techniques outlined in Chapter 4 and 5, a number of case studies have been undertaken. This chapter introduces three problems formulated as single-objective optimization problems, which are then revisited as multiple objective optimizations in Chapter 7. For each problem, the effect of the novel methodologies, i.e. genetic representation, caching and updated mutation operators, are identified.

These single objective optimizations are formulated thus:

$$Minimize: Cost = C_{inf} + C_{pen} \quad \text{iii)}$$

$$C_{inf} = f(D_1, .., D_{N_l}) = \sum_{j=1}^{N_l} C(D_j, L_j) \quad \text{iv)}$$

$$C_{pen} = f(H_1, .., H_{N_n}) = K \cdot \sum_{i=1}^{N_n} \max(0; H_{i,min} - H_i) \quad \text{v)}$$

$$D_j \in D \ (j = 1, .., N_d) \quad \text{vi)}$$

where: *Cost* is the total cost, to be minimized, C_{inf} is the total infrastructure cost, C_{pen} is the penalty cost term, N_l is the number of links in the network for which reinforcement or installation is an option, $C(D_j,L_j)$ is the cost of the j^{th} pipe with diameter D_j (chosen from a discrete set of available diameters D) and length L_j. K is the penalty multiplier constant, H_i is the pressure head at node i (as computed by the hydraulic solver), $H_{i,min}$ is the minimum pressure head requirement sufficient to fully satisfy the demand at node i and N_n is the number of nodes in the network. N_d is the number of decision variables in the optimization.

6.1.1. Genetic Representation

In Chapter 4.2, an introductory examination is made of the effect of varying the genetic representation used for the candidate GA solutions. This section contrasts this variation and the use of heterozygous chromosomes for each of the three applications introduced above.

Each of the networks under consideration was subject to 100 optimization runs on the above basis using the different genotype representations. The results of this experimentation are presented in the form of charts which can be considered as a two dimensional form of "box-plot" (Chambers *et al.,* 1983), an example of which is shown in Figure 6-1.

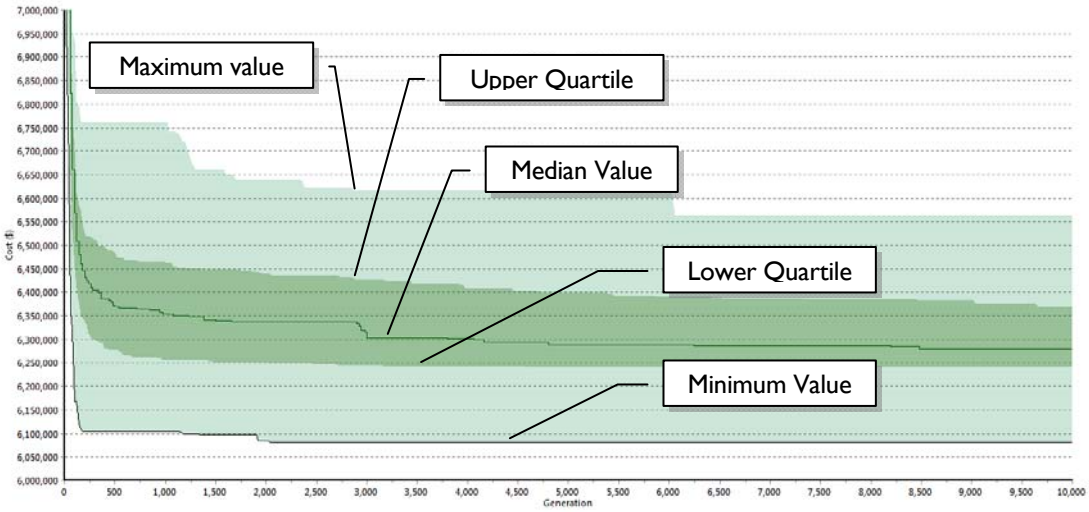

Figure 6-1: Example result graph

The results presented show the maximum and minimum fitness values of the best individual in the population throughout the lifetime of the optimization, along with the upper and lower quartiles and the median fitness. Combined as a graphical presentation, they provide an effective illustration of the algorithmic performance for a single-objective algorithm. It should be noted that in order to reduce the amount of data produced by the optimizations to a manageable level, the state of the population was sampled every 20 generations – thus the resolution of the graphs presented is limited in this respect.

6.1.2. Heterozygous Chromosomes

The New York Tunnels and Piedemonte San Germano problems examined were reformulated as heterozygous problems in which the chromosome of the solution contains not only the pipe diameter to be applied, but also the identity of the pipe to apply it to. In this fashion, it is possible to constrain the optimization by limiting the number of changes

made to the network. In this analysis, ten pipes were permitted to be changed for both problems. It should be noted that from the optimizations performed previously it had been seen that the optimal solutions for both problems were achieved with the modification of up to just six pipes in the network.

Owing to the fact that the Hanoi network is a design problem, in which each pipe *needs* to have a diameter value set, this network has not been tested with a heterozygous formulation.

6.1.3. Caching

For each of the three networks, an experiment has been performed to quantify the performance improvement that might be achieved by adopting the two caching strategies outlined in Chapter 4.4. As the integer-based representations have been shown to outperform those of the true binary strings, the caching analysis will concentrate on that representation. Caching is purely an exercise in reducing the runtime performance of the algorithm and, for deterministic algorithms at least, should have no effect on the final results of the optimization. As the binary strings have a longer and more complex structure to search for in the cache they will intuitively produce significantly poorer performance than the integer or hybrid representations. The search times for the different caches have been determined experimentally with the evaluation function of the GA disabled. This allows a good measure of the average search time of the cache. This, coupled with the cache hit statistics generated over repeated runs of the algorithm allows the computation of any performance saving afforded by using the cache. Owing to the variable chromosome representations employed by the heterozygous algorithms, the caching has not been evaluated against these models.

6.1.4. Adaptive Differential Mutation

To determine the efficacy of the differential mutation it was initially applied to a simple General Assignment Problem in which trend information is largely irrelevant. The General Assignment Problem has no scalar relationships between the decision variable values and any real-world property of the agents they represent – which means that the trend information collected will be unusable. The differential mutation was allowed to operate once the algorithm had proceeded beyond 100,000 iterations.

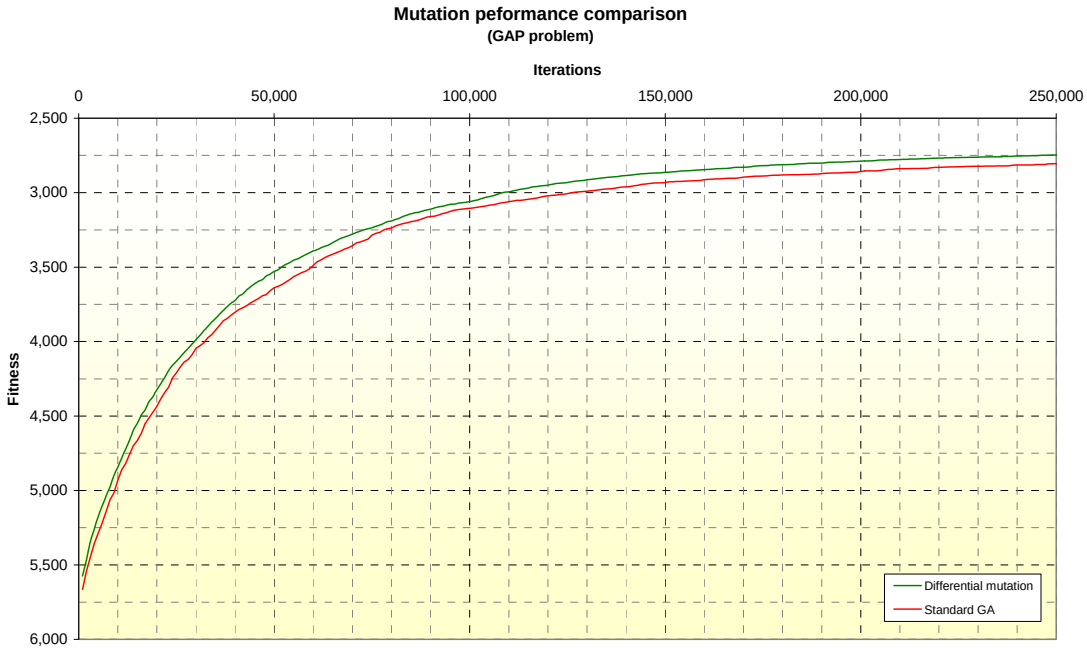

Figure 6-2: Mutation performance comparison - Generalized Assignment Problem

The results shown in Figure 6-2 are the average performances of at least 40 runs for each mutation type. A steady state GA was employed, as the analysis requires statistics collected by the Replacer component of the GA in order to determine the relative fitness of new solutions. This component is not used in generational algorithms because all new solutions are accepted unconditionally into the population for the next generation.

The effectiveness of the differential mutation is somewhat questionable on this problem: beyond the 100,000 iteration point where the revised algorithm starts operating, a marginal improvement in performance can be discerned – although this is not immediately apparent in the graph above. Overall, the differential mutation appears to offer minimal advantages over the standard GA for this problem.

6.1.5. Distributed Performance

Computer	Processor	Memory	Network
X	2.4 GHz Intel Core 2 Quad	4 GB	Fast
Y	2.4 GHz AMD Athlon64x2	2 GB	Fast
Z	1.8 GHz Intel Core 2 Duo	2 GB	Fast

Table 6-1: Hardware specifications of distributed test environment computers

Each of the three networks under consideration were employed to assess the effectiveness of the Distributed Evaluation for EPANET. Table 6-1 relates the computers employed for this task. Computer X is equipped with a quad-core processor, whilst Computers X and Y both have dual core processors. Computer X, as the most powerful, was nominated as the server for the group. Computer X, as the most powerful, was nominated as the server for the group. None of the computers is equipped with a HyperThreading processor – which has been shown to improve performance under initial testing – and accordingly the two dual core machines were configured to run three instances of the deEPANET server each and four instances for Computer X – which would also be responsible for running the client on which the optimization would actually proceed.

To minimize the effects of external network traffic impacting on the performance metrics, the machines were connected on a private, gigabit Ethernet network via a switch. Before each distributed test was run, the same optimization was performed on each individual machine in order to obtain a baseline figure for its performance.

Because of the hardware differences between these evaluations and those presented for the initial testing in Chapter 5, the performance figures are not directly comparable and should not be taken as a measure of the relative complexity of the problems.

6.2. New York Tunnels

6.2.1. Problem Formulation

The problem that has come to be known as "New York Tunnels" was introduced by Schaake and Lai (1969) as an illustration of a large-scale optimization problem for the reinforcement of the water supply for New York City. The "Tunnels" name stems from the fact that the pipes are of inordinately large diameter, ranging from 60 inches (1.5 metres) to 204 inches (5.2 metres). In this problem, each of the 21 pipes in the network may be duplicated with one of 15 commercially available pipe sizes or left unduplicated. This gives a solution space of $16^{21} = 1.93\times10^{25}$ solutions.

This network has become a favourite benchmark for optimization applications and has been employed widely in the literature. From Schaake and Lai's original solution of $78.1m the best known solution for the problem has been advanced by, among others, Morgan & Goulter (1985 – $39.229m) and Murphy *et al.* (1993 - $38.814m). Savić and Walters (1997) used the problem to illustrate the sensitivity of such optimization problems to small changes in the coefficients used in calculating the frictional losses observed in the system, demonstrating solutions ranging from $37.140m to $40.452m representing the best solutions found for the gamut of coefficients used for this problem by other workers in the field. At the time of writing, the best-known feasible solution to the problem of $38.644m was first published by Meier *et al.* (2003) using an Ant Colony Simulation (ACS) approach.

6.2.2. Network Configuration

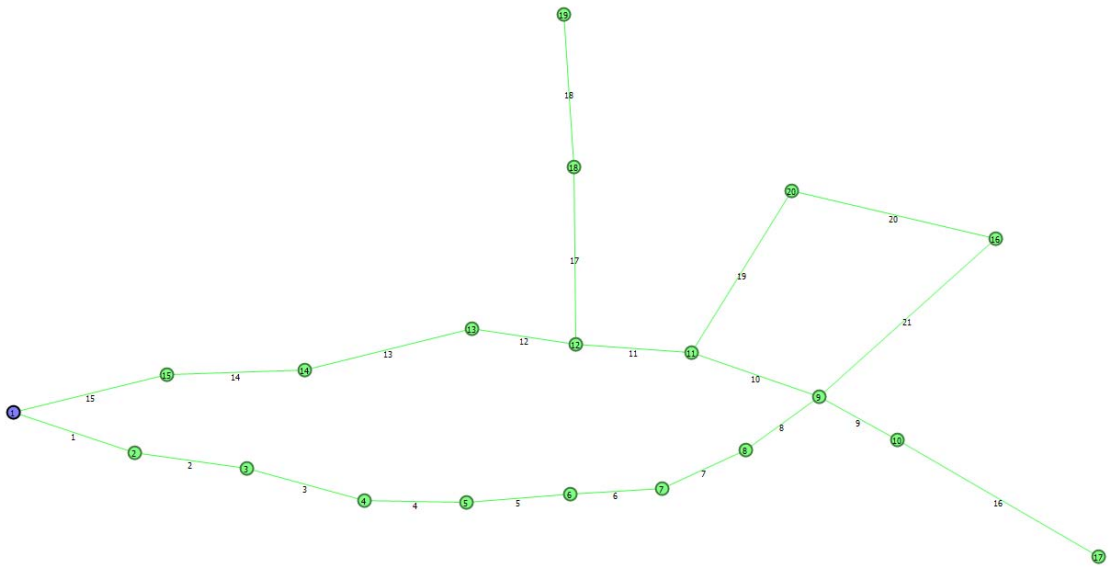

Figure 6-3: New York Tunnels Topology

The network topology consists of two loops and two branches supplied under gravity by a single, fixed-head reservoir. In the original, pressure deficient configuration, nodes 16, 18, 19, 20 and 21 at the periphery of the network fall below the required minimum pressures (see Table 6-2). This problem was originally formulated in Imperial units – metric equivalents are given.

Node ID	Elevation (feet)	Demand (cubic feet per second)	Demand (litres per second)	Minimum Pressure (pounds per square inch)	Minimum Pressure (metres H_2O)
2	0	92.4	2,616.47	255.0	179.28
3	0	92.4	2,616.47	255.0	179.28
4	0	88.2	2,497.55	255.0	179.28
5	0	88.2	2,497.55	255.0	179.28
6	0	88.2	2,497.55	255.0	179.28
7	0	88.2	2,497.55	255.0	179.28
8	0	88.2	2,497.55	255.0	179.28
9	0	170.0	4,813.86	255.0	179.28
10	0	1.0	28.31	255.0	179.28
11	0	170.0	4,813.86	255.0	179.28
12	0	117.1	3,315.90	255.0	179.28
13	0	117.1	3,315.90	255.0	179.28
14	0	92.4	2,616.47	255.0	179.28
15	0	92.4	2,616.47	255.0	179.28
16	0	170.0	4,813.86	260.0	182.80
17	0	57.5	1,682.22	272.8	191.24
18	0	117.1	3,315.90	255.0	179.28
19	0	117.1	3,315.90	255.0	179.28
20	0	170.0	4,813.86	255.0	179.28

Table 6-2: New York Tunnels Node Characteristics

Reservoir ID	Elevation (feet)	Total Head (feet H_2O)	Total Head (metres H_2O)
1	0	300	91.44

Table 6-3: New York Tunnels Reservoir Characteristics

Pipe	From Node	To Node	Diameter (inches)	Diameter (mm)	Length (feet)	Length (m)	H-W Friction Factor
1	1	2	180	4,572	11,600	3,535.68	100
2	2	3	180	4,572	19,800	6,035.04	100
3	3	4	180	4,572	7,300	2,225.04	100
4	4	5	180	4,572	8,300	2,529.84	100
5	5	6	180	4,572	8,600	2,621.28	100
6	6	7	180	4,572	1,9100	5,821.68	100
7	7	8	132	3,352.8	9,600	2,926.08	100
8	8	9	132	3,352.8	12,500	3,810.00	100
9	9	10	180	4,572	9,600	2,926.08	100
10	11	9	204	5,156.2	11,200	3,413.76	100
11	12	11	204	5,156.2	14,500	4,419.6	100
12	13	12	204	5,156.2	12,200	3,718.56	100

Pipe	From Node	To Node	Diameter		Length		H-W Friction Factor
			(inches)	(mm)	(feet)	(m)	
13	14	13	204	5,156.2	24,100	7,345.68	100
14	15	14	204	5,156.2	21,100	6,431.28	100
15	1	15	204	5,156.2	15,500	4,724.4	100
16	10	17	72	1,828.8	26,400	8,046.72	100
17	12	18	72	1,828.8	31,200	9,509.76	100
18	18	19	60	1,524.0	24,000	7,315.2	100
19	11	20	60	1,524.0	14,400	4,389.12	100
20	20	16	60	1,524.0	38,400	11,704.32	100
21	9	16	72	1,828.8	26,400	8,046.72	100

Table 6-4: New York Tunnels Pipe Characteristics

Pipe option	Diameter		Cost		Pipe option	Diameter		Cost	
	(inches)	(mm)	($/foot)	($/metre)		(inches)	(mm)	($/foot)	($/metre)
0	No Duplication		0.00		**8**	120	3,048.0	416.46	1,366.34
1	36	914.4	93.59	307.05	**9**	132	3,352.8	468.71	1,537.76
2	48	1,219.2	133.70	438.65	**10**	144	3,657.6	522.11	1,712.96
3	60	1524.0	176.32	578.48	**11**	156	3,962.4	576.59	1,891.70
4	72	1828.8	221.05	725.23	**12**	168	4,267.2	632.09	2,073.79
5	84	2,133.6	267.61	877.99	**13**	180	4,572.0	688.54	2,258.99
6	96	2,438.4	315.80	1,036.09	**14**	192	4,876.8	745.91	2,447.21
7	108	2,743.2	365.46	1,199.02	**15**	204	5,181.6	804.14	2,638.25

Table 6-5: New York Tunnels Pipe Duplication Options

The pipe costs per unit length seen in Table 6-5 are shown to two decimal places. Within the optimization software itself, however, the cost versus diameter function defined by Schaake & Lai (1969) is used thus:

$$Cost = 1.1 \cdot Diameter^{1.24} \cdot Length \qquad \text{vii)}$$

Where *Diameter* is in inches and *Length* in feet and *Cost* is in US Dollars.

6.2.3. GA Configuration

An Elitist Generational GA was employed for the Gene Expression and Heterozygous comparisons, preserving the two best solutions in each population. Single-point crossover was used with a probability of 95% occurrence. The "standard" mutation operator was used which gave a 70% probability of a single gene being mutated. A penalty term was introduced into the optimization to penalize infeasible solutions, which produce insufficient pressure at the demand nodes equating to $100,000,000 per psi of head deficit.

6.2.4. Genetic Representation

6.2.4.1. Binary String

For the standard binary string representation 23% of the runs converged to the best-known optimal solution and the variation in the range of solutions obtained visible in Figure 6-4.

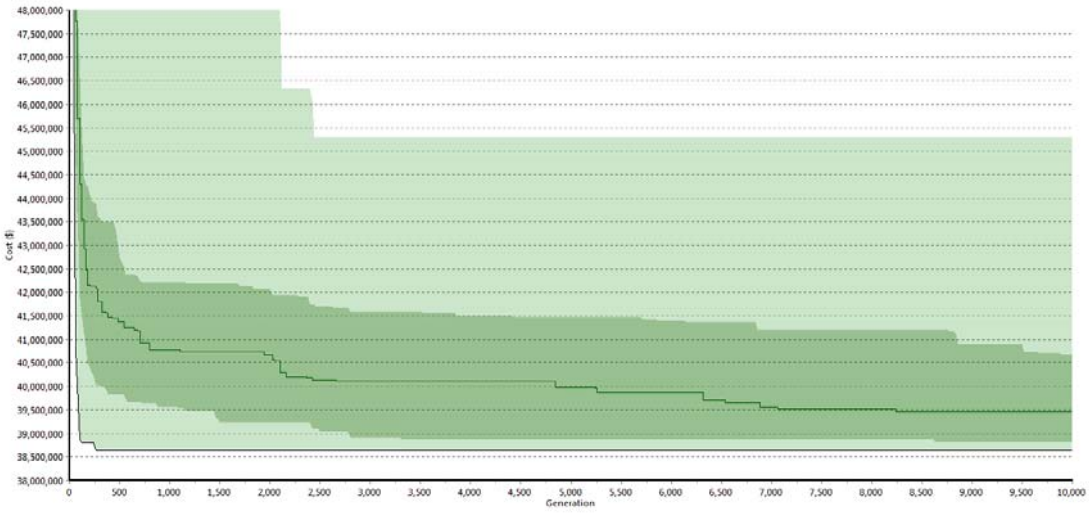

Figure 6-4: Algorithmic Performance: New York Tunnels - Binary String

6.2.4.2. Gray Binary String

As can be seen from Figure 6-5 the Gray-coded Binary String clearly outperforms its conventionally coded relative, with 43% of the solutions converged to the optimal solution as well as a clearly superior algorithmic performance overall.

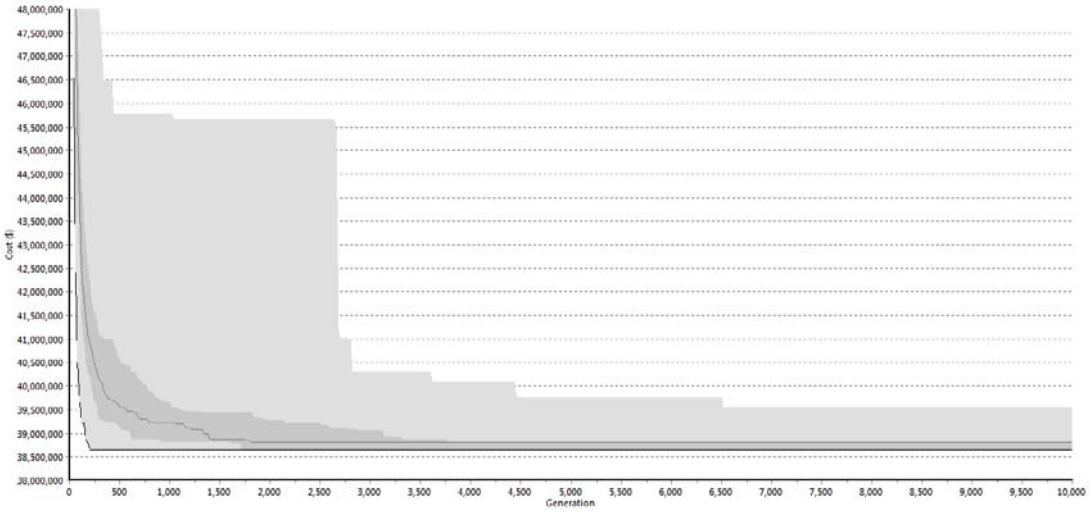

Figure 6-5: Algorithmic Performance: New York Tunnels - Gray Binary String

6.2.4.3. Integer

The use of the Integer encoding for the genes leads to 41% of the runs identifying the best-known solution of $38.644m. Again, the integer representation is clearly shown (Figure 6-6) to be superior to that of the conventional binary string.

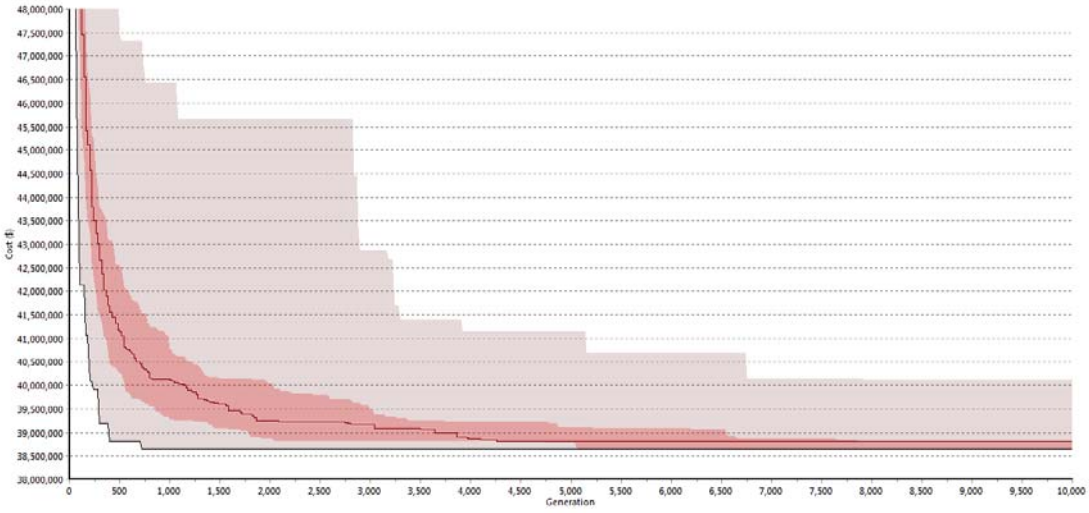

Figure 6-6: Algorithmic Performance: New York Tunnels – Integer

6.2.4.4. Hybrid Integer

The results obtained for the hybrid integer gene, shown in Figure 6-7, demonstrate a tightly confined set of results. The median value reaches the known global optimum after around 5,750 generations (563,600 evaluations) – indeed, 53% of the runs converged to the optimal solution for this representation. This is the only representation for which the median performance for the runs was seen to reach the optimal solution. The Gray-coded version of the standard binary string representation can expected to perform identically, in terms of algorithmic performance, as this hybrid integer gene given that they both use the same underlying genetic representation and are acted upon by the recombination routines in the same fashion and the results shown in Figure 6-5 and Figure 6-7 confirm this assertion.

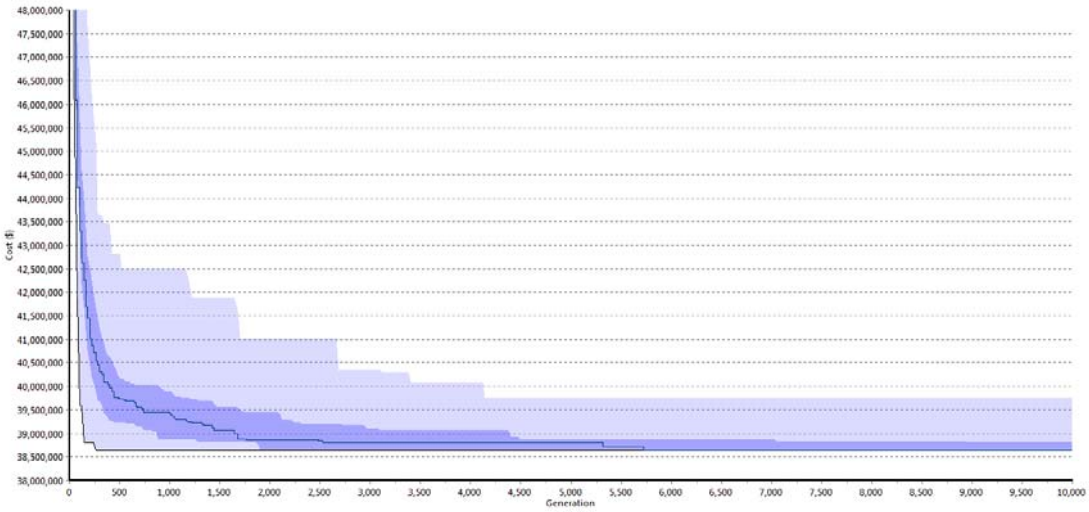

Figure 6-7: Algorithmic Performance: New York Tunnels - Hybrid Binary String

The best run located this minimum after 260 generations (or 25,482 evaluations). This compares favourably with some of the best performances reported in the literature:

Paper	Technique	Solution Cost	Least Evaluations Required
Dandy *et al.* (1996)	GA	\$38.81m	96,750
Wu *et al.* (2001)		\$37.13†	37,186
Lippai *et al.* (1999)		\$38.13m†	46,016
Eusuff & Lansey (2003a)	Shuffled Frog Leaping Algorithm	\$35.27m‡	28,200
Eusuff & Lansey (2003b)		\$38.13m†	31,267
		\$38.81m	21,569
Maier *et al.* (2003)	Ant Colony	\$38.64m	13,928

Table 6-6: New York Tunnels: Comparison with Literature Results

† Solution feasible when using "relaxed" Hazen-Williams headloss coefficient, c.f. Savić & Walters (1997).
‡ Infeasible solution.

6.2.4.5. Comparative Analysis

Figure 6-8 illustrates the best values obtained from each of the genotype representations. It is clear from the figure that the best Hybrid run identifies the minimum after around 260 generations, similar to the conventional binary string. The Gray-coded binary string, however, outperforms both identifying the minimum around 200 generations. In terms of numbers of evaluations, these figures relate to evaluations of 25,482 and 19,602 respectively.

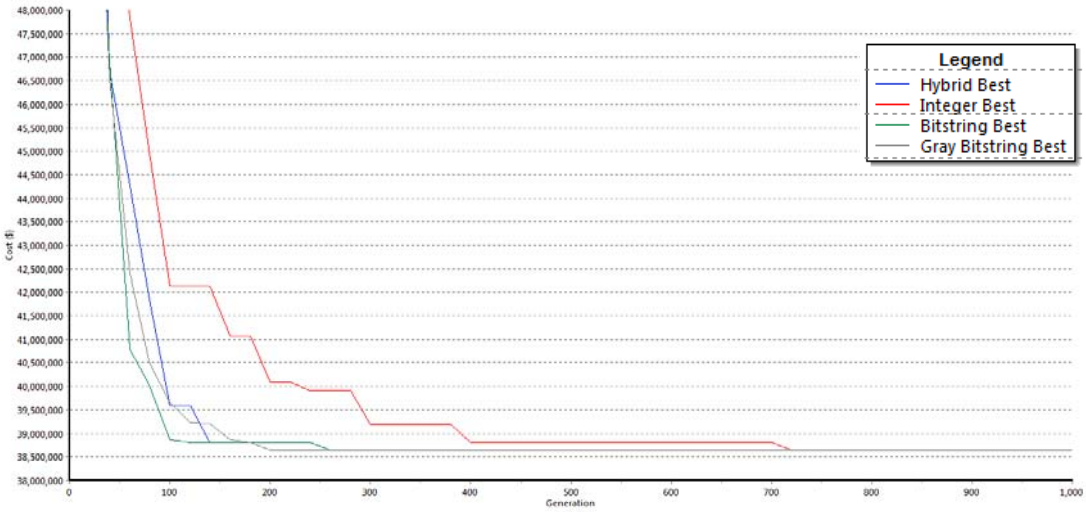

Figure 6-8: Algorithmic Performance: New York Tunnels - Combined Best

The above results are combined in Figure 6-9 which, for clarity, has the maximum and minimum range for each result removed – leaving the upper and lower quartiles and the median value for each genotype representation. The weak performance of the classically coded binary string is highlighted in this composite. The improvement in performance achieved by the Hybrid Integer gene over its conventional integer counterpart is also clear. This can be attributed to the crossover and mutation characteristics the hybrid gene takes from the Gray-coded binary string. The broad equivalence of the Gray-coded binary string and the Hybrid Integer gene is demonstrated by their overlap in the above figure – consequently, the Gray-coded binary string representation will be omitted from further analysis.

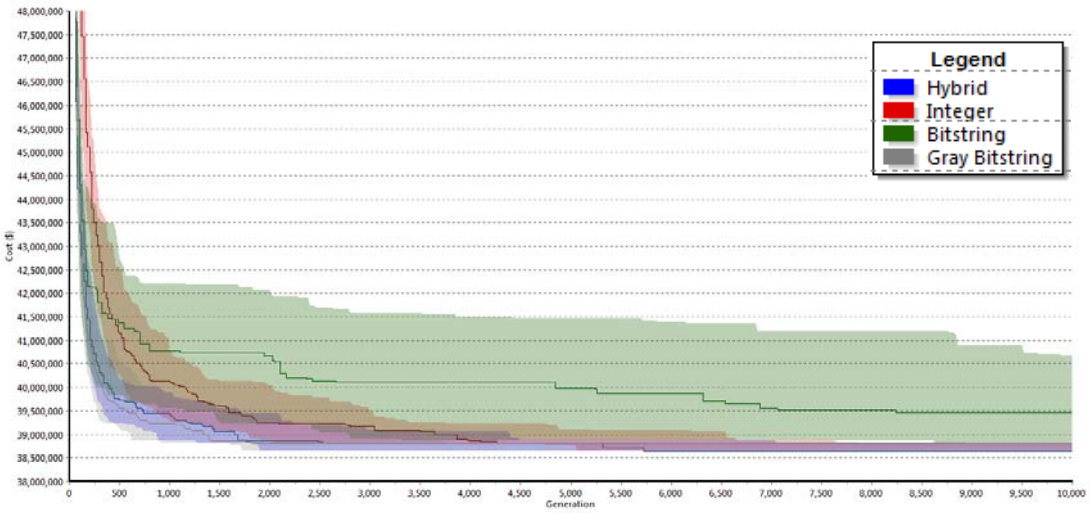

Figure 6-9: Algorithmic Performance: New York Tunnels - Combined Upper/Lower Quartiles

6.2.4.6. Runtime Performance

Each of the optimization runs was timed and averaged to give a realistic performance rate for the optimization. In order to minimize the influence of external factors on the timing, runs were undertaken on a computer equipped with a quad-core processor with no other processes running other than standard operating system services.

Chromosome Representation	Average Performance (evaluations per second)	% of best performance
Binary String	8,893.63	86.9%
Integer	10,234.89	100%
Gray Binary String	8,111.29	79.3%
Hybrid Integer	8,880.60	86.8%

Table 6-7: New York Tunnels Runtime Performance

As can be seen from Table 6-7, the performance of the integer gene outstrips that of the other representations, although its algorithmic performance was weaker than the two Gray-coded alternatives. The hybrid integer continues to outperform the Gray coded binary string and, for this length of chromosome, performs almost identically to the conventional binary string (c.f. Table 4-4). Thus it can be seen that, for this problem, a trade-off exists between run-time performance and the algorithmic performance.

6.2.5. Heterozygous Chromosomes

The heterozygous form of the New York Tunnels has been encoded to allow 10 pipes to be modified at once. The chromosome comprises 10 pairs of genes, the first being allowed to vary between 1and 21 represents the pipe to be modified. The second gene of the pair encodes the pipe diameter as previously. Thus for the New York Tunnels problem, the total chromosome length is reduced by one gene.

6.2.5.1. Binary String

The results from the runs of the heterozygous form of the binary string can be seen in Figure 6-10.

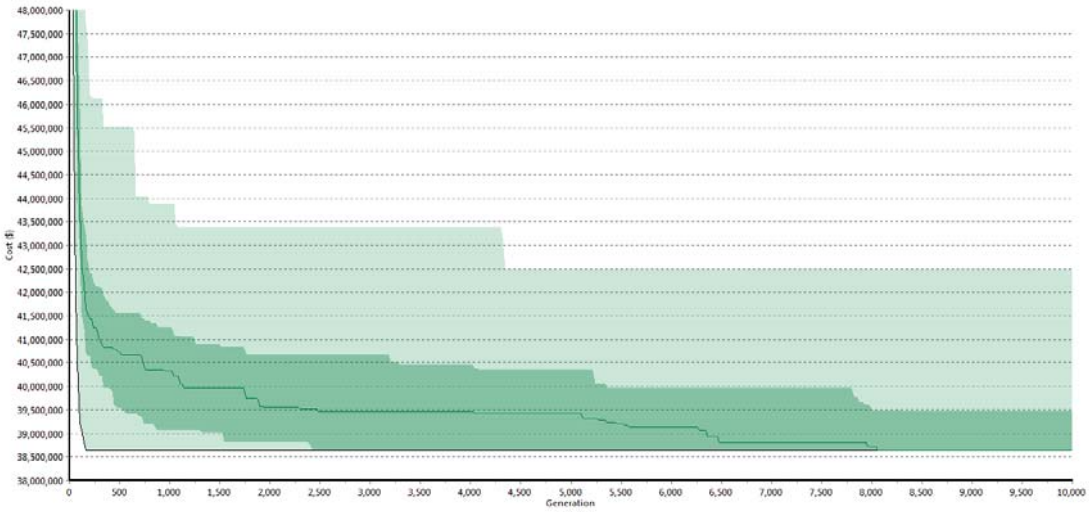

Figure 6-10: Algorithmic Performance: New York Tunnels - Heterozygous Binary String

Overlain with the original results from Chapter 6.2.4 in Figure 6-11 the heterozygous results can be seen to be superior to the original configuration– improving more rapidly and converging to the optimal solution on more occasions (47% vs 23%).

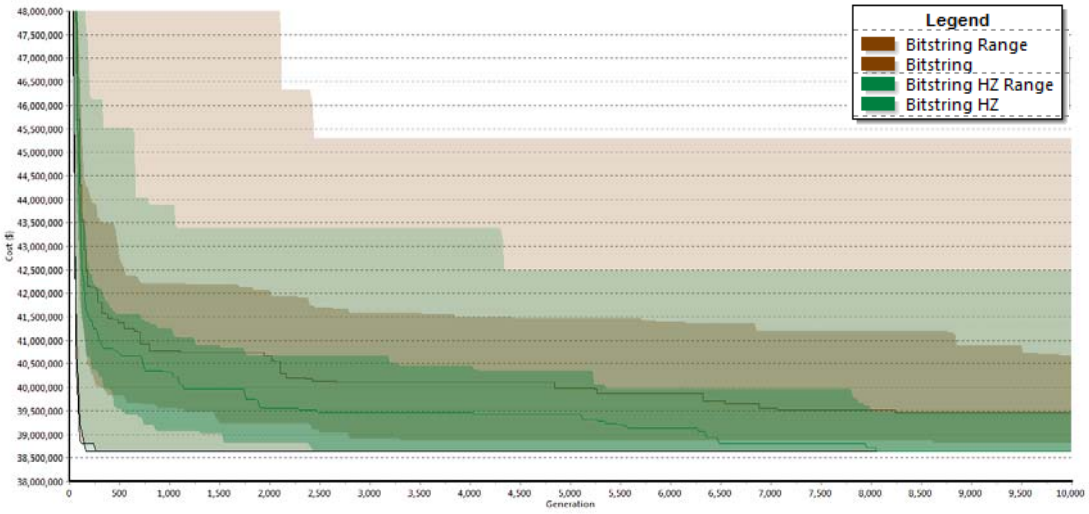

Figure 6-11: Algorithmic Performance: New York Tunnels - Heterozygous Binary String results overlain with conventional results

6.2.5.2. Integer

Figure 6-12 illustrates the results obtained for the heterozygous runs with the integer representation for the genes.

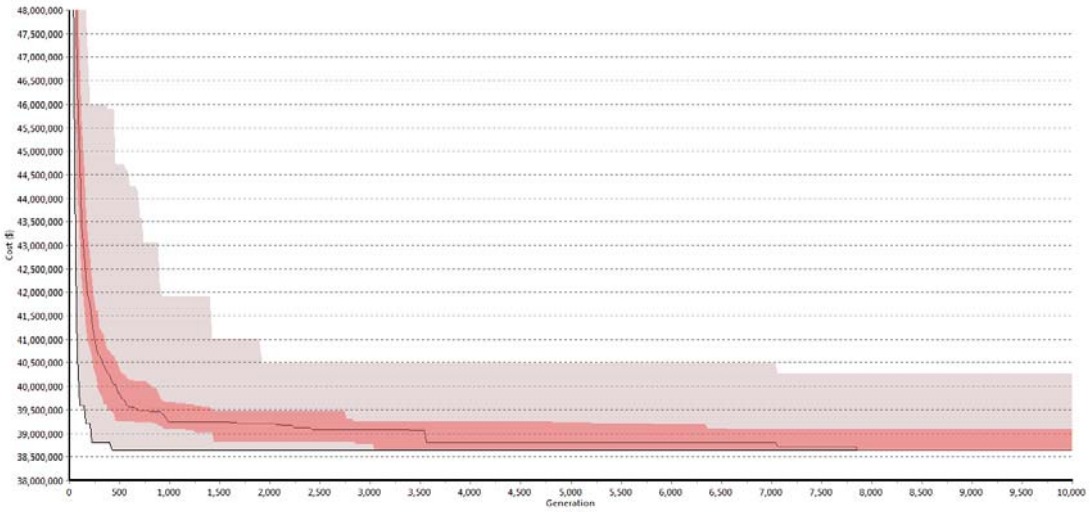

Figure 6-12: Algorithmic Performance: New York Tunnels - Heterozygous Integer

Once more, the heterozygous version of the algorithm can be seen, in Figure 6-13, to be outperforming the conventionally encoded version when the results are overlain. Here 56% of the runs identified the best-known solution compared to just 41% of the conventionally encoded runs.

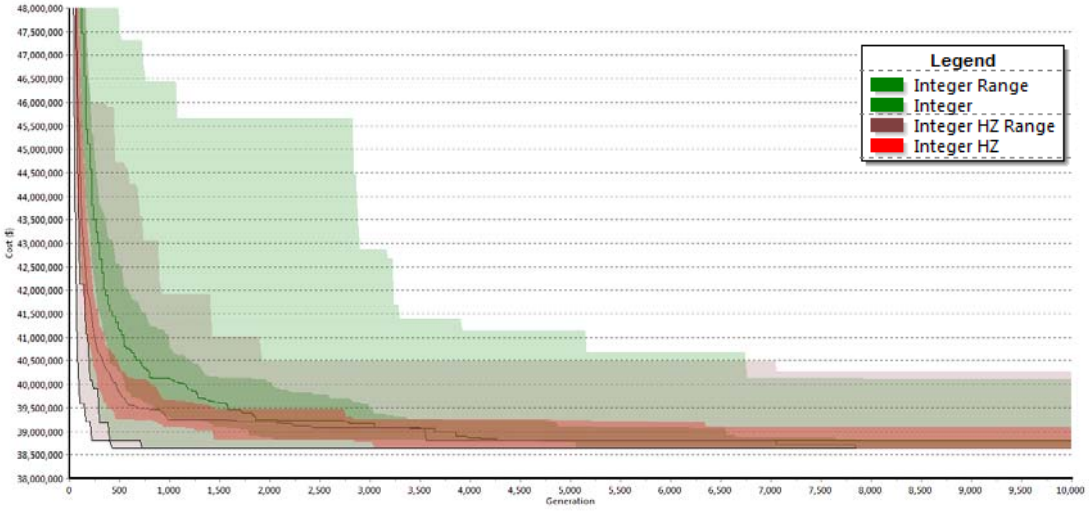

Figure 6-13: Algorithmic Performance: New York Tunnels - Heterozygous Integer results overlain with conventional results

6.2.5.3. Hybrid Integer

The results of the hybrid integer representation employing the heterozygous implementation of the New York Tunnels problem are illustrated in Figure 6-14.

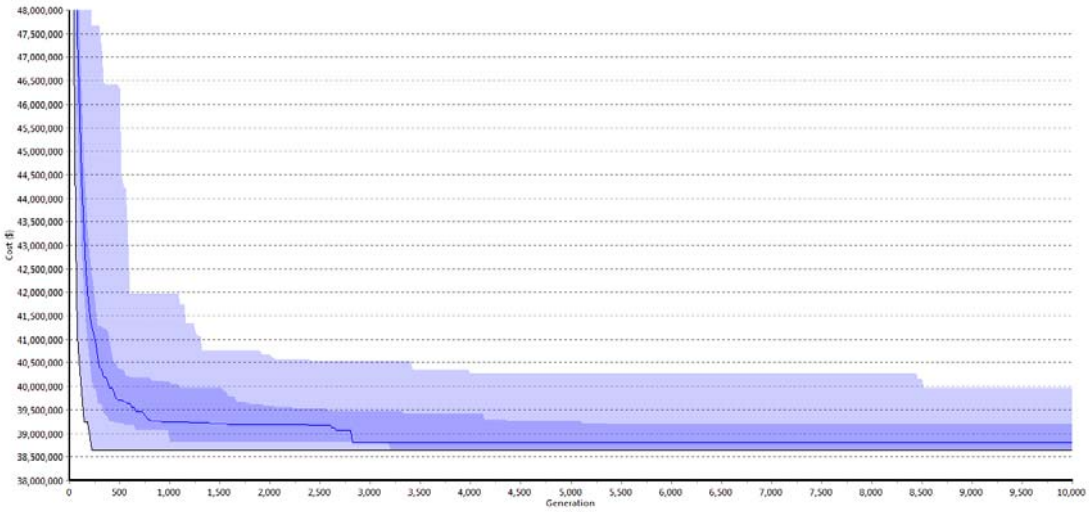

Figure 6-14: Algorithmic Performance: New York Tunnels - Heterozygous Hybrid Binary String

As can be seen from Figure 6-15, the differential in performance between the two versions of the Hybrid Integer is much reduced in comparison to the other representations. Indeed in terms of success in finding the optimal solution, the performance of the Hybrid Integer dipped below that of the pure Integer representation – managing 47% of runs versus 53% of the equivalent, conventionally encoded runs.

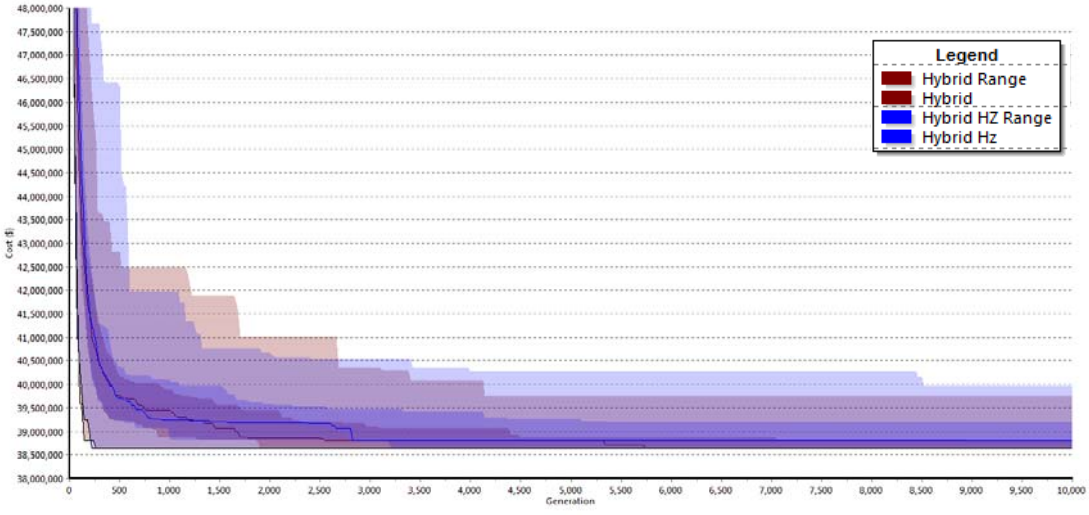

Figure 6-15: Algorithmic Performance: New York Tunnels - Heterozygous Hybrid Integer results overlain with conventional results

6.2.5.4. Comparative Analysis

In general, the best performing runs of the heterozygous configurations, for all representations, converged quicker to the best known solution than their conventionally configured equivalents – as can be seen in Figure 6-16 (c.f. Figure 6-8). Here it can be seen

that the best binary string run identified the optimum solution after just 160 generations (15,682) evaluations with that of the hybrid integer and integer representations after 220 generations (21,660 evaluations) and 420 generations (41,260 evaluations) respectively.

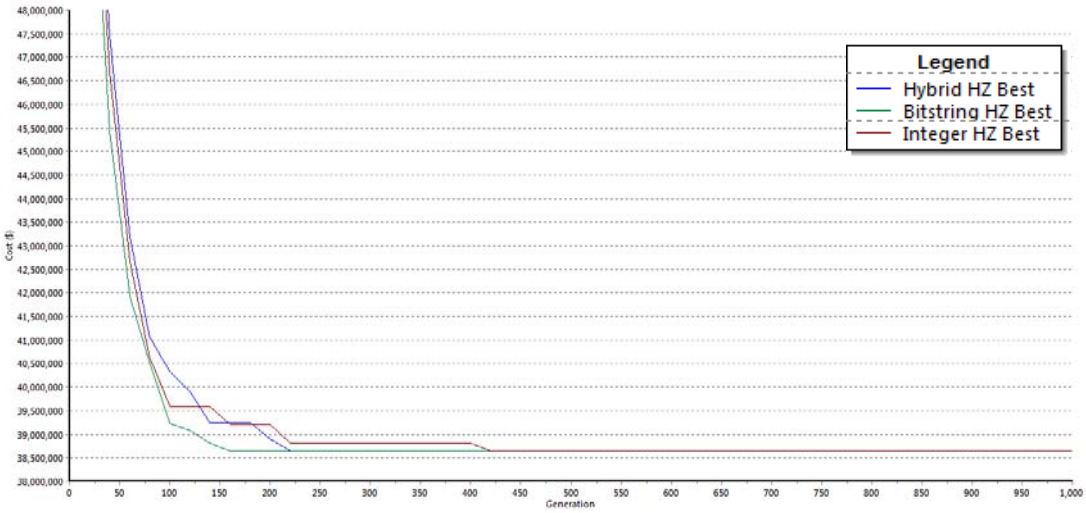

Figure 6-16: Algorithmic Performance: New York Tunnels – Combined Heterozygous Best

Figure 6-17 combines the algorithmic performance for the three genetic representations for the heterozygous problem implementation. For clarity, the maximum/minimum curves are omitted.

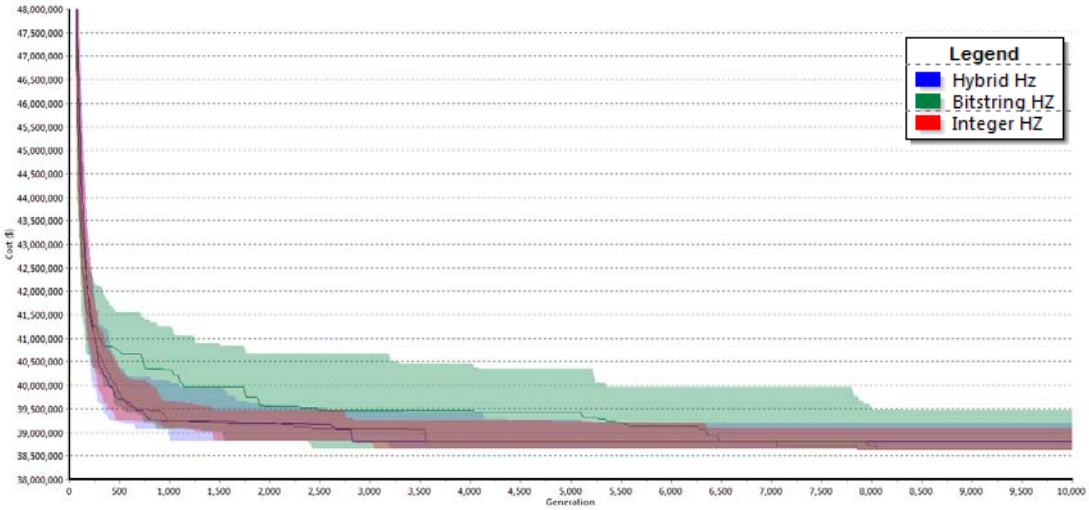

Figure 6-17: Algorithmic Performance: New York Tunnels - Combined Heterozygous Upper/Lower Quartiles

6.2.5.5. Runtime Performance

The principal advantage of adopting a heterozygous encoding for the GA is that it can reduce runtimes of the algorithm by shortening the chromosome length. In this instance, however,

the chromosome length is only one element shorter than the standard representation since each chromosome encodes 10 pipes IDs to modifiy and 10 diameters vs. 21 diameters for the standard representation. Thus with the additional overhead of decoding the genes for identifying the pipes to apply the diameters to, the heterozygous representations are all marginally slower than their conventional counterparts, as shown in Table 6-8. This result would not be expected to be seen with more complex chromosomes, for instance those of the Piedemonte San Germano problem below.

Chromosome Representation	Conventional		Heterozygous	
	(evaluations per second)	*% of best*	*(evaluations per second)*	*% of best*
Binary String	8,893.63	86.9%	7,474.96	73.0%
Integer	10,234.89	100%	8,433.48	82.4%
Hybrid Integer	8,880.60	86.8%	7,590.13	74.2%

Table 6-8: New York Tunnels Heterozygous vs. Conventional Runtime Performance

6.2.6. Caching

Table 6-9 shows that the caching performance for the New York Tunnels problem is relatively poor compared to that experienced with the GAP problems in the initial trials in which up to ~8% of solutions were cached. Here only 1.1% (binary tree) and 2.9% (Judy tree) of solutions were cached despite the solution space for the New York Tunnel problem being considerably smaller.

Cache Size	Runtime performance	Cache Performance		Relative Runtime Performance
	(solutions/second)	*(no. hits)*	*(% of evaluations)*	
None	8,880.60	n/a	n/a	100%
40,000	8,207.58	21,713	1.1%	108.2%
Judy (unlimited)	8,506.32	57,391	2.9%	104.4%

Table 6-9: Cache results: New York Tunnels

The degradation of performance when using the cache with this problem is clear with both cached runtimes exceeding those of the uncached algorithm. This occurs because the objective function is too simple to justify the application of the cache in this instance. However, the performance of the Judy cache, in both runtime performance and the number of cache hits is encouraging.

6.2.7. Adaptive Differential Mutation

The New York Tunnels problem has a significantly smaller solution space than the GAP problem and, accordingly, the differential mutation was introduced much earlier in the

optimization after just 2,000 iterations (4,000 evaluations) had been performed. Figure 6-44 shows that until the 2,000th iteration, the algorithms with and without the differential mutator perform broadly similarly – as would be expected as they are indeed the same algorithm. Beyond that point, though, the effect of switching on the differential mutation – in which trend is considered - is significant. At this point, the differential mutator begins to concentrate on the four or five genes that are critical to this optimization. At first it was suspected that this was a manifestation of an effective increase in the mutation rate (although the probability of mutation occurring does not, itself, change. However, beyond the 2,000 iteration mark the effective mutation rate for an individual gene actually decreases – illustrating how effective it is to steer the mutation to certain genes preferentially.

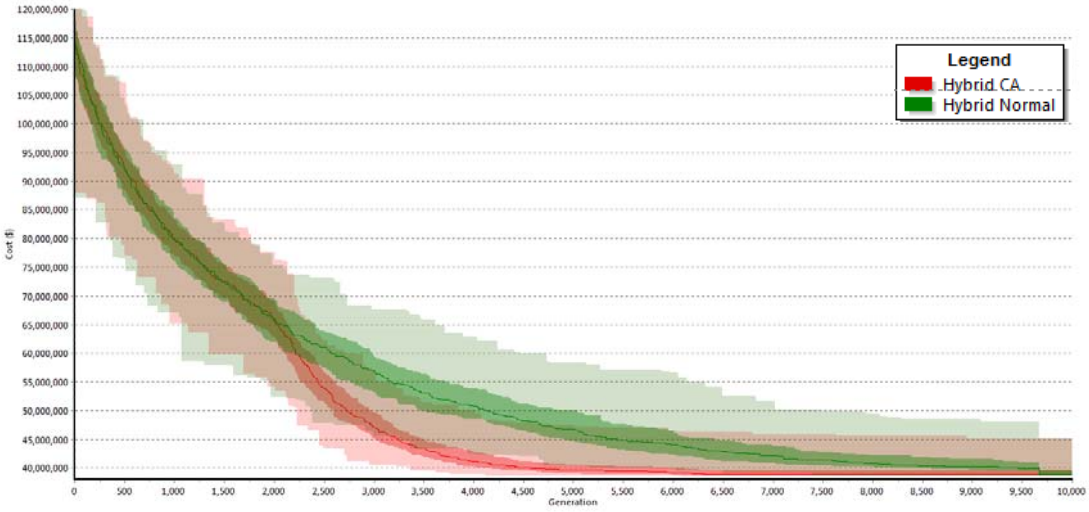

Figure 6-18: Mutation performance comparison - New York Tunnels problem

6.2.8. Distributed Performance

The single objective New York Tunnels problem (normal representation using the hybrid integer gene) was evaluated with the deEPANET distributed hydraulic solver. The baseline performance figures, for the machines employed, suggest that the maximum performance for this problem is as shown in Table 6-10. Two important caveats should be stated about these figures. Firstly, the baseline performance figures do not include any overhead imposed by the operating system which would have a deleterious effect on the throughput when all of the processing capacity of the computer is committed to deEPANET. Secondly, the figures for the computers Y and Z which will be improved marginally because, when employed as servers in the deEPANET network, they no longer have the overhead of running the

optimization part of the algorithm. Instead, they are tasked solely with undertaking the hydraulic simulations.

Computer	Baseline Performance *(evaluations/second)*	Number of Processor Cores	Theoretical Maximum Throughput *(evaluations/second)*
X	8,881	4	35,524
Y	6,227	2	12,454
Z	5,952	2	11,904
Total	**21,060**	**8**	**59,882**

Table 6-10: Theoretical maximum performance for distributed New York Tunnels problem

The results for the distributed optimization are presented in Table 6-11. As can be seen from the baseline performance figures, the computational load of the New York Tunnels hydraulic simulation is trivial for a modern computer with a *single* processor of Computer X managing to perform almost 9,000 evaluations per second as well as managing the optimization algorithm itself.

Computer	Baseline Performance *(evaluations/second)*	Distributed Performance *(evaluations/second)*	
X	8,881	5,826	23,490
		5,929	
		5,898	
		5,837	
Y	6,227	3,112	11,082
		3,628	
		4,342	
Z	5,952	3,046	8,182
		2,492	
		2,644	
Totals	**21,060**		**42,754**

Table 6-11: New York Tunnels distributed performance results

The distributed performance is, however, somewhat disappointing with the total solution throughput approximately doubling – reaching 71% of the theoretical maximum - despite an additional five processor cores being employed in the optimization. The results for the two dual-core computers are somewhat better – given that, individually, they are no longer bearing the load of managing the optimization. The performance of Computer X is negatively impacted by having to serve large volumes of solutions across the network for Computers Y and Z to handle.

6.3. Hanoi

6.3.1. Problem Formulation

The Hanoi problem is introduced by Fujiwara & Kang (1990). It is a network design problem in which 34 pipes may take one of six pipe diameters giving a search space size of $6^{34} = 2.87 \times 10^{26}$ – an order of magnitude larger than the New York Tunnels problem above. A global minimum pressure constraint of 30 metres applies to the optimization.

This is also a familiar benchmark network from the literature and many results have been obtained with different optimization techniques. As with the New York Tunnels problem, Savić & Walters (1997) observed that differences in the coefficients applied to the Hazen-Williams headloss formula make it difficult to make direct comparisons between these results and adopted upper and lower bounds for their analyses of $\omega = 10.9031$ and $\omega = 10.5088$ – being the limits identified as being used in other research. For the $\omega = 10.5088$ constraint, the best know result is that of Cunha and Sousa (1999) who identified it with simulated annealing, costing \$6.056m. This same result has subsequently also been confirmed by Geem *et al.* (2002) using Harmony Search and by Kadu *et al.* (2008). The more restrictive $\omega = 10.9031$ has seen optimal solutions of \$6.195m (Savić & Walters, 1997) and more recently \$6.190m (Kadu *et al.,* 2008). Eusuff & Lansey (2003) identify the former result as being, at the time, the best known result obtained using EPANET's default ω value of 10.6744 which is what will be used in this analysis. They further propose a solution of \$6.073m which is marginally infeasible (by 0.41m at node 13) which was obtained in 26,987 evaluations. Zecchin *et al.* (2007) review the application of a number of classes of algorithm to this subject and conclude that the best-published result for the Hanoi problem using EPANET is that of Zecchin *et al.* (2006) of \$6.134m which was obtained using the Max-Min Ant System of Stützle & Hoos (2000).

6.3.2. Network Configuration

The network comprises three loops and two short branches fed by a single fixed-head reservoir as shown in Figure 6-19. Table 6-12 to Table 6-15 describe the characteristics of the network components.

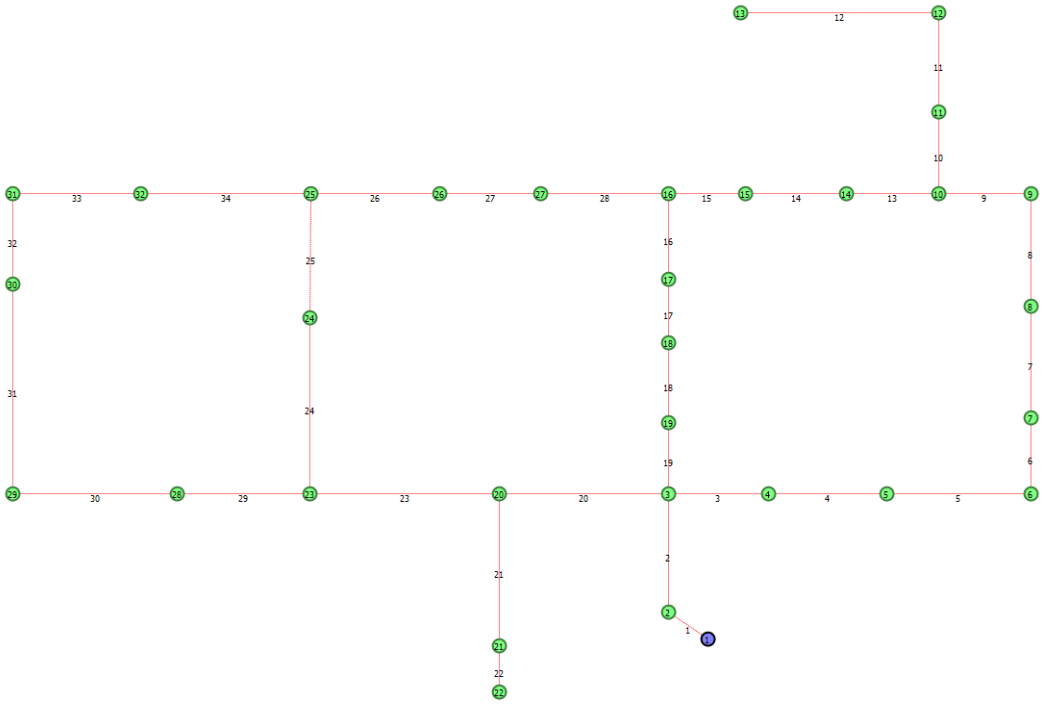

Figure 6-19: Hanoi Network Topology

Node ID	Elevation *(metres)*	Demand *(cubic metres per hour)*	Minimum Pressure *(metres H_2O)*
2	0	890	30.0
3	0	850	30.0
4	0	130	30.0
5	0	725	30.0
6	0	1,005	30.0
7	0	1,350	30.0
8	0	550	30.0
9	0	525	30.0
10	0	525	30.0
11	0	500	30.0
12	0	560	30.0
13	0	940	30.0
14	0	615	30.0
15	0	280	30.0
16	0	310	30.0
17	0	865	30.0
18	0	1,345	30.0
19	0	60	30.0
20	0	1,275	30.0
21	0	930	30.0
22	0	485	30.0

Node ID	Elevation *(metres)*	Demand *(cubic metres per hour)*	Minimum Pressure *(metres H_2O)*
23	0	1,045	30.0
24	0	820	30.0
25	0	170	30.0
26	0	900	30.0
27	0	370	30.0
28	0	290	30.0
29	0	36	30.0
30	0	360	30.0
31	0	105	30.0
32	0	805	30.0

Table 6-12: Hanoi Node Characteristics

Reservoir ID	Elevation *(metres)*	Total Head *(metres)*
1	0	100

Table 6-13: Hanoi Reservoir Characteristics

Pipe	From Node	To Node	Length *(metres)*	H-W Friction Factor
1	1	2	100	130
2	2	3	1,350	130
3	3	4	900	130
4	4	5	1,150	130
5	5	6	1,450	130
6	6	7	450	130
7	7	8	850	130
8	8	9	850	130
9	9	10	800	130
10	10	11	950	130
11	11	12	1,200	130
12	12	13	3,500	130
13	10	14	800	130
14	14	15	500	130
15	15	16	550	130
16	17	16	2,730	130
17	18	17	1,750	130
18	19	18	800	130
19	3	19	400	130
20	3	20	2,200	130
21	20	21	1,500	130
22	21	22	500	130

Pipe	From Node	To Node	Length *(metres)*	H-W Friction Factor
23	20	23	2,650	130
24	23	24	1,230	130
25	24	25	1,300	130
26	26	25	850	130
27	27	26	300	130
28	16	27	750	130
29	23	28	1,500	130
30	28	29	2,000	130
31	29	30	1,600	130
32	30	31	150	130
33	32	31	860	130
34	25	32	950	130

Table 6-14: Hanoi Pipe Characteristics

The Hanoi problem is defined in mixed, metric/imperial units, the available pipe diameters being specified in inches.

Pipe option	Diameter *(inches)*	Diameter *(mm)*	Cost *($ per metre)*
0	12	304.8	45.73
1	16	406.4	70.40
2	20	508.0	98.39
3	24	609.6	129.33
4	30	762.0	180.75
5	40	1,016.0	278.28

Table 6-15: Hanoi Pipe Options

The cost function is non-linear and is expressed in Equation viii) with the results summarized in Table 6-15.

$$Cost = 1.1 \cdot Diameter^{1.5} \cdot Length \qquad \text{viii)}$$

Where *Diameter* is in inches and *Length* in metres and *Cost* is in US Dollars

6.3.3. GA Configuration

The GA configuration employed was identical to that of the New York Tunnels problem above except the penalty cost was modified to be $250,000 per metre of head deficit

6.3.4. Genetic Representation

6.3.4.1. Binary String

The results from the runs of the Hanoi problem with the binary string representation are shown in Figure 6-20. As with the New York Tunnels problem before (Figure 6-4), the results show a wide variation between runs.

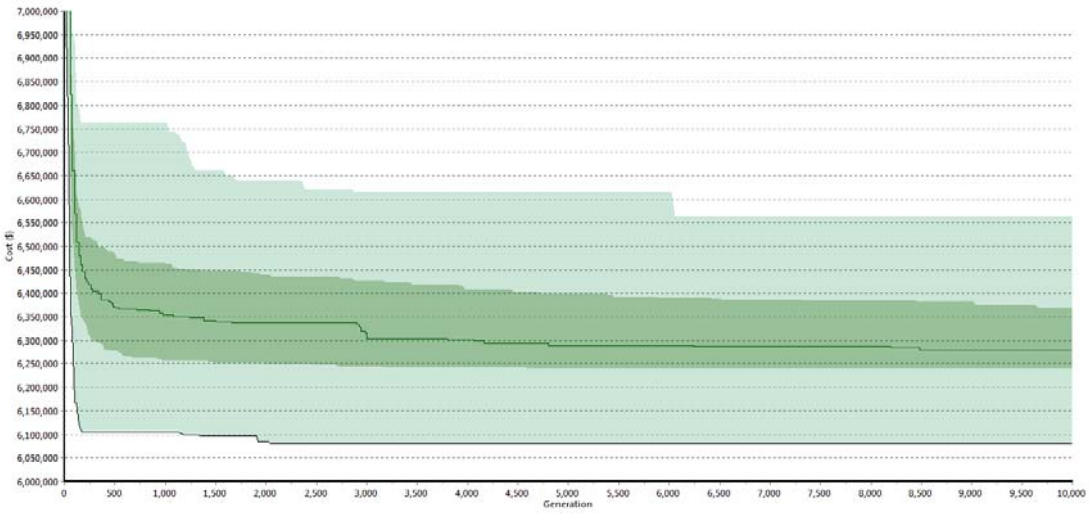

Figure 6-20: Algorithmic Performance: Hanoi - Binary String

These results for the Hanoi are immediately of interest as some of the algorithm runs identify a solution of $6.081m – considerably cheaper than the next best known solution of $6.190m (Kadu, 2008). Altogether just 5% of runs identify this solution with the median remaining high at $6.279m over the 10,000 generations of the optimization.

6.3.4.2. Integer

In contrast to the New York Tunnels, which has a solution space an order of maginitude smaller, the Integer representation can be seen in Figure 6-21 to be outperforming that of the standard binary string. A slight improvement over the binary string representation, the integer runs identified the (newly found) optimal solution on 7% of its runs.

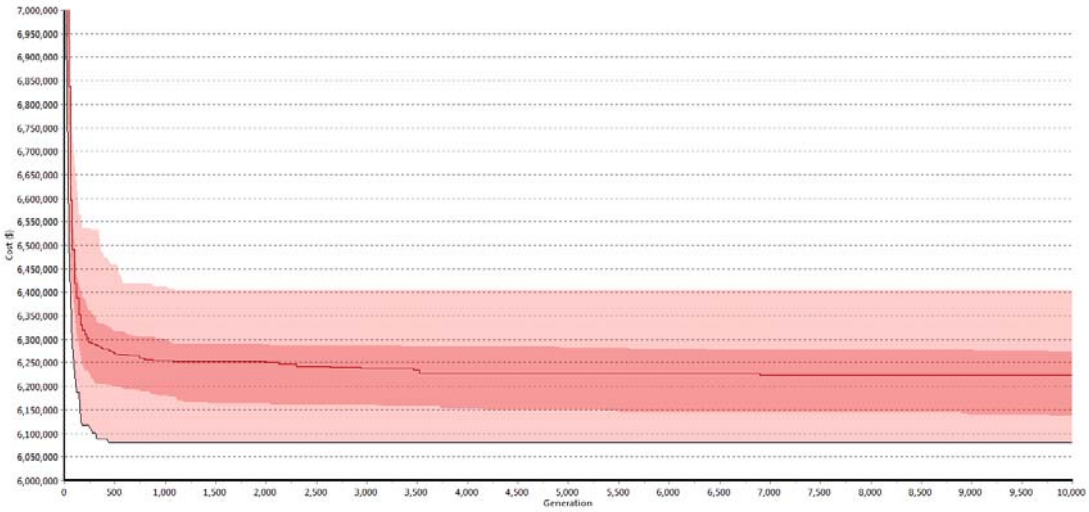

Figure 6-21: Algorithmic Performance: Hanoi – Integer

6.3.4.3. Hybrid Integer

The hybrid integer run results, shown in Figure 6-22, represent a further improvement on the other representations with 28% of runs identifying the best known solution of $6.081m.

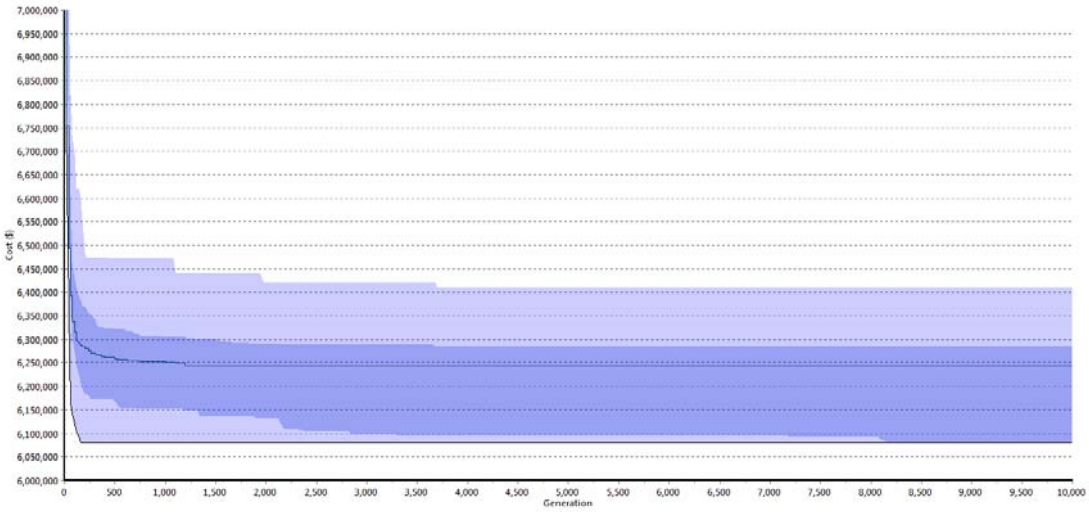

Figure 6-22: Algorithmic Performance: Hanoi - Hybrid Integer

6.3.4.4. Comparative Analysis

The plot of the best results, seen in Figure 6-23, demonstrates that, at its most rapid, the best known solution was identified after 160 generations (15,682 evaluations). This contrasts favourably with the most rapid solutions generated in 53,000 evaluations for the simulated annealing approach of Cunha & Sousa (1999) and 18,000 of Kadu *et al.* (2008) using their modified GA approach.

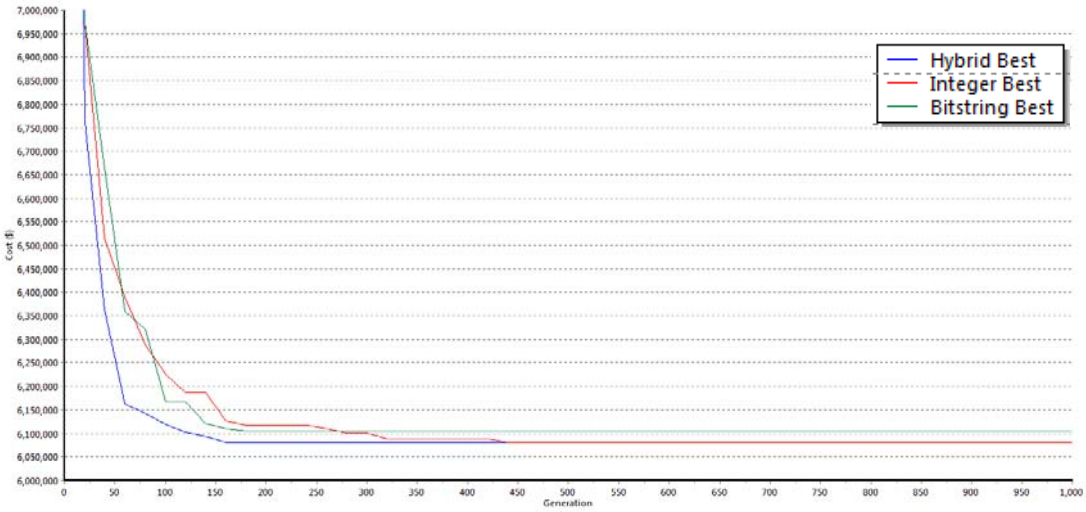

Figure 6-23: Algorithmic Performance: Hanoi - Combined Best

The superiority of the hybrid integer representation is less clear-cut in this example in terms of algorithmic performance. Figure 6-24 shows the aggregated results for all three representations analysed. For clarity, the maximum and minimum curves have been omitted from this figure. It can be seen that the performance of the hybrid integer is generally better than that of the other representations – although the median performance of the integer form is better over the lifetime of the optimization.

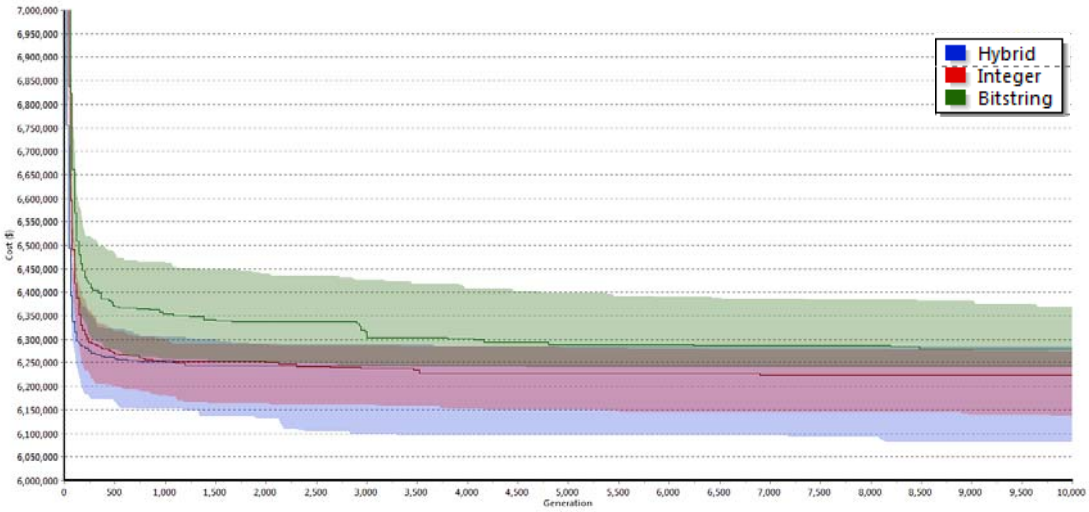

Figure 6-24: Algorithmic Performance: Hanoi - Combined Upper/Lower Quartiles

6.3.4.5. Runtime Performance

However, it can be seen that on this longer chromosome the Hybrid Integer now outperforms the Binary string whilst the Integer representation maintains its lead in performance (Table 6-16).

Chromosome Representation	Average Performance (evaluations per second)	% of best performance
Binary String	6,934.57	87.5%
Integer	7,921.42	100%
Hybrid Integer	7,107.57	89.7%

Table 6-16: Hanoi Runtime Performance

6.3.5. Caching

Despite the larger solution space of the Hanoi problem with respect to New York Tunnels, the number of cache hits has improved dramatically. However, from the graphs of the algorithmic performance (Figure 6-22) it can be seen that there is little improvement in the population in the latter stages of the optimization, which would improve the likelihood of the optimization generating previously encountered individuals.

Cache Size	Runtime performance	Cache Performance		Relative Runtime Performance
	(solutions/second)	*(no. hits)*	*(% of evaluations)*	
None	7,107.57	n/a	n/a	100%
40,000	7,237.85	28,411	1.4%	98.2%
Judy (unlimited)	7,553.21	83,682	4.2%	94.1%

Table 6-17: Cache results: Hanoi

6.3.6. Adaptive Differential Mutation

A similar pattern to the performance of the New York Tunnels problem can be seen when the differential mutator is applied to the Hanoi problem (Figure 6-25) although the rate of improvement appears to be less significant – possibly due to a larger number of pipes proving critical to the system performance.

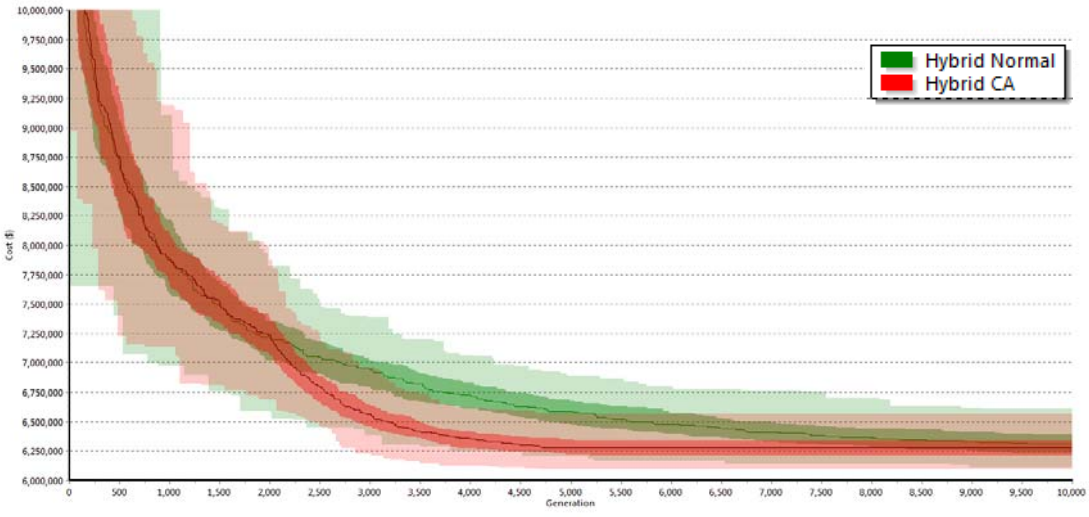

Figure 6-25: Mutation performance comparison - Hanoi

6.3.7. Distributed Performance

As can be seen from the baseline performance figures for the Hanoi problem (Table 6-18), the hydraulic simulation for this 34 pipe problem is seen to produce a greater computational load than that of the 21 pipe, 2 loop New York Tunnels problem (Table 6-7): the baseline evaluation figures being around 20% lower than for the smaller problem when using the Hybrid Integer representation.

Computer	Baseline Performance *(evaluations/second)*	Number of Processor Cores	Theoretical Maximum Throughput *(evaluations/second)*
X	7,108	4	28,432
Y	5,024	2	10,048
Z	4,702	2	9,404
Total	**16,834**	**8**	**47,884**

Table 6-18: Theoretical maximum performance for distributed Hanoi problem

Despite the baseline performance being inferior to that of the New York Tunnels runs, it is immediately apparent with reference to Table 6-19 that the throughput for Computer X has increased, in terms of the number of evaluations perform. It is suggested that this is a result of the two other server computers, Y and Z, taking longer to perform the jobs allocated to them and thus reducing the number of solutions that were distributed across the network (17,436 vs 19,264).

Computer	Baseline Performance *(Evaluations/Second)*	Distributed Performance *(Evaluations/Second)*	
X	7,108	6,022	24,000
		5,988	
		6,103	
		5,887	
Y	5,024	4,422	9,779
		4,283	
		1,074	
Z	4,702	3,401	7,657
		3,332	
		924	
Totals	**16,834**		**41,436**

Table 6-19: Hanoi distributed performance results

This reduction allows the Client computer X to allocate more time to its four server threads, allowing for an increased throughput from those. This leads to the distributed system achieving 86.6% of the the theoretical maximum reported in Table 6-18. If this is the case

then this trend should be continued with the computationally more demanding Piedemonte San Germano network.

6.3.8. Optimal Solution Details

The optimal solution for the Hanoi problem identified by the GA throughout this experimentation does not appear to have been published before. In Table 6-20 the details of this solution are reported and contrasted with that of Cunha & Sousa (1999) which is the lowest cost solution published – albeit with a relaxed Hazen-Williams coefficient where ω = 10.5088 – and the lowest cost solution published that is feasible with EPANET of Zecchin *et al.* (2006) which has a cost of $6,134m. An unmodified version of the standalone EPANET2 software of Rossman (2000) has been used to populate the results in the table.

Diameter						
Pipe	Thesis		Cunha & Sousa (1999)		Zecchin *et al.* (2006)	
	(inches)	*(mm)*	*(inches)*	*(mm)*	*(inches)*	*(mm)*
1	40	1,016	40	1,016	40	1,016
2	40	1,016	40	1,016	40	1,016
3	40	1,016	40	1,016	40	1,016
4	40	1,016	40	1,016	40	1,016
5	40	1,016	40	1,016	40	1,016
6	40	1,016	40	1,016	40	1,016
7	40	1,016	40	1,016	40	1,016
8	40	1,016	40	1,016	40	1,016
9	40	1,016	40	1,016	40	1,016
10	30	762	30	762	30	762
11	24	609.6	24	609.6	24	609.6
12	24	609.6	24	609.6	24	609.6
13	20	508	20	508	16	406.4
14	16	406.4	16	406.4	12	304.8
15	12	304.8	12	304.8	12	304.8
16	12	304.8	12	304.8	12	304.8
17	16	406.4	16	406.4	20	508
18	24	609.6	20	508	24	609.6
19	20	508	20	508	24	609.6
20	40	1,016	40	1,016	40	1016
21	20	508	20	508	20	508
22	12	304.8	12	304.8	12	304.8
23	40	1,016	40	1,016	40	1,016
24	30	762	30	762	30	762
25	30	762	30	762	30	762

Pressure			
Node	Thesis	Cunha & Sousa (1999)	Zecchin *et al.* (2006)
	(metres H_2O)		
1	100.00	100.00	100.00
2	97.14	97.17	97.14
3	61.67	62.00	61.67
4	56.92	57.23	57.08
5	51.02	51.32	51.38
6	44.81	45.07	45.40
7	43.35	43.61	44.01
8	41.61	41.85	42.36
9	40.23	40.44	41.06
10	39.20	39.40	40.11
11	37.64	37.85	38.55
12	34.21	34.43	35.12
13	30.01	30.24	30.91
14	35.52	35.49	37.21
15	33.72	33.44	32.89
16	31.30	30.36	32.16
17	33.41	30.51	41.36
18	49.93	44.29	48.55
19	55.09	55.90	54.33
20	50.61	50.89	50.61
21	41.26	41.58	41.26
22	36.10	36.42	36.10
23	44.52	44.73	44.53
24	38.93	39.03	39.39
25	35.34	35.34	36.18

Pipe	Diameter					
	Thesis		Cunha & Sousa (1999)		Zecchin *et al.* (2006)	
	(inches)	*(mm)*	*(inches)*	*(mm)*	*(inches)*	*(mm)*
26	20	508	20	508	24	609.6
27	12	304.8	12	304.8	12	304.8
28	12	304.8	12	304.8	12	304.8
29	16	406.4	16	406.4	16	406.4
30	12	304.8	12	304.8	16	406.4
31	12	304.8	12	304.8	12	304.8
32	16	406.4	16	406.4	16	406.4
33	16	406.4	16	406.4	16	406.4
34	24	609.6	24	609.6	20	508
Cost	**$6,081,127.54**		**$6,056,370.68**		**$6,134,015.72**	

Node	Pressure		
	Thesis	Cunha & Sousa (1999)	Zecchin *et al.* (2006)
	(metres H_2O)		
26	31.70	31.44	32.55
27	30.76	30.15	31.61
28	38.94	39.12	35.90
29	30.13	30.21	31.23
30	30.42	30.47	30.29
31	30.70	30.75	30.77
32	33.18	33.20	32.04

Pipe: smaller diameter than optimal solution

Pipe: larger diameter than optimal solution

Table 6-20: Comparison of optimal solutions to Hanoi problem

6.4. Piedemonte San Germano

6.4.1. Problem Formulation

The Piedemonte San Germano network model (hereafter PSG) is a real-world network model from a small Italian village in southern Lazio. The network is in a highly looped configuration and is characterised by extremely low flows in some of the loops even at peak demand. The network was introduced by Tricarico *et al.* (2006) in which extensive statistical analysis of the demand characteristics of the network were undertaken and predictions of future demand were made in line with the projected population expansion of the village. Each of the 45 pipes may be duplicated with one of 14 commercially available diameters, or left unduplicated, leading to a search space for this problem of $15^{45} = 8.4 \times 10^{52}$. A minimum head constraint of 20m is applied across the network. The future demand scenarios from Tricarico *et al.* (2006) has been considered for the single-objective problem: a more onerous demand condition than the present day state derived in order to represent predicted future demand. For this future demand scenario, the best-published solution is €31,002 (Tricarico *et al.*, 2006). It should be noted that the costs used in this model relate only to the cost of purchasing the additional pipes required and other factors, such as the cost of installation, are not considered.

6.4.2. Network Configuration

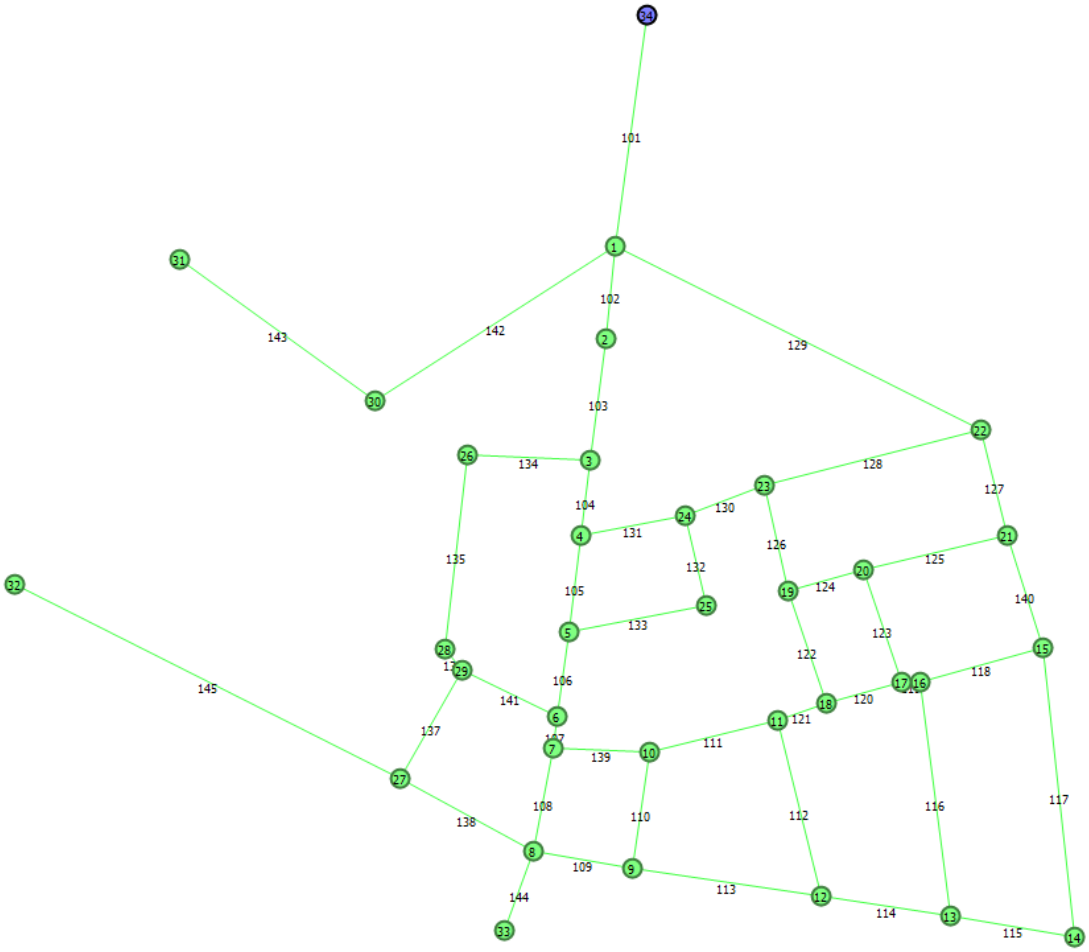

Figure 6-26: Piedemonte San Germano Network Topology

Node ID	Elevation *(metres)*	Demand (litres per second)	Minimum Pressure *(metres H_2O)*
1	120.6	2.158	140.6
2	118.0	1.385	138.0
3	115.4	1.080	135.4
4	113.7	1.190	133.7
5	111.0	1.217	131.0
6	110.0	1.080	130.0
7	108.0	1.080	128.0
8	107.9	1.080	127.9
9	106.9	1.080	126.9
10	108.8	1.096	128.8
11	108.0	1.190	128.0
12	105.9	1.190	125.9
13	104.6	1.134	124.6
14	104.4	3.082	124.4

Node ID	Elevation *(metres)*	Demand (litres per second)	Minimum Pressure *(metres H_2O)*
15	107.4	1.583	127.4
16	107.7	1.080	127.7
17	107.7	1.072	127.7
18	108.0	1.163	128.0
19	109.5	1.147	129.5
20	109.4	1.080	129.4
21	109.2	1.249	129.2
22	111.4	1.292	131.4
23	112.0	1.163	132.0
24	112.4	1.107	132.4
25	109.6	1.080	129.6
26	115.6	1.080	135.6
27	109.0	1.436	129.0
28	111.0	1.249	131.0
29	111.0	1.027	131.0
30	118.2	1.427	138.2
31	122.6	3.501	142.6
32	114.5	3.360	134.5
33	106.9	3.845	126.9

Table 6-21: Piedemonte San Germano Node Characteristics

Reservoir ID	Total Head *(metres)*
34	151.0

Table 6-22: Piedemonte San Germano Reservoir Characteristics

Pipe	From Node	To Node	Diameter *(mm)*	Length (metres)	C-M Friction Factor
1	34	1	80	121.0	0.02
2	1	2	80	52.0	0.02
3	2	3	80	70.0	0.02
4	3	4	80	38.0	0.02
5	4	5	80	50.0	0.02
6	5	6	80	45.0	0.02
7	6	7	80	71.0	0.02
8	7	8	80	65.0	0.02
9	8	9	80	52.0	0.02
10	9	10	60	59.0	0.02
11	10	11	60	60.0	0.02
12	11	12	60	94.0	0.02
13	12	9	80	97.0	0.02

Pipe	From Node	To Node	Diameter *(mm)*	Length (metres)	C-M Friction Factor
14	12	13	80	46.0	0.02
15	13	14	80	66.0	0.02
16	13	16	60	124.0	0.02
17	14	15	80	152.0	0.02
18	15	16	60	63.0	0.02
19	16	17	60	3.0	0.02
20	17	18	60	94.0	0.02
21	18	11	60	28.0	0.02
22	18	19	60	61.0	0.02
23	17	20	60	60.0	0.02
24	19	20	60	115.0	0.02
25	20	21	60	80.0	0.02
26	19	23	60	57.0	0.02
27	21	22	80	54.0	0.02
28	22	23	60	113.0	0.02
29	22	1	80	232.0	0.02
30	23	24	60	44.0	0.02
31	24	4	60	56.0	0.02
32	24	25	50	44.0	0.02
33	25	5	60	76.0	0.02
34	3	26	60	64.0	0.02
35	26	28	80	98.0	0.02
36	28	29	80	10.0	0.02
37	29	27	80	97.0	0.02
38	27	8	80	68.0	0.02
39	7	10	60	51.0	0.02
40	15	21	80	60.0	0.02
41	6	29	60	64.0	0.02
42	1	30	80	179.0	0.02
43	30	31	63	130.0	0.02
44	8	33	80	43.0	0.02
45	27	32	63	224.0	0.02

Table 6-23: Piedemonte San Germano Pipe Characteristics

Pipe option	Diameter (mm)	Cost (€ per metre)
0	None	0.00
1	90.0	33.73
2	110.0	40.60
3	125.0	41.92
4	140.0	46.88
5	160.0	55.54
6	180.0	58.53
7	200.0	65.32

Pipe option	Diameter (mm)	Cost (€ per metre)
8	250.0	82.35
9	280.0	95.39
10	315.0	106.29
11	355.0	131.89
12	400.0	159.12
13	450.0	187.66
14	500.0	219.16

Table 6-24: Piedemonte San Germano Pipe Duplication Options

6.4.3. GA Configuration

Again, the same GA configuration was applied for this model as to the other two networks. The penalty cost was set to €250,000 per metre of head deficit.

6.4.4. Genetic Representation

6.4.4.1. Binary String

As can be seen from

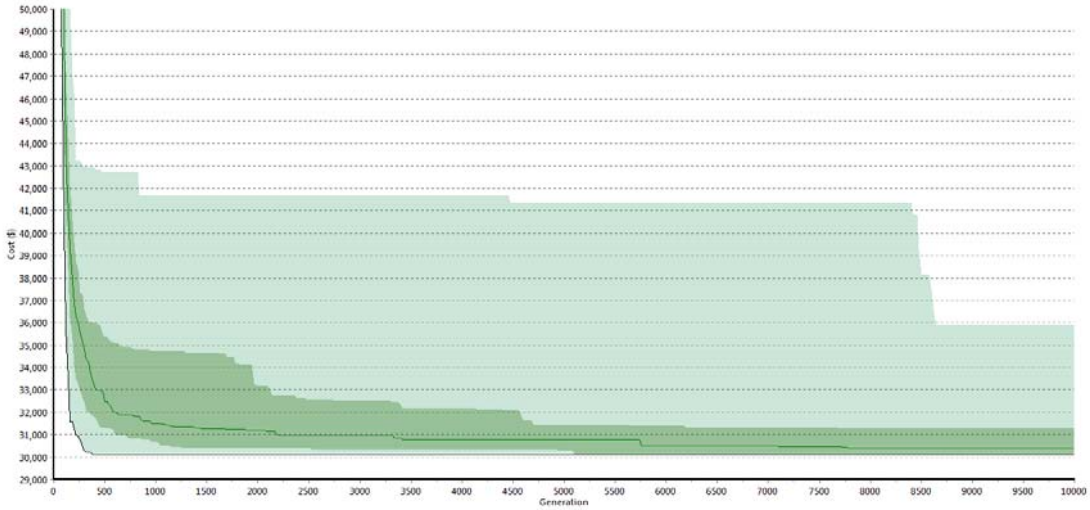

Figure 6-27, some of the runs (37%) converge to a deterministic optimal solution of €30,082. This value is a significant improvement on the previous best result achieved of €31,002 (Tricarico et al,. 2005)– somewhat surprising given that the same basic algorithm had been employed.

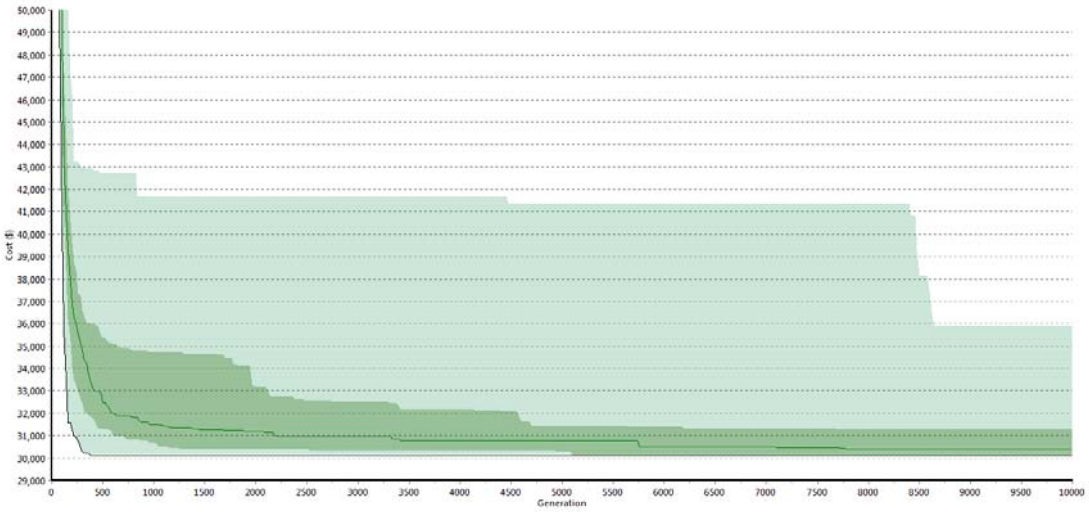

Figure 6-27: Algorithmic Performance: PSG - Binary String

Further investigation into this result revealed that the optimization algorithm employed by Tricarico *et al.* (2005) was compromised in the way that the selection routine was employed. Instead of exerting a positive selection pressure on the algorithm it, effectively, selected organisms at random from the population. However, the algorithm appears to work effectively, albeit extremely slowly, because the NSGA-II algorithm employed has an implicit selection pressure applied through its ranking procedure during each generation. However, this was seen to, in no way, promote the selection of fitter individuals for recombination in the first instance.

6.4.4.2. Integer

As with the Hanoi problem, the integer representation appears to produce inferior results (Figure 6-28) to the binary string representation with 24% of the solutions converging to the newly identified best-known solution.

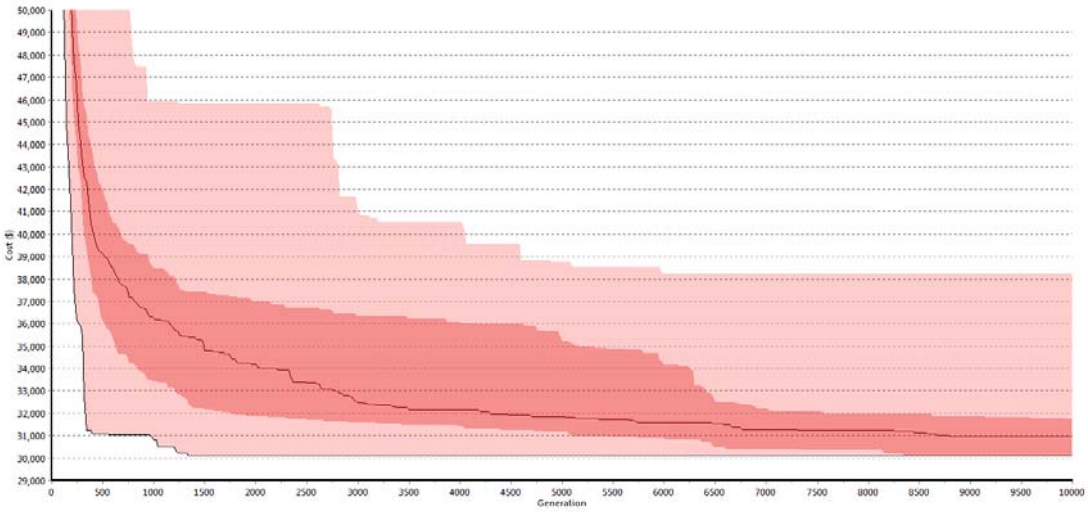

Figure 6-28: Algorithmic Performance: PSG – Integer

6.4.4.3. Hybrid Integer

Once again, the performance of the Hybrid Integer is clearly superior to that of the other representations with 94% of the solutions converging to the best-known solution. The results illustrated in Figure 6-29 demonstrate excellent performance with the median (i.e. 50% of the runs) reaching the optimal solution after just 2,240 generations.

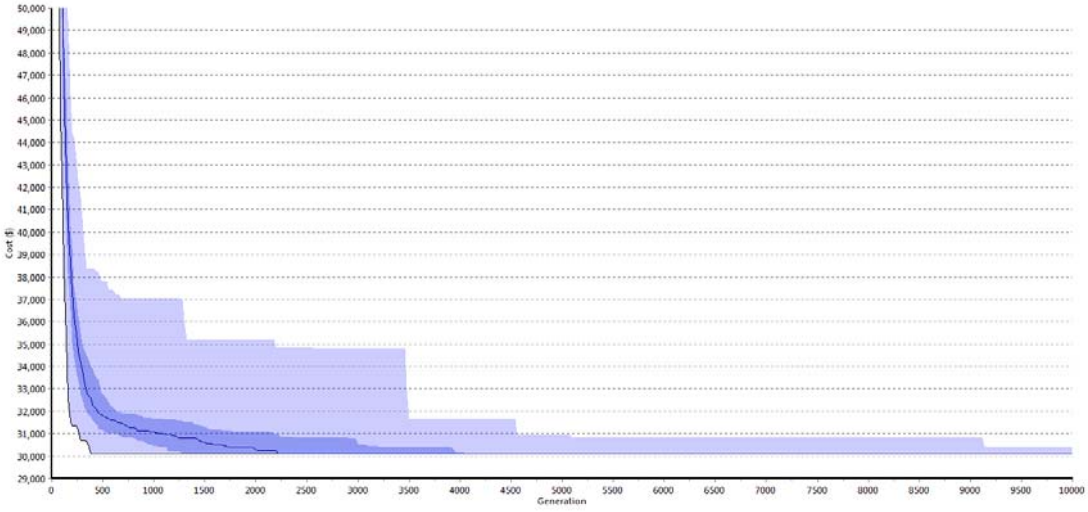

Figure 6-29: Algorithmic Performance: PSG - Hybrid Integer

6.4.4.4. Comparative Analysis

At their best, the hybrid integer and binary string representations identified the best known solution of €30,082 after 380 generations (37,340 evaluations) versus the 1,340 generations (131,420 evaluations) achieved by the integer representation. Tricarico *et al.* (2005) do not identify the performance of their algorithm and so a direct comparison is not possible.

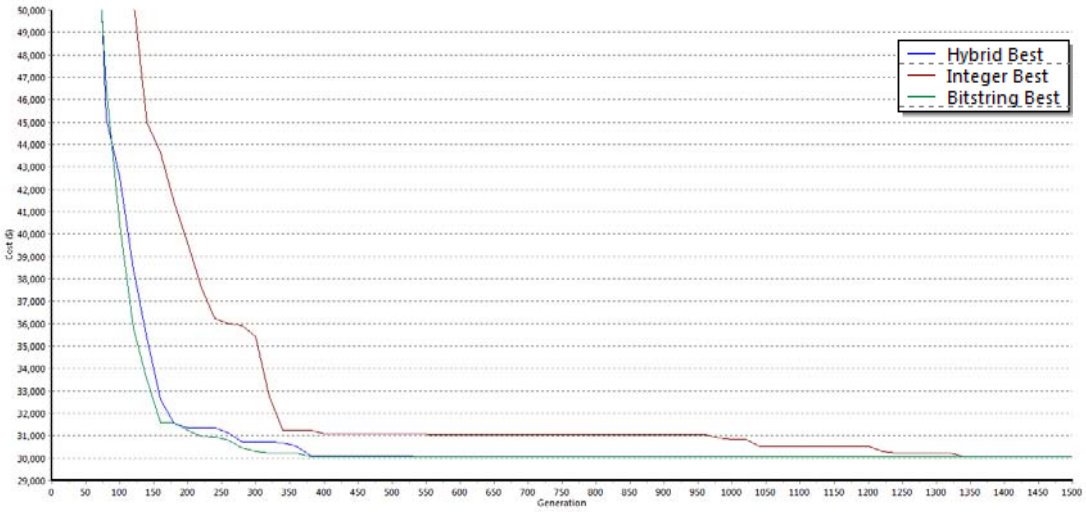

Figure 6-30: Algorithmic Performance: PSG - Combined Best

The aggregated results for the three representations, seen in Figure 6-31, demonstrate clearly the superior average performance of the hybrid integer representation over the other two formats. For clarity, the maximum and minimum curves for each distribution have been omitted in this figure.

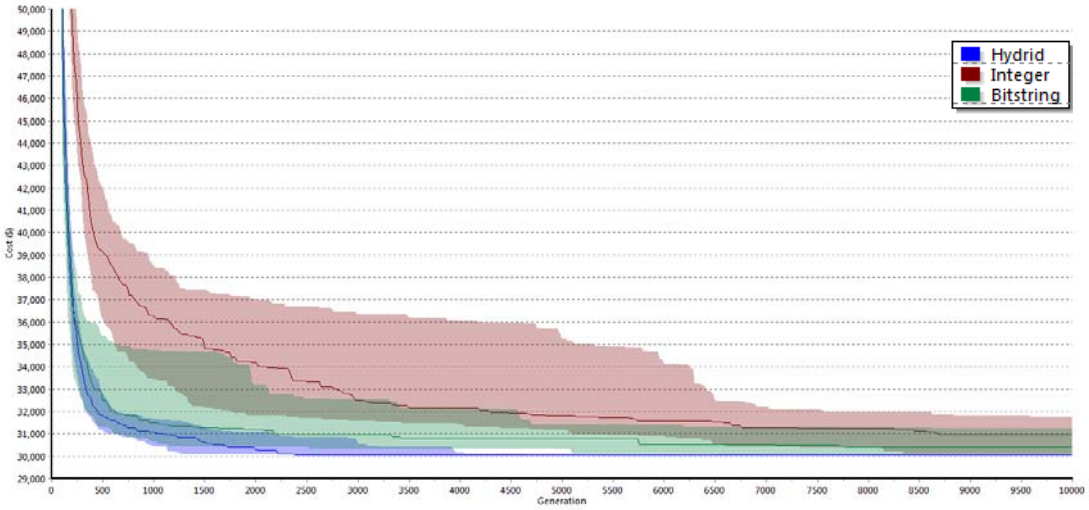

Figure 6-31: Algorithmic Performance: PSG - Combined Upper/Lower Quartiles

6.4.4.5. Runtime Performance

The Integer representation is, once more, the fastest representation with the Binary String and Hybrid Integer returning very similar performances as shown in Table 6-25.

Chromosome Representation	Average Performance *(evaluations per second)*	% of best performance
Binary String	4,129.84	90.0%
Integer	4,589.79	100%
Hybrid Integer	4,056.80	88.4%

Table 6-25: PSG Runtime Performance

6.4.5. Heterozygous Chromosomes

As with the New York Tunnels problem, Piedemonte San Germano has been reconfigured as a heterozygous problem in which the chromosome is encoded to allow 10 pipes to be modified at once. This selection of 10 pipes is an arbitrary decision made to limit the scope of the optimization and any number of pipes may have been chosen in this fashion. Thus the chromosome, in this example, comprises 10 pairs of genes. The first gene being allowed to vary between 1and 45 to represent the pipe to be modified. The second gene of the pair encodes the pipe diameter, as previously. Thus for the Piedemonte San Germano problem, the total chromosome length is reduced from 45 genes to 20.

6.4.5.1. Binary String

Figure 6-32 shows the results obtained for the heterozygous, binary string runs. This same data is overlain with that obtained with the conventionally-coded binary string in Figure 6-33. A mere 7% of the heterozygous solutions found the best-known optimal solution for the problem, compared to 37% of the conventionally coded solution. 88% of the runs converged to a solution of €32,061.60. It is not clear why there is such a preference for this result with the heterozygous configuration.

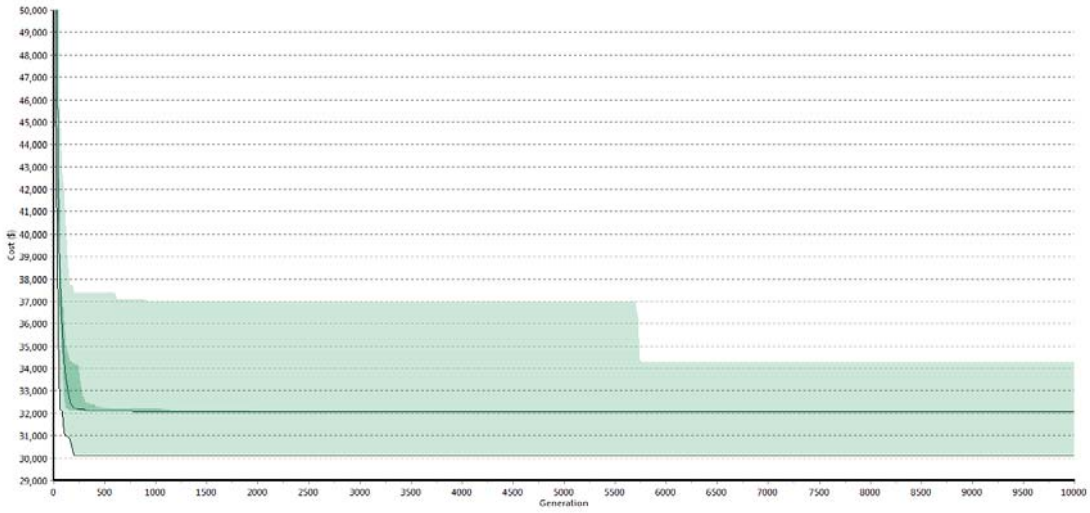

Figure 6-32: Algorithmic Performance: PSG – Heterozygous Binary String

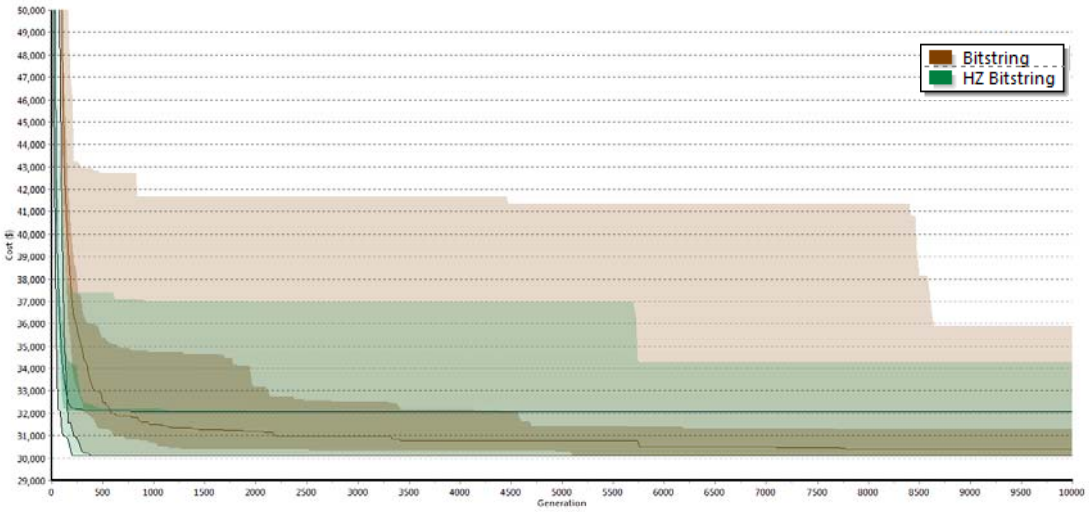

Figure 6-33: Algorithmic Performance: PSG- Heterozygous Binary String results overlain with conventional results

6.4.5.2. Integer

Nineteen percent of the runs of the heterozygous Integer algorithm identified the optimal solution, compared to 24% for the conventional-coded equivalent. Of the remainder, 80% converged to the solution of €32,061.60 as can be seen from Figure 6-34. These results can be compared with those of the conventional encoding in Figure 6-35 in which the conventional results are overlain.

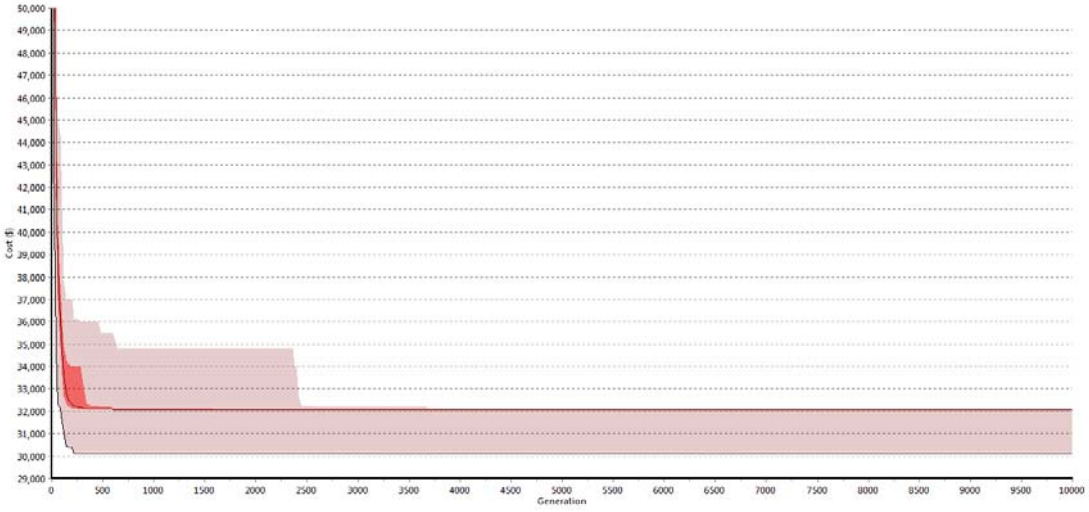

Figure 6-34: Algorithmic Performance: PSG – Heterozygous Integer

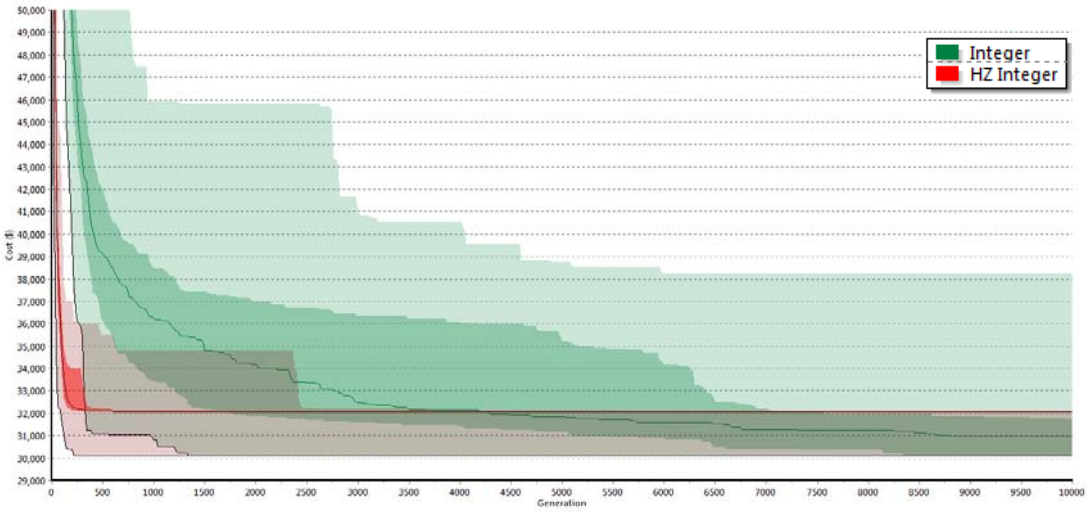

Figure 6-35: Algorithmic Performance: PSG- Heterozygous Integer results overlain with conventional results

6.4.5.3. Hybrid Integer

In contrast to the conventional algorithm where 94% of the hybrid integer-based solutions converged to the best-known optimum, the heterozygous algorithm could manage this in only 19% of cases (Figure 6-36). The remainder of the runs all converged to the same solution of €32,061.60 as has been seen in the optimizations of the other heterozygous representations. Figure 6-37 shows the heterozygous hybrid integer results overlain with those of the conventional encoding.

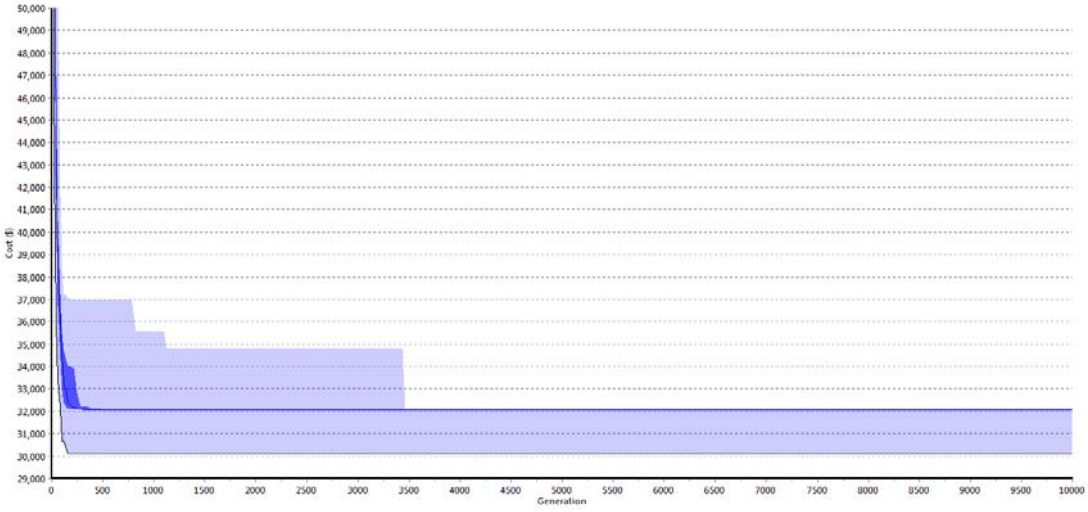

Figure 6-36: Algorithmic Performance: PSG – Heterozygous Hybrid Integer

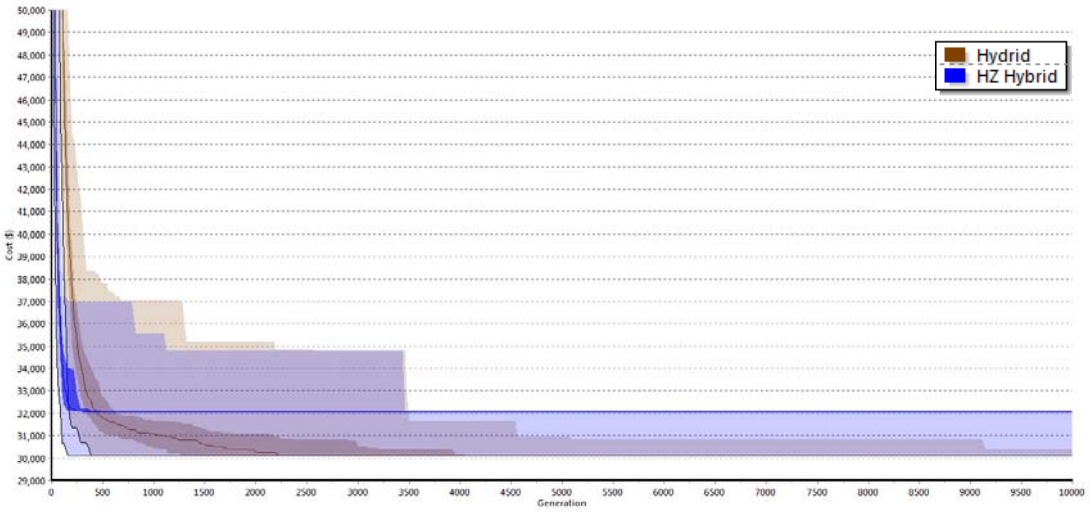

Figure 6-37: Algorithmic Performance: PSG- Heterozygous Hybrid Integer results overlain with conventional results

6.4.5.4. Comparative Analysis

Figure 6-38 illustrates the best performance for the runs of each of the three representations employed in the heterozygous configuration. It can be seen clearly that the heterozygous representations identify the optimal solution quicker for each of the representations:

- hybrid integer: 160 generations (15,682 evaluations) vs. 380 generations (37,340 evaluations).
- binary string: 200 generations (19,602 evaluations) vs. 380 generations (37,340 evaluations).
- integer: 220 generations (21,562 evaluations) vs. 1,340 generations (131,322 evaluations).

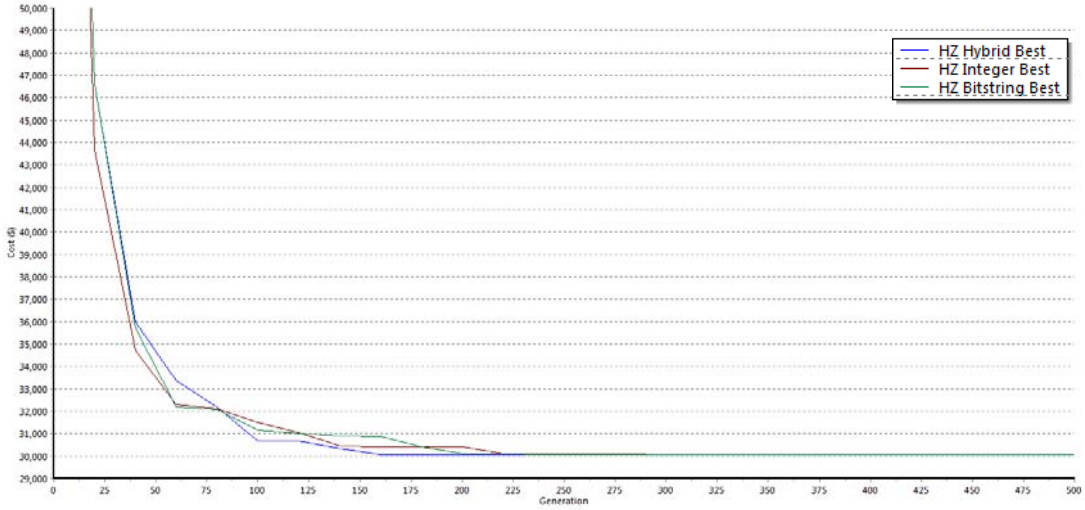

Figure 6-38: Algorithmic Performance: PSG – Heterozygous Combined Best

The combined performance for the three heterozygous representations is portrayed in Figure 6-39. For clarity, the minimum and maximum curves have been omitted from this graph which shows the upper/lower quartiles and the median for the performance of the combined runs. This graph shows that there is little substantial difference between the three representations, although the hybrid integer can be seen to marginally outperform the other two representations. In addition, the convergence of the algorithms, in most cases, to the solution of €32,061.60 is further highlighted.

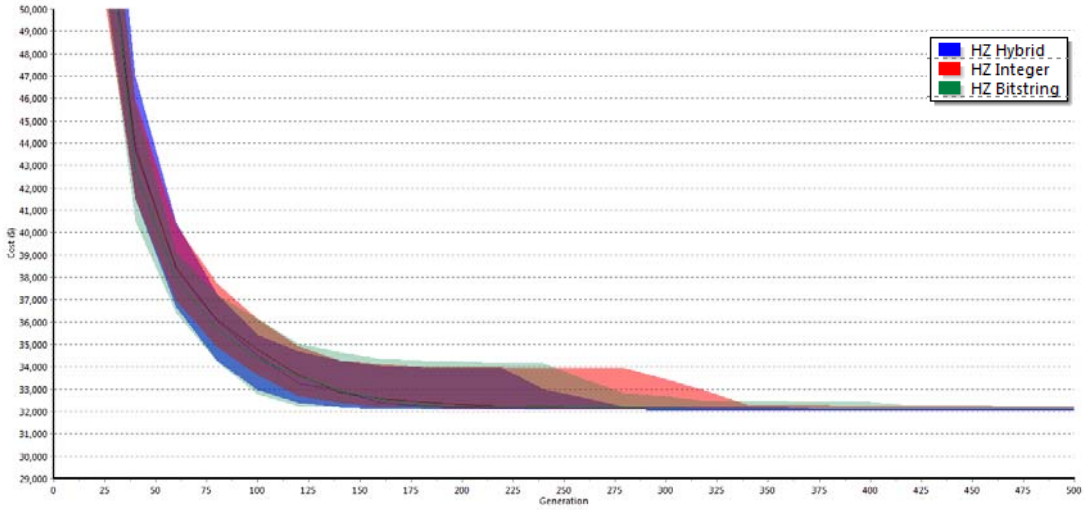

Figure 6-39: Algorithmic Performance: PSG - Combined Upper/Lower Quartiles

With reference to the equivalent New York Tunnels result (Figure 6-17) it would appear that the relative algorithmic performance advantage of the hybrid gene diminishes as

chromosome length decreases. In addition, the algorithmic performance of the heterozygous representation of the problem improves, relative to the full representation, as the number of decision variables grows, as might be expected.

6.4.5.5. Runtime Performance

As anticipated, the longer chromosomes of the conventional representation cause it to show significantly reduced performance with respect to the heterozygous approach – as seen in Table 6-26. Once more, as with the other analyses, the integer runtime proves to be the quickest representation with the hybrid integer and binary string having roughly equal runtime performance.

Chromosome Representation	Conventional		Heterozygous	
	(evaluations per second)	% of best	(evaluations per second)	% of best
Binary String	4,129.84	71.2%	5,471.21	94.8%
Integer	4,589.79	79.6%	5,770.00	100%
Hybrid Integer	4,056.80	70.3%	5,543.54	96.1%

Table 6-26: PSG Heterozygous Runtime Performance vs. Conventional Performance

6.4.6. Caching

Although the Piedemonte San Germano network optimization has by far the largest solution space of the three steady state algorithms investigated, it also produces the best caching results as seen in Table 6-27. However, as with the Hanoi problem part of this success would appear to be because the optimization converges quickly – at least when using the hybrid integer representation – to the optimal solutions. As a consequence, an increasing proportion of the population is likely to be encountering repeatedly as the algorithm proceeds.

Cache Size	Runtime performance	Cache Performance		Relative Runtime Performance
	(solutions/second)	*(no. hits)*	*(% of evaluations)*	
None	4,056.80	n/a	n/a	100%
40,000	4,221.44	44,122	2.2%	96.1%
Judy (unlimited)	4,371.55	155,782	7.8%	92.8%

Table 6-27: Cache results: Piedemonte San Germano

6.4.7. Adaptive Differential Mutation

Running the Piedemonte San Germano model with the Adaptive Mutation caused some difficulty. Firstly, the model did not optimize well with the steady-state genetic algorithm – the prior runs of this problem have been performed using a generational GA whilst the

Adaptive Mutation is coded to work only with a steady-state optimization. As can be seen in Figure 6-40, the runs converged to solutions around €60,000 – almost double the best-known solution of around €30,082. Because of the considerably larger search space for this problem, the introduction of the differential mutator was delayed until 10,000 iterations or 20,000 evaluations. The above figure clearly shows that the mutator continues to play a positive part in promoting the convergence of the population although, with the increased search space, the effect appears to be much diminished compared to the other problems. This phenomenon is likely to result from the Piedemonte San Germano being less sensitive to be the value of a few "critical genes" – i.e. pipe reinforcement selections – than the other, smaller models.

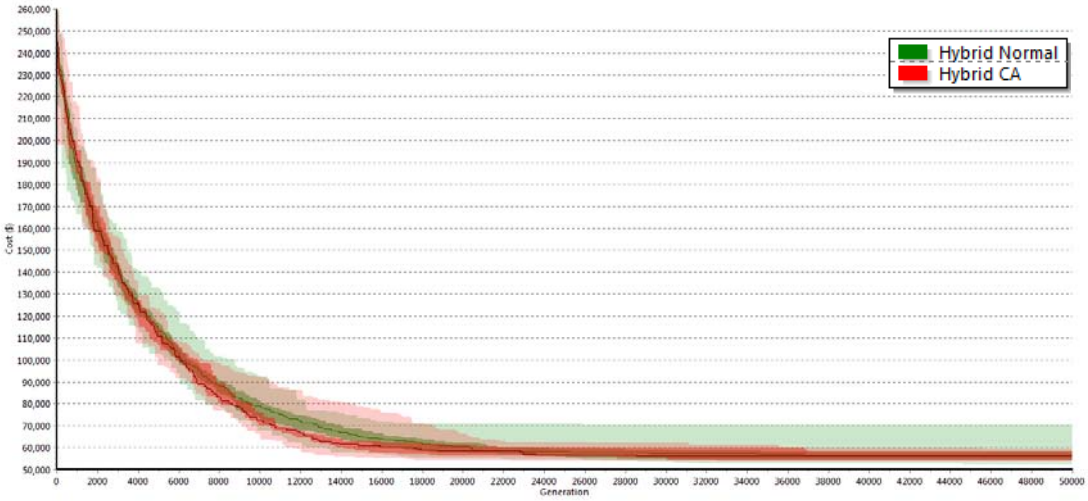

Figure 6-40: Mutation performance comparison - Piedemonte San Germano

6.4.8. Distributed Performance

The Piedemonte San Germano problem is the most complex of the three networks analysed in this chapter and shows a baseline performance approximately half that of the New York Tunnels network (Table 6-7).

Computer	Baseline Performance *(evaluations/second)*	Number of Processor Cores	Theoretical Maximum Throughput *(evaluations/second)*
X	4,057	4	16,228
Y	3,043	2	6,086
Z	2,515	2	5,030
Total	**9,615**	**8**	**27,344**

Table 6-28: Theoretical maximum performance for distributed Piedemonte San Germano problem

The results presented in Table 6-29 appear to confirm the trend that continuing reduction in network load results in an increase in throughput for the distributed system. Here it can be seen that, relative to the theoretical maximum determined in Table 6-28, the system as a whole achieves 91% performance – continuing the increases seen in comparison to the other, smaller network simulation problems.

Computer	Baseline Performance *(Evaluations/Second)*	Distributed Performance *(Evaluations/Second)*	
X	4,057	3,755	14,004
		3,424	
		3,320	
		3,505	
Y	3,043	2,620	5,973
		2,581	
		772	
Z	2,515	2,152	4,923
		2,073	
		698	
Totals	**9,615**		**24,900**

Table 6-29: Piedemonte San Germano distributed performance results

6.5. Conclusions

The hybridized integer gene introduced in Chapter 4.2.3 is demonstrated to be the most efficient representation for the single-objective hydroinformatics applications analysed in terms of algorithmic performance – clearly beating the conventional binary string and integer representations. However, it is shown to be slower than both the conventional binary string the integer representation for the simpler problems. However, for the Piedemonte San Germano example, which has a chromosome length of 45 genes, the computational performance advantage of the conventional binary string is reversed, implying that, as might be expected, the overhead of managing binary strings increases with the string length. In Chapter 7, these representations will be evaluated on a fourth problem with a much longer chromosome. It should be noted also that the version of the hybridized integer gene employed is Gray-coded and this is an additional overhead on the computational performance – making the performance of this novel representation more impressive. In addition, it is believed that these optimizations have identified new, best known solutions for both the Hanoi ($6.081m) and Piedemonte San Germano (€30,082) network problems whilst matching the best known solution for the New York Tunnels problem ($38.644m - Meier *et al.*, 2003).

The application of caching to these problems is shown to be mostly ineffectual in terms of runtime savings because of the relatively trivial nature of the hydraulic computation. It is encouraging though that the efficiency of the Judy cache routines is evident even for

relatively short chromosome lengths – demonstrating improvements in performance even as the chromosome length improves – though this is likely due to the nature of the individual problems and their convergence behaviour.

Employing a mutator that can "learn" from the improvements occasioned by other mutations is shown to have a beneficial effect on the performance of the algorithm. However, the effect of the Adaptive Differential Mutator is constrained significantly by the number of "critical genes" within a specific problem. However, given that the computational cost of maintaining the statistics associated with the mutations is relatively trivial, it is possible that it will assist larger optimizations as well as providing valuable, additional information on the search space by identifying genes that have a particular influence on the solution.

Chapter 7. Multiple Objective Optimization Problems

7.1. Introduction

The network optimization problems presented in Chapter 6 have been reformulated as multiple-objective optimization problems. The single objective problems illustrated previously accommodated a second objective through the use of a "penalty cost" function which penalises the violation of one or more additional constraints. An alternative approach is to model each constraint as an objective in its own right. Multiple-objective algorithms do not converge towards a single solution rather towards a Pareto-optimal front that represents the trade-off between the objectives as illustrated in Figure 7-1.

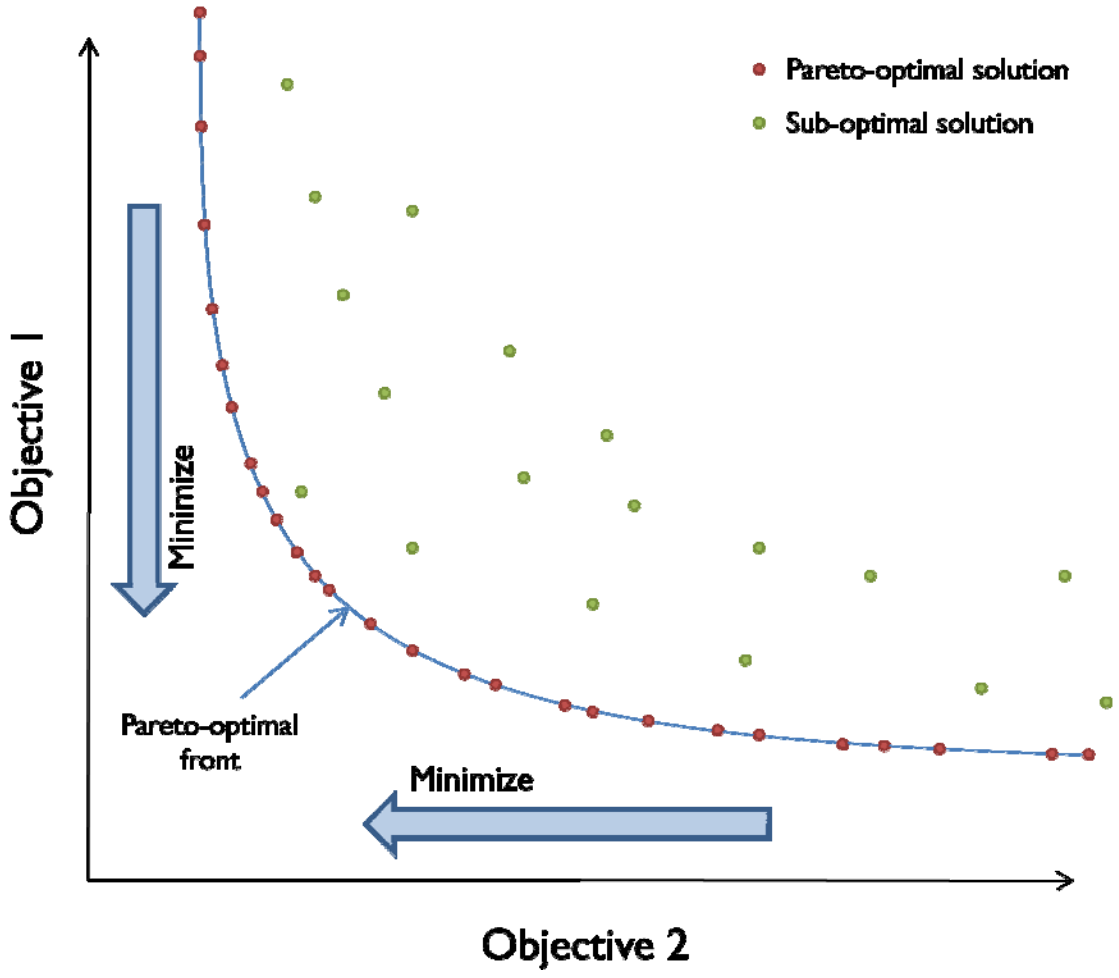

Figure 7-1: Multiple Objective Pareto-Optimal Front

In the example illustrated in Figure 7-1, a two objective problem, both objectives are to be minimized thus the optimization algorithm seeks to drive the Pareto front into the bottom-left corner of the figure.

The multi-objective algorithm employed in this analysis is the Non-Dominated, Sorted GA-II (NSGA-II) of Deb *et al.* (2001). This algorithm is a refinement of Srinivas & Deb's (1994) NSGA algorithm and is noteworthy for being one of the earliest multiple-objective algorithms that is self-tuning. Diversity in the population is managed through the use of a Crowding Sort which preferentially preserves members of the population who are well spaced from their neighbours along the Pareto fronts as they are generated.

Given that multiple objective algorithms produce Pareto-fronts containing multiple, non-dominated solutions each, in its own right, an optimal solution it is more difficult to compare the quality of results obtained through different techniques. To this end, two multiple objective performance metrics introduced by Zitzler & Thiele (1999) have been applied to the obtained Pareto-fronts to gauge their relative strengths: space and coverage metrics.

Space Metric

The "size of space covered" metric *S* evaluates the hypervolume enclosed by the points on the Pareto-optimal front. For a problem in which both objectives (cost and aggregate head deficit) are to be minimised – as is the case here – the *S* metric represents a smaller volume as the quality of the front improves (i.e., the lower the value, the better the front is). This is a relatively simplistic measure performance and, of itself, is not a good measure of how good a front is relative to another.

Coverage Metric

To achieve this comparison between fronts, a coverage metric, *C*, is employed. This metric expresses the proportion of points from one front that dominate those from another. Thus, a *C* value of 1.0 indicates that all of the points in Pareto front *A* are equal to or dominate those in Pareto front *B*. This gives an acceptable measure of the relative strength of the Pareto-fronts but does nothing to inform about how good a front is in absolute terms.

$$C(A,B) = \frac{\left|\{b \in B \,|\, \exists a \in A : a \preceq b\}\right|}{|B|} \qquad \text{ix)}$$

The analyses presented in this chapter are performed on 100 runs of each algorithm. As well as a graphical analysis, the *C* and *S* metrics are applied to each of the Pareto fronts developed

from these runs after 20, 100 and 1,000 generations in order to gauge the progress of the algorithms throughout the optimization.

Each of the problems is formulated with the two objectives in equations x)and xi) thus:

$$Minimize: C_{inf} = f(D_1, .., D_{N_l}) = \sum_{j=1}^{N_l} C(D_j, L_j) \quad \text{x)}$$

$$Minimize: T = f(H_1, .., H_{N_n}) = \sum_{i=1}^{N_n} \max(0; H_{i,min} - H_i) \quad \text{xi)}$$

$$D_j \in D \ (j = 1, .., N_d) \quad \text{xii)}$$

where: C_{inf} is the total infrastructure cost, N_l is the number of links in the network for which reinforcement is an option, $C(D_j,L_j)$ is the cost of the j^{th} pipe with diameter D_j (chosen from a discrete set of available diameters D) and length L_j. T is the total head deficit (negative if a pressure surplus exists), H_i is the pressure head at node i (as computed by the hydraulic solver), $H_{i,min}$ is the minimum pressure head requirement sufficient to fully satisfy the demand at node i and N_n is the number of nodes in the network. N_d is the number of decision variables in the optimization.

Distributed performance is not analysed in this chapter, as the sole differential in performance will be the runtime of the genetic algorithm itself. As far as the server computers involved in deEPANET are concerned, there is no difference to producing solutions for a multiple-objective algorithm rather than a single objective.

7.2. New York Tunnels

7.2.1. Genetic Representation

7.2.1.1. Binary String

A scatter plot illustrating the results for the binary string representation can be seen in Figure 7-2. This figure shows the distribution of solutions for the 100 runs after 20, 100 and 1,000 generations. After 20 generations (2,000 evaluations), none of the runs had identified the

best-known feasible single objective solution of $38,643,500 (Meier *et al.*, 2003), whilst 100% had identified the solution at the other extreme which, for an investment of $0 results in an aggregate head deficit of 353.13m. Following 100 generations, the success rate for the optimal solution has improved to 3% of runs and ultimately at the end of the runs, after 1,000 generations, 18% of runs.

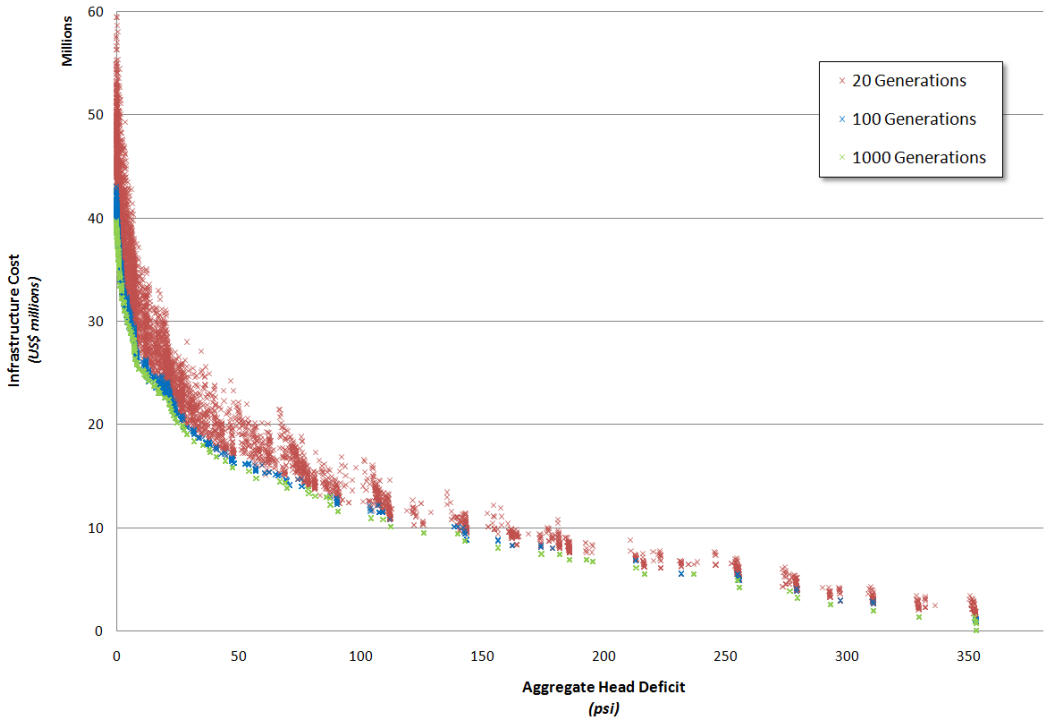

Figure 7-2: New York Tunnels – Multiple Objective Binary String Results

7.2.1.2. Integer

The results from the integer representation are shown in Figure 7-3. After 20 generations none of the runs had identified the single objective optimal solution, however all had identified the other extreme of the distribution. By the completion of 1,000 generations, 14% of the runs had continued to identify the optimal solution – fewer than for the binary string representation.

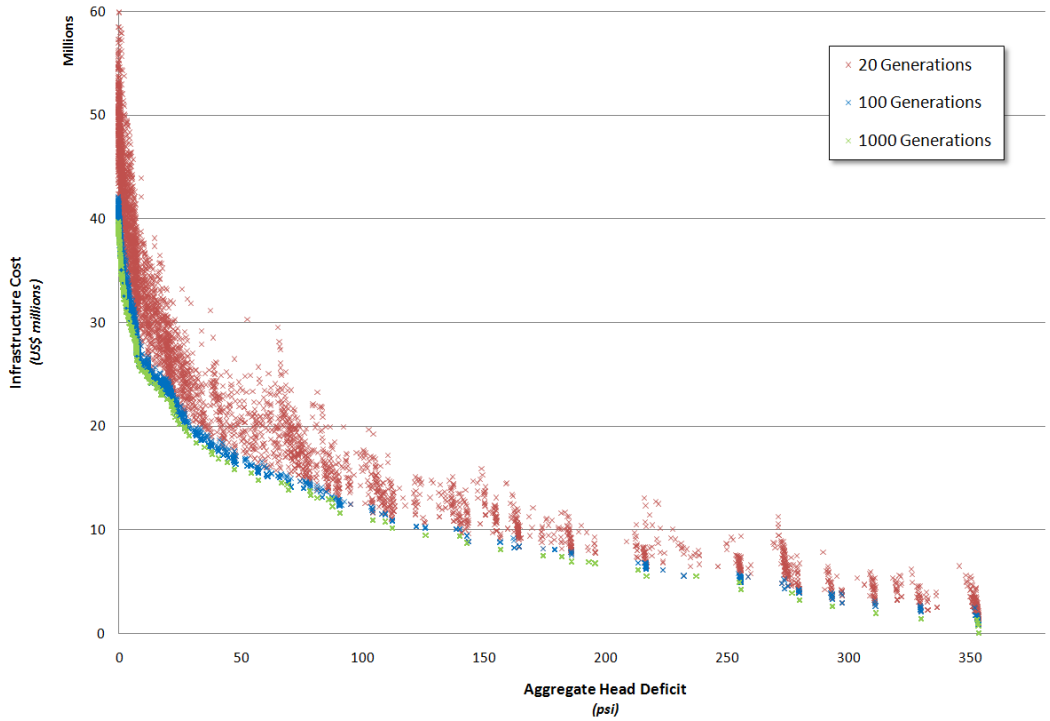

Figure 7-3: New York Tunnels – Multiple Objective Integer Results

7.2.1.3. Hybrid Integer

The results for the Hybrid Integer representation are shown in Figure 7-4. After 1,000 generations, merely 3% of the runs had identified the single-objective optimal solution of $38.644m: a considerably inferior result to the other two representations. In fact, by 100 generations (10,000 evaluations), *none* of the runs had encountered this solution.

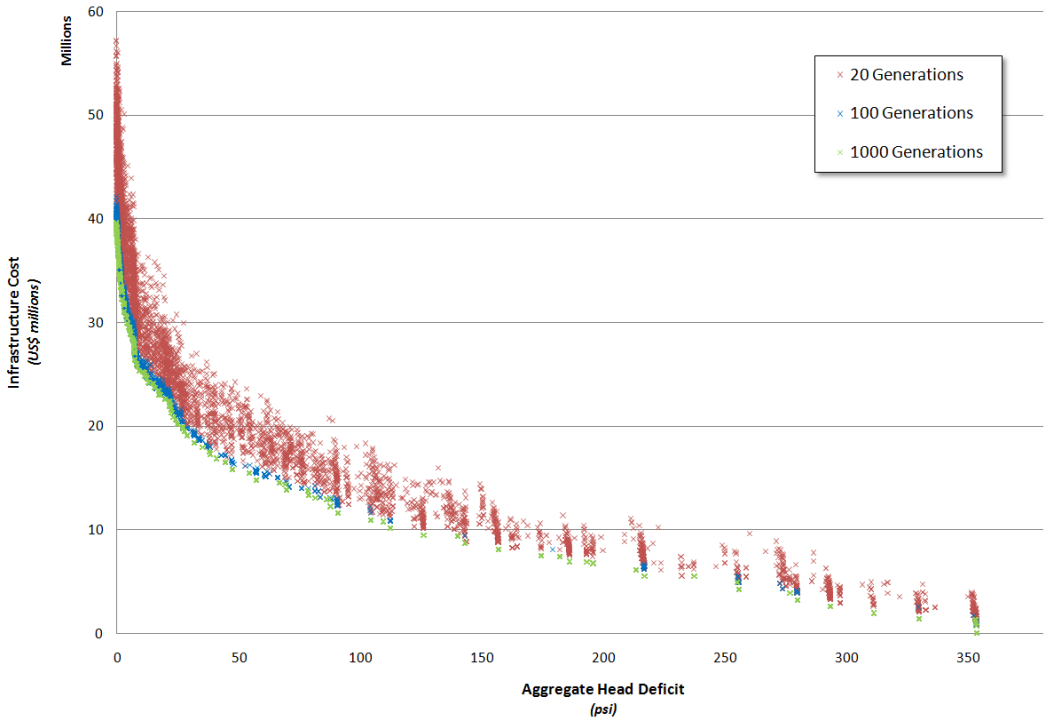

Figure 7-4: New York Tunnels – Multiple Objective Hybrid Integer Results

7.2.1.4. Comparative Analysis

Space Metrics

Examination of the *S* space metrics (Figure 7-5) for these runs suggests that the binary string representation outperforms both the integer and hybrid integer versions.

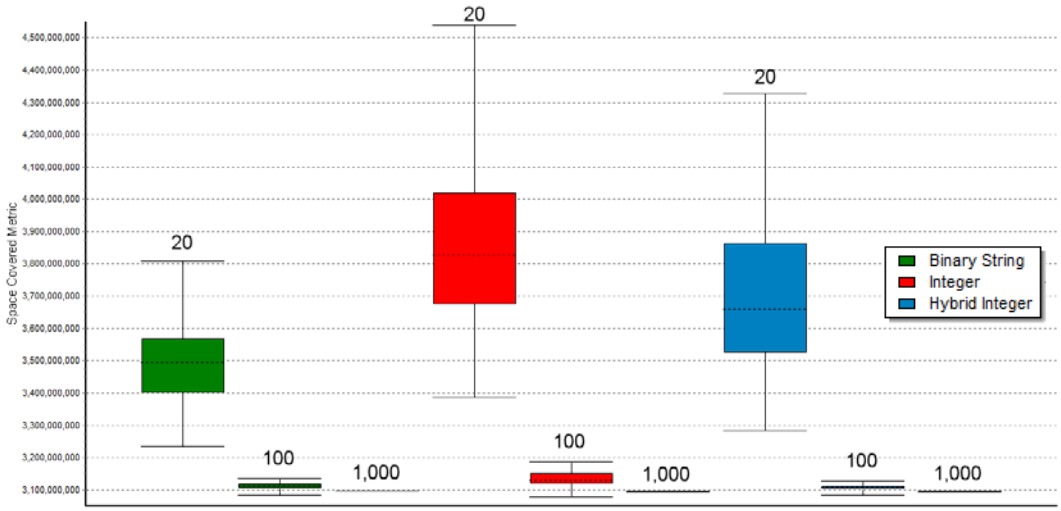

Figure 7-5: Box plots of *S* metric for Multiple Objective New York Tunnels after 20, 100 & 1,000 generations

From the figure, it can be seen that the binary string (green) Box plots indicates that these runs were more consistent and generally produced lower *S* values – which is preferable for a minimization problem such as this. After 1,000 generations, all of the representations are seen to be occupying a similar proportion of the solution space.

Coverage Metrics

The coverage metrics for the runs were calculated for the Pareto-optimal fronts obtained after 20, 100 and 1,000 generations of each of the 100 runs undertaken for each representation. The aggregate values obtained for the metric *C(A,B)* are listed in Table 7-1 to Table 7-3 for 20, 100 and 10,000 generations respectively. The Binary String representation can be seen in Table 7-1to be outperforming that of the integer and hybrid integer – covering 65.3% and 59.3% respectively.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.138	0.388	0.311
		Mean	**0.387**	**0.653**	**0.593**
		Max	0.652	0.980	0.943
	Integer	Min	0.000	0.041	0.000
		Mean	**0.234**	**0.453**	**0.328**
		Max	0.550	0.900	0.736
	Hybrid Integer	Min	0.000	0.171	0.022
		Mean	**0.263**	**0.551**	**0.438**
		Max	0.485	0.940	0.792

Table 7-1: *C* metrics for Multiple Objective New York Tunnels after 20 generations

The results shown in Table 7-2 demonstrate that after 100 generations, the variation in coverage between the representations is diminished and the gap between the binary string and the hybrid integer is reduced – though the integer representation can be seen to continue to be the weakest performing of the representations.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.073	0.120	0.020
		Mean	**0.215**	**0.258**	**0.167**
		Max	0.375	0.343	0.289
	Integer	Min	0.096	0.163	0.020
		Mean	**0.215**	**0.260**	**0.159**
		Max	0.354	0.364	0.330
	Hybrid Integer	Min	0.110	0.163	0.060
		Mean	**0.249**	**0.282**	**0.187**
		Max	0.385	0.414	0.320

Table 7-2: *C* metrics for Multiple Objective New York Tunnels after 100 generations

The low values seen in Table 7-3, for all combinations of representation, suggest that after 1,000 generations, that all of the populations have converged to very similar Pareto-fronts – which can be assessed visually with reference to the distribution of green dots in Figure 7-2 to Figure 7-4 - and that there is no advantage to having started with one representation or the other.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.000	0.000	0.000
		Mean	**0.018**	**0.016**	**0.022**
		Max	0.070	0.050	0.050
	Integer	Min	0.010	0.000	0.020
		Mean	**0.056**	**0.031**	**0.054**
		Max	0.110	0.090	0.100
	Hybrid Integer	Min	0.000	0.000	0.000
		Mean	**0.031**	**0.024**	**0.036**
		Max	0.110	0.080	0.090

Table 7-3: *C* metrics for Multiple Objective New York Tunnels after 1,000 generations

7.2.1.5. Runtime Performance

The figures presented in Table 7-4 demonstrate that the runtime performance of the multiple objective GA is significantly diminished over that of the single objective optimization results (Table 6-7). This difference is accounted for by the additional complexity of the NSGA-II algorithm that requires a number of sorting routines to be applied to the population, according to the number of objectives being considered.

Chromosome Representation	Average Performance *(evaluations per second)*	% of best performance
Binary String	1,868.98	86.3%
Integer	2,164.93	100%
Hybrid Integer	1,886.79	87.2%

Table 7-4: New York Tunnels Multiple Objective Runtime Performance

7.2.2. Heterozygous Chromosomes

7.2.2.1. Binary String

The heterozygous binary string results, presented in Figure 7-6, are broadly comparable with those of the normal representation with the exception that, visually, they exhibit a wider distribution of results for the 20 and 100 generation points. By the end of the optimizations, 11% of the runs had identified the single-objective optimal solution compared to 18% of runs employing the normal encoding of the problem.

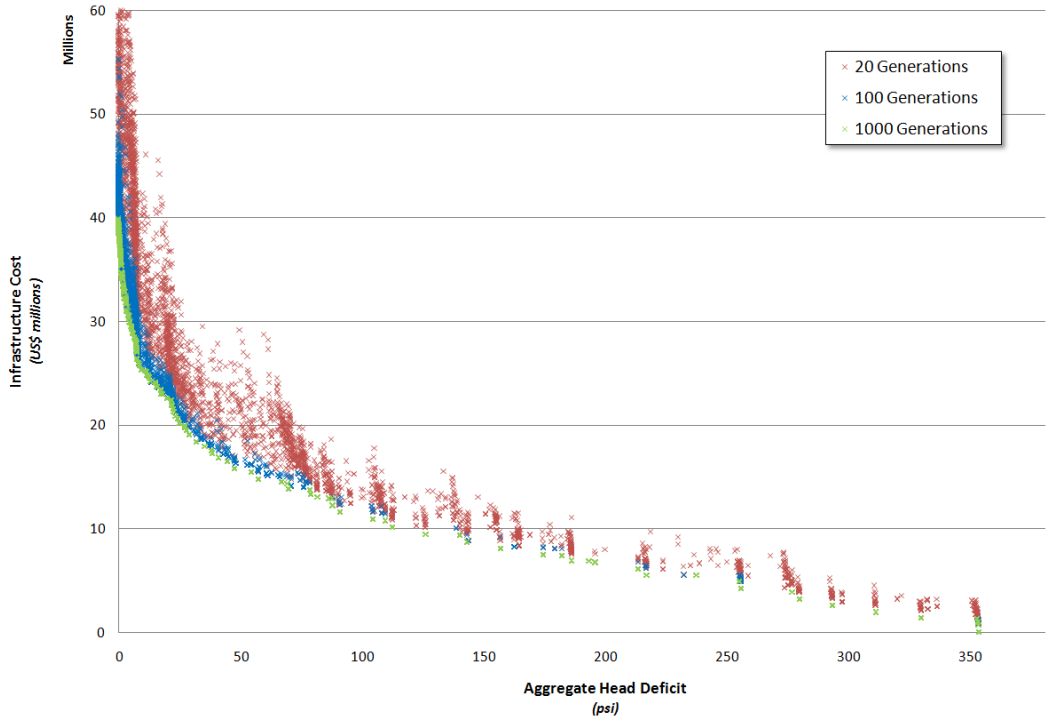

Figure 7-6: New York Tunnels – Multiple Objective Heterozygous Binary String Results

7.2.2.2. Integer

Figure 7-7 illustrates the heterozygous integer results which appear to be very similar to those obtained with the standard representation, albeit with a less tightly confined distribution for the solutions found after 100 generations. Over the lifetime of the optimization, 20% of the

runs identified the single-objective optimal solution compared with 14% for the normal encoding.

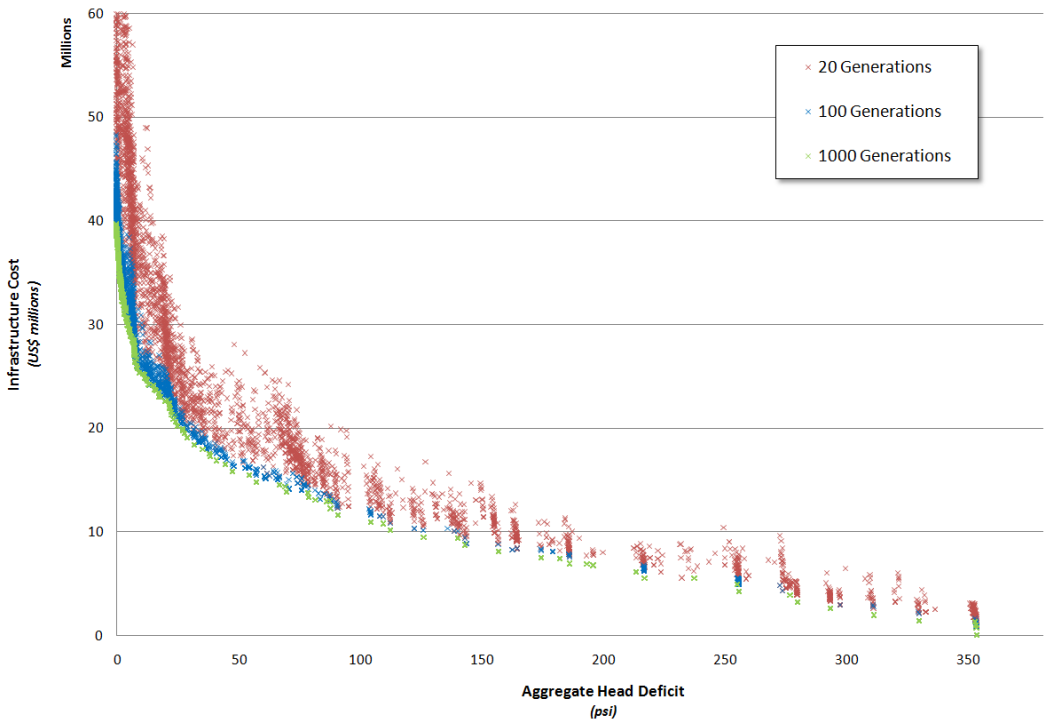

Figure 7-7: New York Tunnels – Multiple Objective Heterozygous Integer Results

7.2.2.3. Hybrid Integer

The performance of the hybrid integer representation is much improved in heterozygous form (Figure 7-8) with 17% of solutions identifying the single-objective optimal solution compared to 3% of the normally encoded runs.

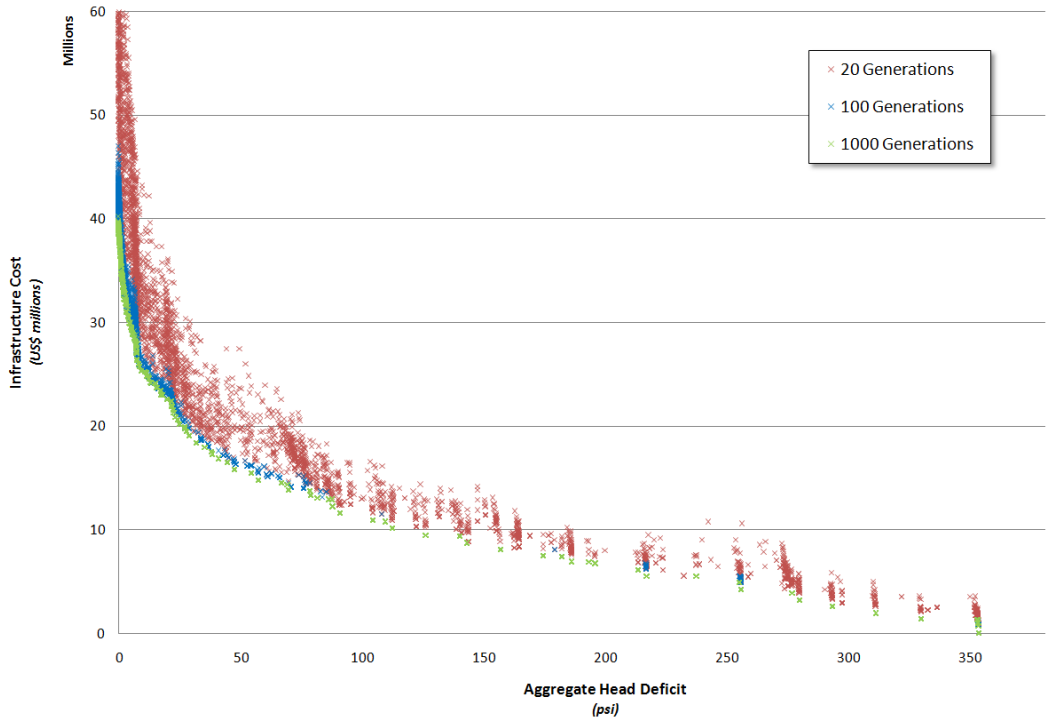

Figure 7-8: New York Tunnels – Multiple Objective Heterozygous Hybrid Integer Results

7.2.2.4. Comparative Analysis

Graphical

Figure 7-9 allows a side-by-side graphical comparison of the algorithm performance for the three representations and two encodings. The less confined distributions, after 20 generations, exhibited by the heterozygous binary string and heterozygous hybrid integer representations are apparent in this figure.

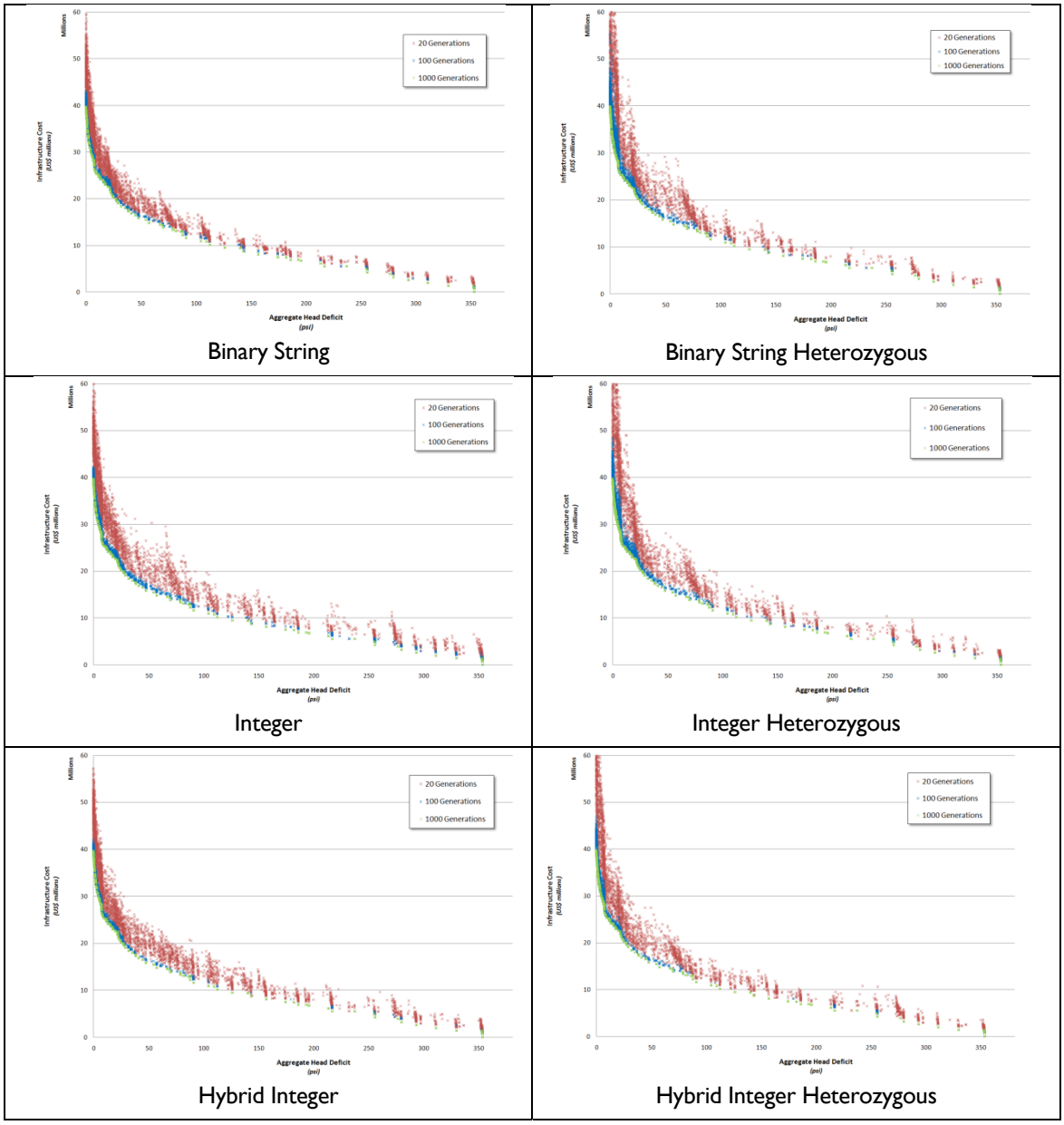

Figure 7-9: Graphical Comparison of Multiple Objective New York Tunnels results

Space Metrics

The *S* metric results for the heterozygous results are considered in Figure 7-10. These results show better performance for the hybrid integer representation compared to the other two – with the integer representation again performing the worst of all three.

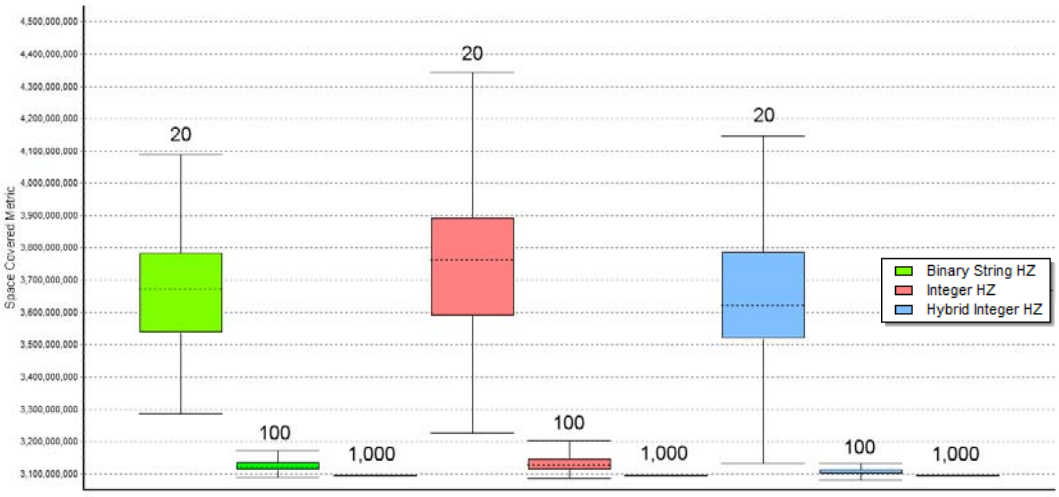

Figure 7-10: Box plots of *S* metric for Multiple Objective Heterozygous New York Tunnels after 20, 100 & 1,000 generations

This analysis is extended in Figure 7-11 where the performance of the normal and heterozygous encodings of the problem can be compared. As can be seen, the heterozygous versions generally outperform their normally encoded counterparts, with the exception of the standard binary string representation, which performs best of all – a possible indication that the multiple-objective algorithm favours the greater stochasticity introduced by this format.

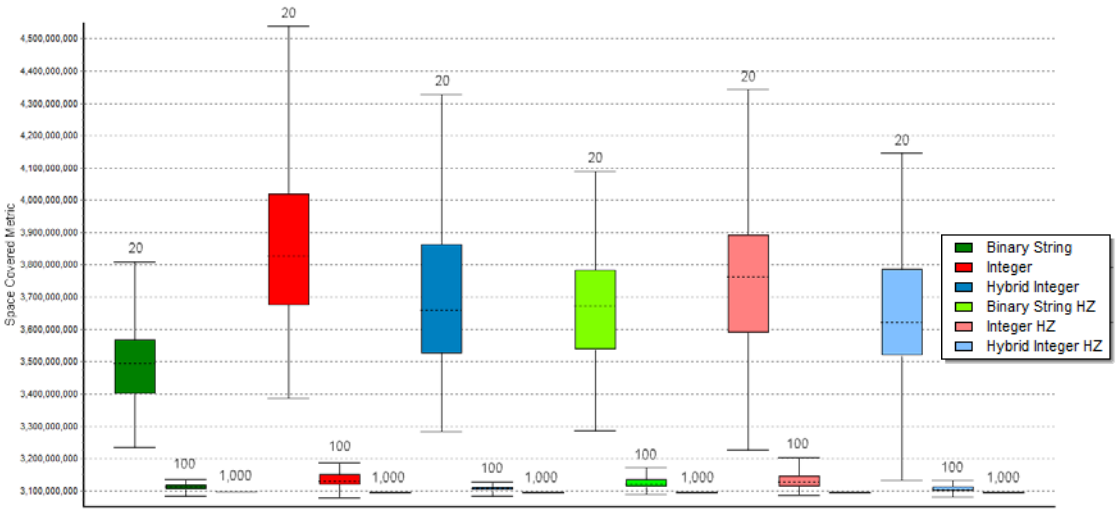

Figure 7-11: Box plots of *S* metric for Multiple Objective New York Tunnels for Normal and Heterozygous New York Tunnels after 20, 100 & 1,000 generations

Coverage Metrics

The coverage metrics for the above runs are reported in Table 7-5 - Table 7-7 which relate to the coverage performance of the algorithm after 20, 100 and 1,000 generations respectively. These tables allow a direct comparison to be made between the performance of the normally

and heterozygous-encoded algorithms for each of the three representations. Table 7-5 clearly shows the strong performance of the normal binary string representation followed by that of the normal hybrid integer representation.

				Front B					
				Conventional			Heterozygous		
				Binary String	Integer	Hybrid Integer	Binary String	Integer	Hybrid Integer
Front A	Normal	Binary String	Min	0.138	0.388	0.311	0.453	0.543	0.395
			Mean	**0.387**	**0.653**	**0.593**	**0.642**	**0.740**	**0.680**
			Max	0.652	0.980	0.943	0.852	0.933	0.892
		Integer	Min	0.000	0.041	0.000	0.141	0.130	0.047
			Mean	**0.234**	**0.453**	**0.328**	**0.400**	**0.482**	**0.398**
			Max	0.550	0.900	0.736	0.679	0.911	0.730
		Hybrid Integer	Min	0.000	0.171	0.022	0.156	0.217	0.140
			Mean	**0.263**	**0.551**	**0.438**	**0.505**	**0.599**	**0.521**
			Max	0.485	0.940	0.792	0.736	0.911	0.784
	Heterozygous	Binary String	Min	0.030	0.171	0.022	0.172	0.217	0.140
			Mean	**0.186**	**0.538**	**0.419**	**0.419**	**0.574**	**0.499**
			Max	0.348	0.940	0.792	0.642	0.911	0.784
		Integer	Min	0.000	0.049	0.022	0.047	0.109	0.047
			Mean	**0.123**	**0.366**	**0.247**	**0.297**	**0.399**	**0.342**
			Max	0.283	0.840	0.736	0.698	0.867	0.768
		Hybrid Integer	Min	0.000	0.131	0.065	0.094	0.130	0.116
			Mean	**0.168**	**0.420**	**0.307**	**0.387**	**0.487**	**0.423**
			Max	0.417	0.780	0.736	0.679	0.867	0.784

Table 7-5: *C* metrics for Multiple Objective Heterozygous New York Tunnels after 20 generations

The results presented in Table 7-6 after 100 generations demonstrate the continued (relative) dominance of the normally encoded binary string and hybrid integer over the other representations – albeit to a lesser degree than that apparent after 20 generations. The heterozygous representations show poor performance relative to their normally encoded counterparts – as evidenced by the low values presented in the lower-left quadrant of Table 7-6.

				Front B					
				Conventional			Heterozygous		
				Binary String	Integer	Hybrid Integer	Binary String	Integer	Hybrid Integer
Front A	Normal	Binary String	Min	0.073	0.120	0.020	0.144	0.151	0.099
			Mean	**0.215**	**0.258**	**0.167**	**0.338**	**0.311**	**0.298**
			Max	0.375	0.343	0.289	0.453	0.489	0.424
		Integer	Min	0.096	0.163	0.020	0.222	0.154	0.176
			Mean	**0.215**	**0.260**	**0.159**	**0.328**	**0.303**	**0.302**
			Max	0.354	0.364	0.330	0.453	0.468	0.414
		Hybrid Integer	Min	0.110	0.163	0.060	0.247	0.154	0.198
			Mean	**0.249**	**0.282**	**0.187**	**0.352**	**0.334**	**0.319**
			Max	0.385	0.414	0.320	0.453	0.500	0.414
	Heterozygous	Binary String	Min	0.031	0.074	0.020	0.041	0.080	0.077
			Mean	**0.106**	**0.260**	**0.168**	**0.222**	**0.309**	**0.293**
			Max	0.260	0.414	0.320	0.421	0.500	0.414
		Integer	Min	0.010	0.050	0.000	0.093	0.011	0.022
			Mean	**0.120**	**0.153**	**0.068**	**0.256**	**0.214**	**0.196**
			Max	0.271	0.237	0.186	0.442	0.405	0.354
		Hybrid Integer	Min	0.010	0.110	0.010	0.072	0.090	0.033
			Mean	**0.114**	**0.170**	**0.076**	**0.271**	**0.234**	**0.194**
			Max	0.229	0.242	0.247	0.442	0.457	0.343

Table 7-6: *C* metrics for Multiple Objective Heterozygous New York Tunnels after 100 generations

Once more, the very small coverage ratios related in Table 7-7 indicate that all of the six combinations of representation analysed converge to almost identical Pareto fronts with few significant differences between them.

				Front B					
				Conventional			Heterozygous		
				Binary String	Integer	Hybrid Integer	Binary String	Integer	Hybrid Integer
Front A	Normal	Binary String	Min	0.000	0.000	0.000	0.000	0.000	0.000
			Mean	**0.018**	**0.016**	**0.022**	**0.031**	**0.053**	**0.030**
			Max	0.070	0.050	0.050	0.070	0.100	0.070
		Integer	Min	0.010	0.000	0.020	0.010	0.030	0.020
			Mean	**0.056**	**0.031**	**0.054**	**0.063**	**0.075**	**0.056**
			Max	0.110	0.090	0.100	0.110	0.110	0.090
		Hybrid Integer	Min	0.000	0.000	0.000	0.000	0.000	0.000
			Mean	**0.031**	**0.024**	**0.036**	**0.043**	**0.062**	**0.039**
			Max	0.110	0.080	0.090	0.090	0.120	0.080
	Heterozygous	Binary String	Min	0.000	0.000	0.000	0.000	0.000	0.000
			Mean	**0.024**	**0.029**	**0.045**	**0.036**	**0.068**	**0.045**
			Max	0.120	0.090	0.100	0.100	0.120	0.090
		Integer	Min	0.000	0.000	0.000	0.000	0.000	0.000
			Mean	**0.022**	**0.014**	**0.026**	**0.030**	**0.047**	**0.030**
			Max	0.080	0.060	0.070	0.070	0.100	0.070
		Hybrid Integer	Min	0.000	0.000	0.000	0.000	0.010	0.000
			Mean	**0.033**	**0.027**	**0.035**	**0.042**	**0.066**	**0.039**
			Max	0.080	0.070	0.070	0.080	0.110	0.070

Table 7-7: *C* metrics for Multiple Objective Heterozygous New York Tunnels after 1,000 generations

7.2.2.5. Runtime Performance

As can be seen from Table 7-8, for the New York Tunnels problem in which the chromosome length is very similar between the conventional and heterozygous encodings, the additional overhead of interpreting the heterozygous chromosomes leads to a deleterious effect on performance for this representation.

Chromosome Representation	Conventional		Heterozygous	
	(evaluations per second)	% of best	(evaluations per second)	% of best
Binary String	1,868.98	86.3%	1,574.30	72.7%
Integer	2,164.93	100%	1,801.72	83.2%
Hybrid Integer	1,886.79	87.2%	1,630.70	75.3%

Table 7-8: New York Tunnels Multiple Objective Heterozygous Runtime Performance vs. Conventional Performance

7.2.3. Caching

The caching results for the multiple objective New York Tunnels problem are presented in Table 7-9. As with the single objective formulation of the problem, despite the cache being

used, the runtimes are increased when the caching is enabled – for either type of cache. This is the result of the search performance of the caches taking, on average, a longer interval than the hydraulic simulation of the network. The ten-fold increase in the proportion of cache hits compared to the maximum of 3% for the single objective problem supports the concept of the caching methodology.

Cache Size	Runtime performance	Cache Performance		Relative Runtime Performance
	(solutions/second)	*(no. hits)*	*(% of evaluations)*	
None	1,886.79	n/a	n/a	100%
40,000	1,355.01	28,290.3	28.3%	139.2%
Judy (unlimited)	1,292.55	34,538.9	34.5%	146.0%

Table 7-9: Cache results: Multiple Objective New York Tunnels

7.3. Hanoi

7.3.1. Genetic Representation

7.3.1.1. Binary String

A considerable overlap can be seen in Figure 7-12, which shows the binary string results, between the 100 generation and 1,000 generation distributions suggesting that this representation converges rapidly toward the Pareto-optimal front. None of the 100 runs of this optimization configuration resulted in the best know solution of $6.081m being identified.

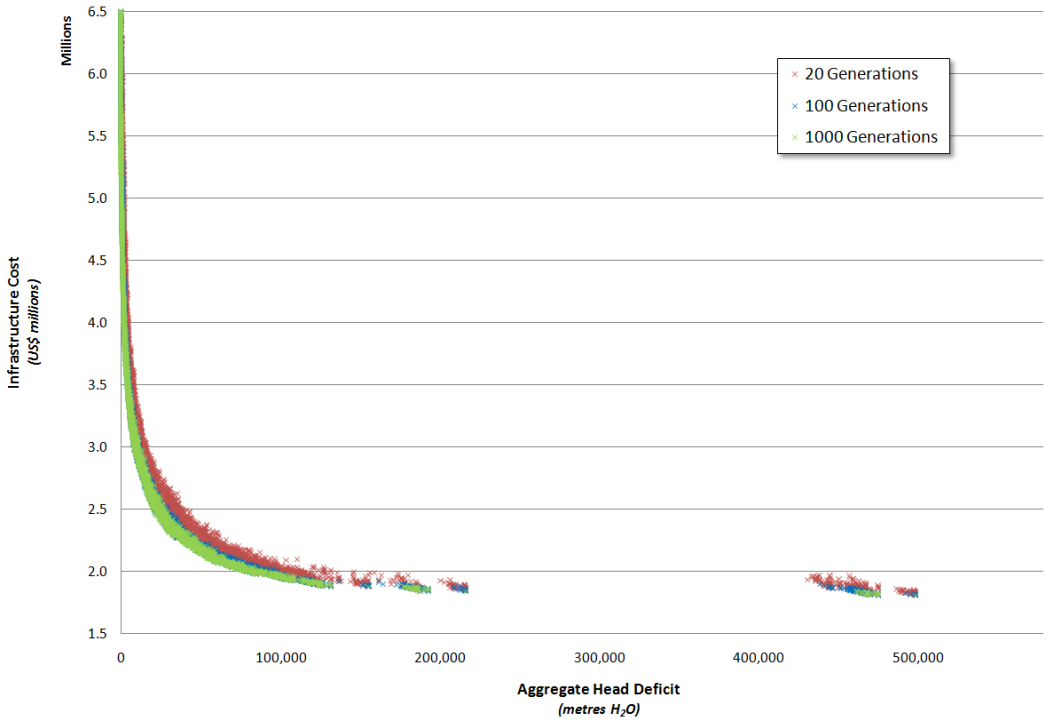

Figure 7-12: Hanoi – Multiple Objective Binary String Results

7.3.1.2. Integer

The results for the integer representation, presented in Figure 7-13, show a greater diversity of results than those seen with the binary string representation. The overlap between the 100 generation and 1,000 generation distributions, whilst present, is diminished. In common with the binary string representation, none of the runs identified the best-known solution for this problem.

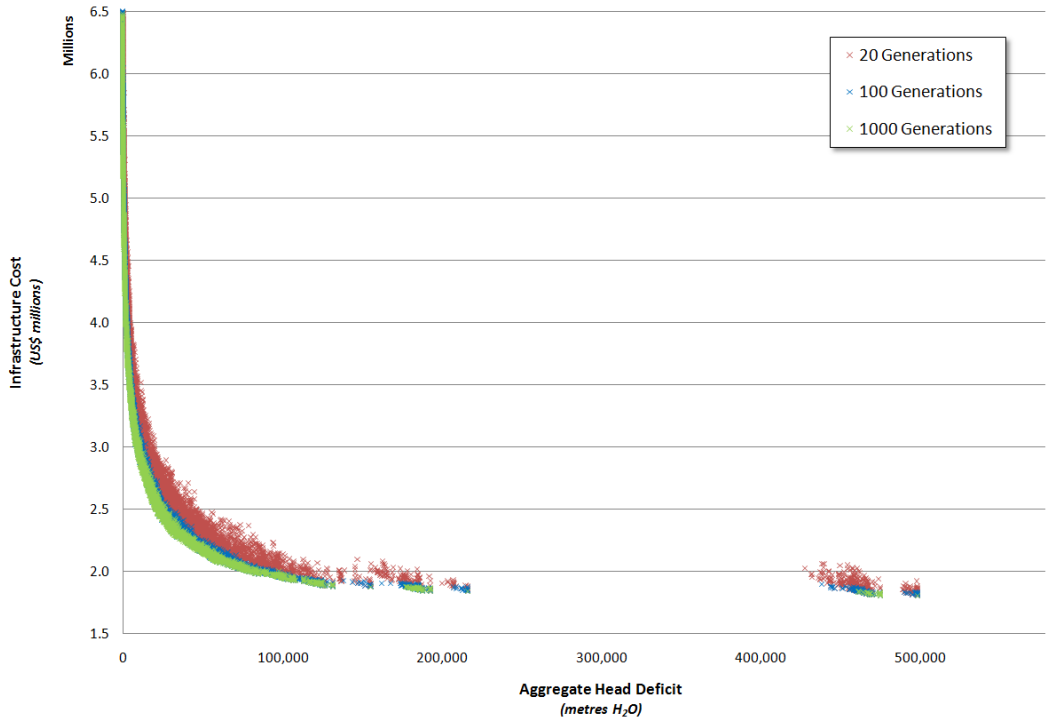

Figure 7-13: Hanoi – Multiple Objective Integer Results

7.3.1.3. Hybrid Integer

Figure 7-14 illustrates the results of the hybrid integer representation – which appear very similar to those of the integer. Again, no runs of this optimization identified the optimum.

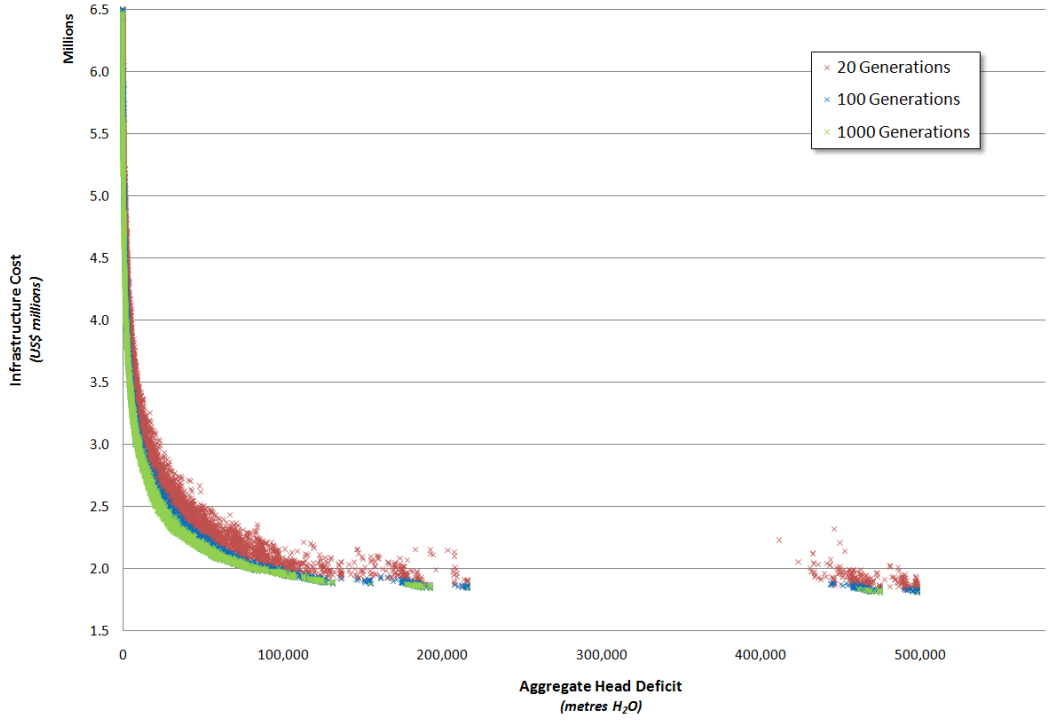

Figure 7-14: Hanoi – Multiple Objective Hybrid Integer Results

7.3.1.4. Comparative Analysis

Graphical

Figure 7-15 allows a direct graphical comparison of the results presented above:

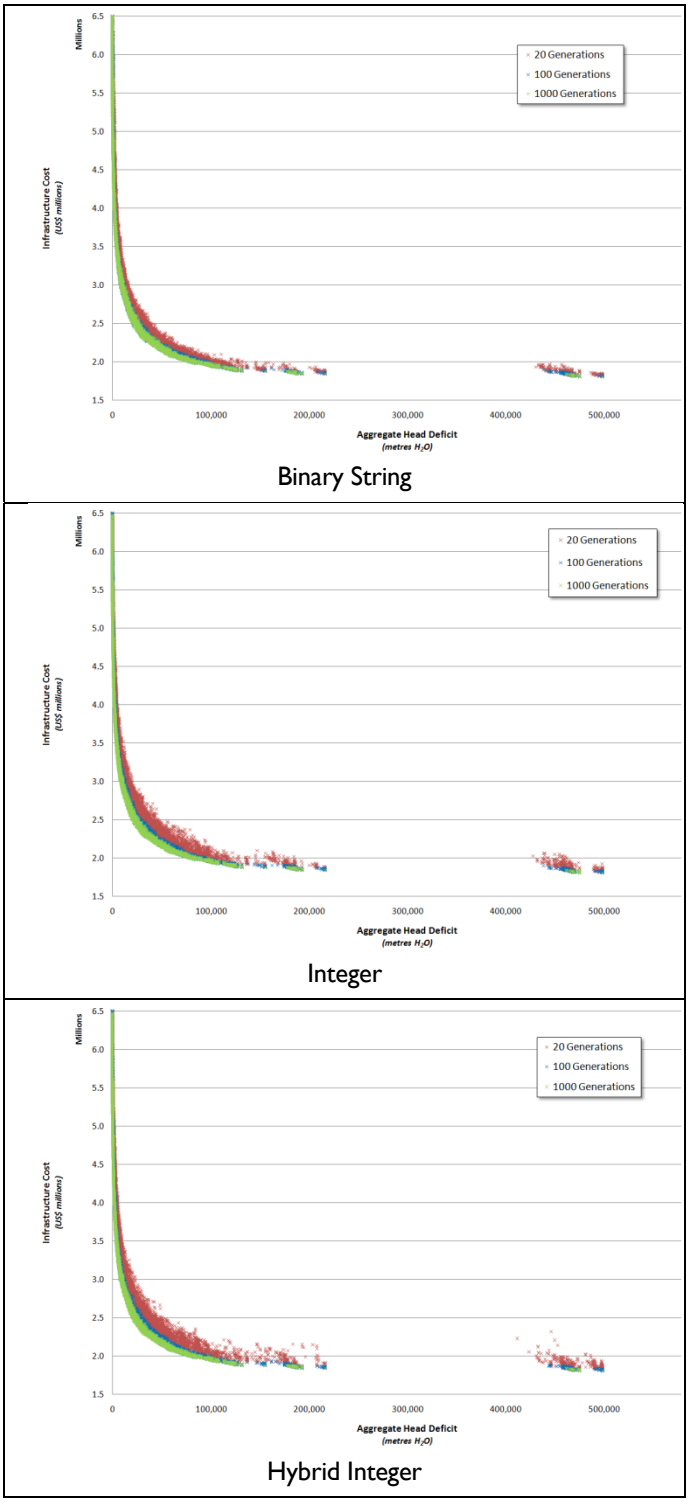

Figure 7-15: Graphical Comparison of Multiple Objective Hanoi results

The less well-defined results of the integer and hybrid integer representations, particularly after 20 generations, are visible in this figure.

Space Metrics

The *S* metric for the Hanoi optimization is related in the box plots in Figure 7-16. These show the superior convergence of the binary string in the earlier stages of the optimization but suggest that the integer, and particularly the hybrid integer, runs converge better.

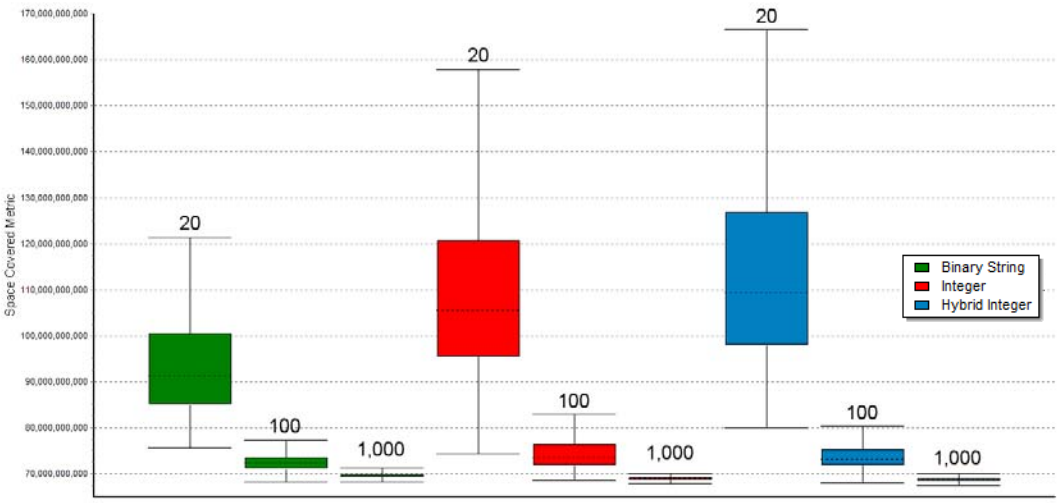

Figure 7-16: Box plots of *S* metric for Multiple Objective Hanoi after 20, 100 and 1,000 generations

Coverage Metrics

The coverage metrics for the Hanoi optimization are reported in Table 7-10 - Table 7-12 for 20, 100 and 1,000 generations respectively. Table 7-10 shows that the binary string representation again dominates that of the other two representations – whilst the integer representation outperforms the hybrid integer at this stage.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.062	0.197	0.232
		Mean	**0.421**	**0.438**	**0.494**
		Max	0.899	0.740	0.831
	Integer	Min	0.050	0.110	0.250
		Mean	**0.323**	**0.415**	**0.514**
		Max	0.617	0.720	0.854
	Hybrid Integer	Min	0.040	0.062	0.105
		Mean	**0.244**	**0.343**	**0.432**
		Max	0.535	0.582	0.740

Table 7-10: *C* metrics for Multiple Objective Hanoi after 20 generations

After 100 generations very little, relative, difference can be discerned in the performance of the representations – as seen in Table 7-11 – where the mean figures for the variation in coverage are almost equal for each combination.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.080	0.140	0.100
		Mean	**0.328**	**0.316**	**0.323**
		Max	0.620	0.530	0.590
	Integer	Min	0.090	0.160	0.100
		Mean	**0.347**	**0.359**	**0.349**
		Max	0.600	0.600	0.590
	Hybrid Integer	Min	0.100	0.120	0.080
		Mean	**0.352**	**0.368**	**0.355**
		Max	0.630	0.660	0.660

Table 7-11: *C* metrics for Multiple Objective Hanoi after 100 generations

The relative coverage of the fronts at the conclusion of the optimizations is shown in Table 7-12. This shows that at this stage, the integer and hybrid integer representations are performing better than the binary string and that the hybrid integer is marginally outperforming the integer – a fact reflected in the box plots in Figure 7-16.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.090	0.050	0.040
		Mean	**0.297**	**0.154**	**0.154**
		Max	0.540	0.360	0.400
	Integer	Min	0.210	0.150	0.120
		Mean	**0.444**	**0.264**	**0.261**
		Max	0.610	0.420	0.490
	Hybrid Integer	Min	0.130	0.120	0.080
		Mean	**0.444**	**0.271**	**0.264**
		Max	0.610	0.430	0.490

Table 7-12: *C* metrics for Multiple Objective Hanoi after 1,000 generations

7.3.1.5. Runtime performance

The runtime result presented in Table 7-13 demonstrates that, relative to the binary string representation, the hybrid integer is considerably better performing on the length of chromosome employed by this problem. The pure integer representation remains the quickest of all.

Chromosome Representation	Average Performance *(evaluations per second)*	% of best performance
Binary String	1,579.28	84.8%
Integer	1,862.80	100%
Hybrid Integer	1,747.57	93.8%

Table 7-13: Hanoi Multiple Objective Runtime Performance

7.3.2. Caching

With the caching enabled, the throughput of the algorithm using the hybrid integer representation exceeds that of the uncached integer (Table 7-13). As with the New York Tunnels problem, these results represent a marked improvement over those of the single objective version of the problem. However, given the greater computational demand of the Hanoi problem for the hydraulic solver, the advantages of the cache utilisation are clearly demonstrated with savings in runtime of between 6.5 and 12%.

Cache Size	Runtime performance *(solutions/second)*	Cache Performance *(no. hits)*	*(% of evaluations)*	Relative Runtime Performance
None	1,747.57	n/a	n/a	100%
40,000	1,868.46	5,916.3	5.9%	93.5%
Judy (unlimited)	1,980.47	7,972.1	8.0%	88.2%

Table 7-14: Cache results: Multiple Objective Hanoi

7.4. Piedemonte San Germano

7.4.1. Genetic Representation

7.4.1.1. Binary String

A scatter plot illustrating the results for the binary string representation after 20, 100 and 1,000 generations can be seen in Figure 7-17. For this representation, no solutions for any of the runs identified the optimal solution determined in the single-objective analysis of €30,082.10. A large discontinuity is visible in the aggregate head deficit objective. This discontinuity is apparent owing to the fact that the feed from the reservoir (pipe 101 in Figure 6-26) is critical to system performance. In the absence of this reinforcement, the head deficits throughout the system are high c.30,000m and higher; whilst reinforcement, even of the smallest diameter option, results in head deficits of c.12,000m and lower.

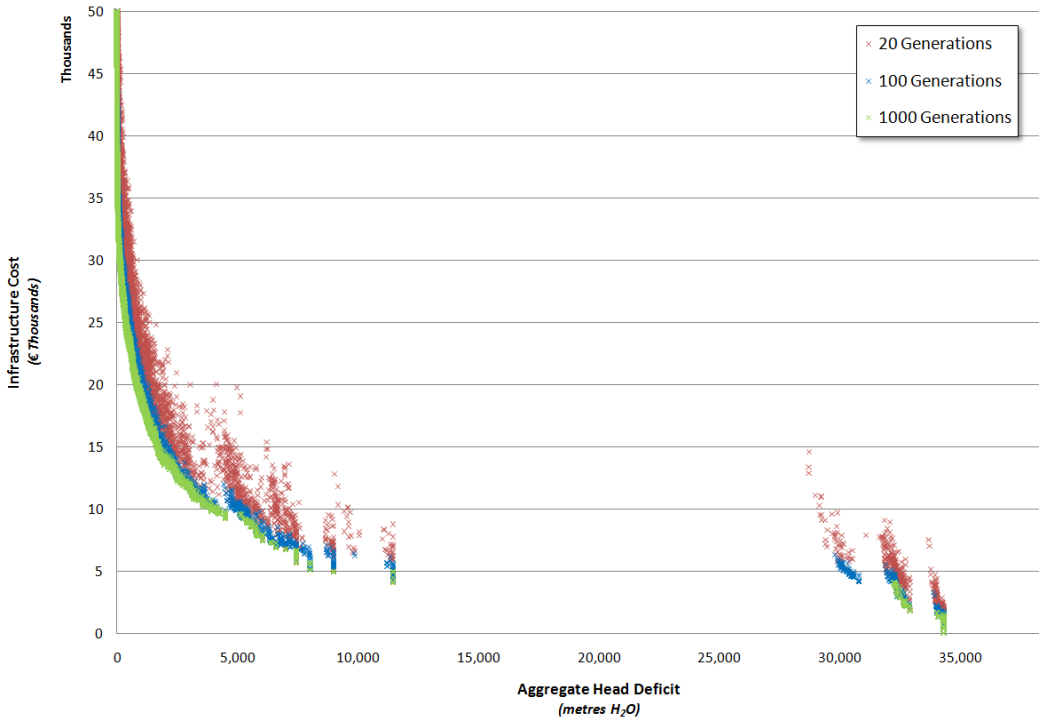

Figure 7-17: Piedemonte San Germano– Multiple Objective Binary String Results

7.4.1.2. Integer

Figure 7-18 relates the results for the integer representation when applied to the multiple objective PSG problem. These results can be seen to cover a wider distribution than those of the binary string. As with the latter representation, none of the runs identified the best-known solution. It can be seen from these results that a large number of results are identified with near-zero head deficits and cost varying between c.€60,000 and €30,000. This wide variation occurs as the PSG model has a number of very short pipes that are included in the optimization and yet have an insignificant effect on the performance of the hydraulic model, which is characterised by extremely low flows in some locations. Thus, from the point of view of the optimization, the reinforcement of these pipes contributes significantly to the capital cost objective but has little bearing on that of the head deficit. Ideally, these pipes should be identified *a priori* and omitted from the optimization process.

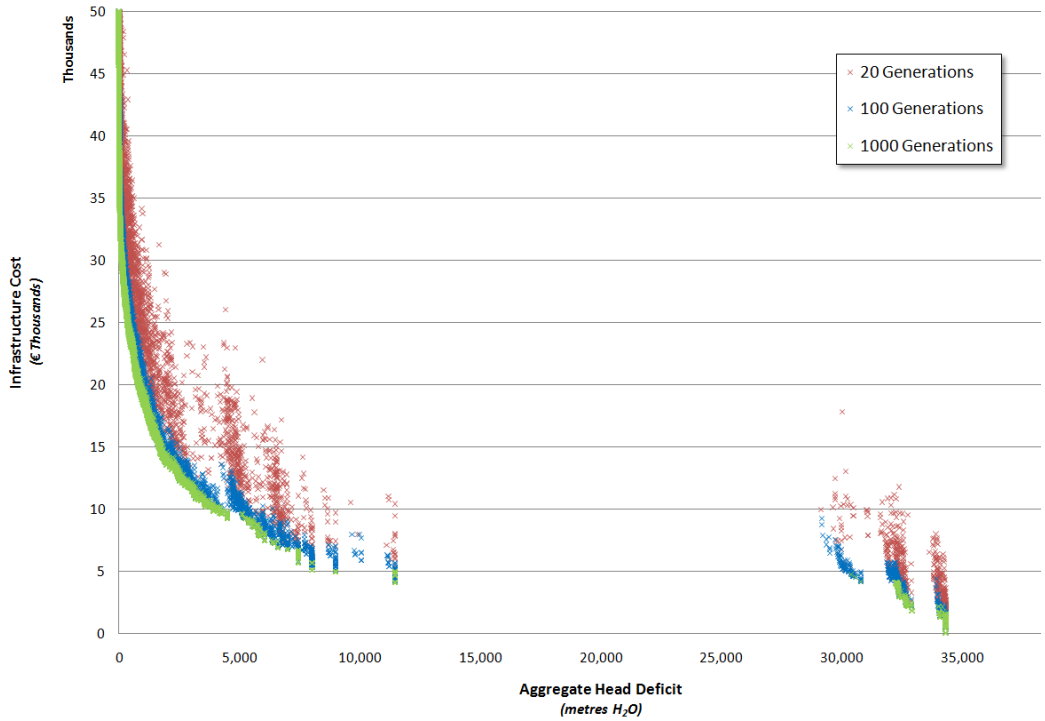

Figure 7-18: Piedemonte San Germano– Multiple Objective Integer Results

7.4.1.3. Hybrid Integer

The results for the Hybrid Integer representation are shown in Figure 7-19. Again, after 1,000 generations, none of the runs had identified the optimal solution of €30,082.10.

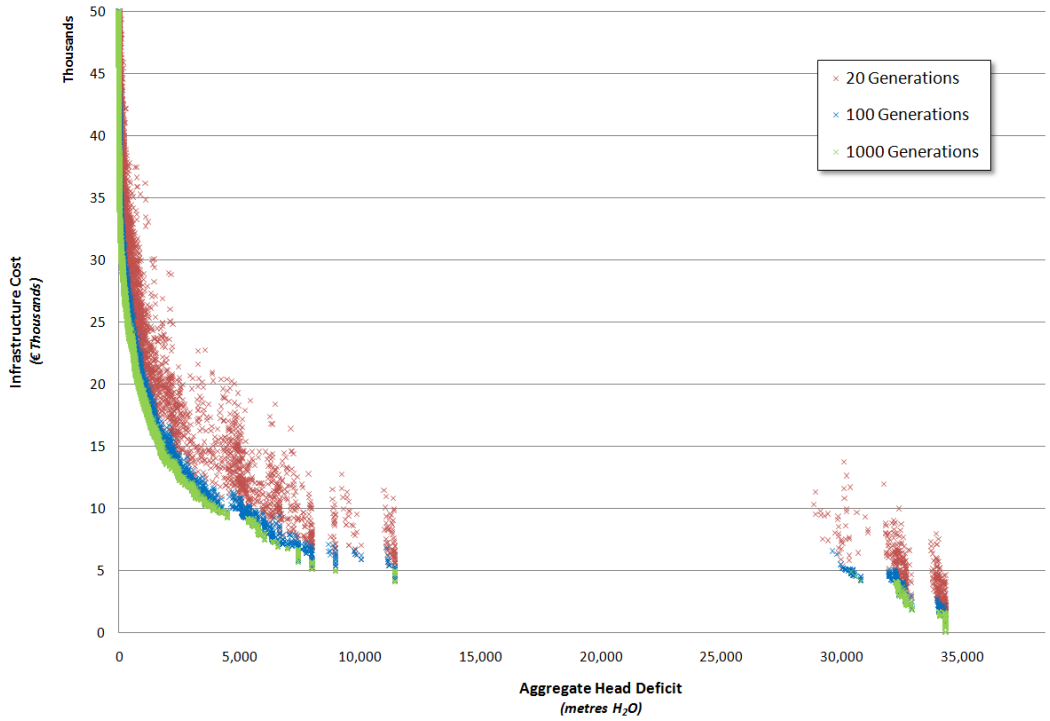

Figure 7-19: Piedemonte San Germano– Multiple Objective Hybrid Integer Results

7.4.1.4. Comparative Analysis

Space Metrics

Examination of the *S* space metrics (Figure 7-20) for these runs demonstrate that, in common with the New York Tunnels and Hanoi problems, the binary string representation outperforms both the integer and hybrid integer versions in terms of convergence towards an optimal solution. The performance of the hybrid integer representation appears superior to that of the integer representation throughout the lifetime of the optimization. However, the box plots suggest that, ultimately, the hybrid integer and integer representations have generally converged to superior Pareto-optimal fronts – although analysis of the *C* coverage metric is necessary to confirm this.

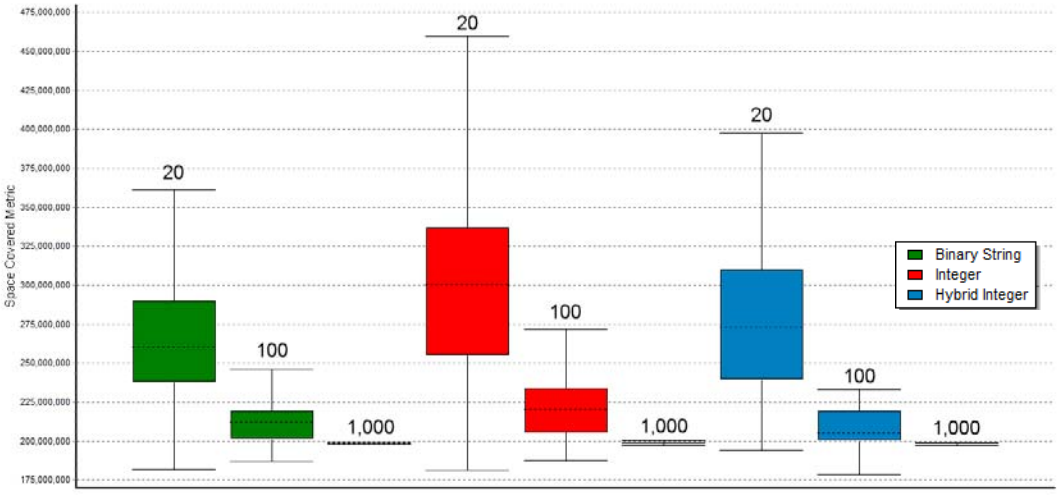

Figure 7-20: Box plots of *S* metric for Multiple Objective Piedemonte San Germano after 20, 100 and 1,000 generations

Coverage Metrics

The coverage metrics for the runs were calculated for the Pareto-optimal fronts obtained after 20, 100 and 1,000 generations of each of the 100 runs undertaken for each representation. The aggregate values obtained for the metric *C(A,B)* are listed in Table 7-15 to Table 7-17 for 20, 100 and 10,000 generations respectively. The Binary String representation can again be seen in Table 7-15 to be outperforming that of the integer and hybrid integer – covering 58.1% and 51.6% of the points of fronts respectively.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.222	0.241	0.193
		Mean	**0.415**	**0.581**	**0.516**
		Max	0.672	0.974	0.868
	Integer	Min	0.000	0.000	0.000
		Mean	**0.300**	**0.439**	**0.371**
		Max	0.567	0.949	0.755
	Hybrid Integer	Min	0.014	0.130	0.053
		Mean	**0.331**	**0.483**	**0.424**
		Max	0.697	0.949	0.906

Table 7-15: *C* metrics for Piedemonte San Germano after 20 generations

The same *C* metric after 100 generations (Table 7-16) shows that the binary string representation continues to perform well, although the hybrid integer, on average, can now be seen to cover a greater proportion of points than those of the other two representations.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.200	0.320	0.160
		Mean	**0.363**	**0.484**	**0.336**
		Max	0.500	0.707	0.590
	Integer	Min	0.110	0.090	0.050
		Mean	**0.260**	**0.388**	**0.254**
		Max	0.460	0.636	0.470
	Hybrid Integer	Min	0.140	0.190	0.090
		Mean	**0.410**	**0.513**	**0.365**
		Max	0.630	0.747	0.630

Table 7-16: *C* metrics for Piedemonte San Germano after 100 generations

At the completion of the optimizations, the *C* metrics (Table 7-17) show very similar levels of average coverage although the hybrid integer representation can be seen to outperform the other representations, covering a greater proportion of their Pareto-optimal solutions.

			Front B		
			Binary String	*Integer*	*Hybrid Integer*
Front A	*Binary String*	Min	0.090	0.090	0.100
		Mean	**0.204**	**0.256**	**0.197**
		Max	0.300	0.410	0.310
	Integer	Min	0.070	0.080	0.070
		Mean	**0.191**	**0.230**	**0.180**
		Max	0.330	0.420	0.320
	Hybrid Integer	Min	0.100	0.090	0.090
		Mean	**0.247**	**0.292**	**0.214**
		Max	0.350	0.450	0.360

Table 7-17: *C* metrics for Piedemonte San Germano after 1000 generations

7.4.1.5. Runtime Performance

The runtime performance comparison for these representations is presented in Table 7-18. In common with the other network problems analysed, the integer representation performs best with a marginal difference apparent between the binary string and hybrid integer results.

Chromosome Representation	Average Performance (evaluations per second)	% of best performance
Binary String	3,837.22	90.8%
Integer	4,223.89	100%
Hybrid Integer	3,766.36	89.0%

Table 7-18: Piedemonte San Germano Multiple Objective Runtime Performance

7.4.2. Heterozygous Chromosomes

7.4.2.1. Binary String

The heterozygous binary string results, presented in Figure 7-21, are broadly comparable with those of the normal representation with the exception that, visually, they exhibit a narrower distribution of results for the 20 and 100 generation points – the opposite of the equivalent results obtained for the smaller New York Tunnels problem, suggesting that the heterozygous arrangement performs better for convergence for the larger problem. As with the normally-encoded version of this problem, none of the binary string runs was able to identify the optimal solution determined in the single-objective optimization.

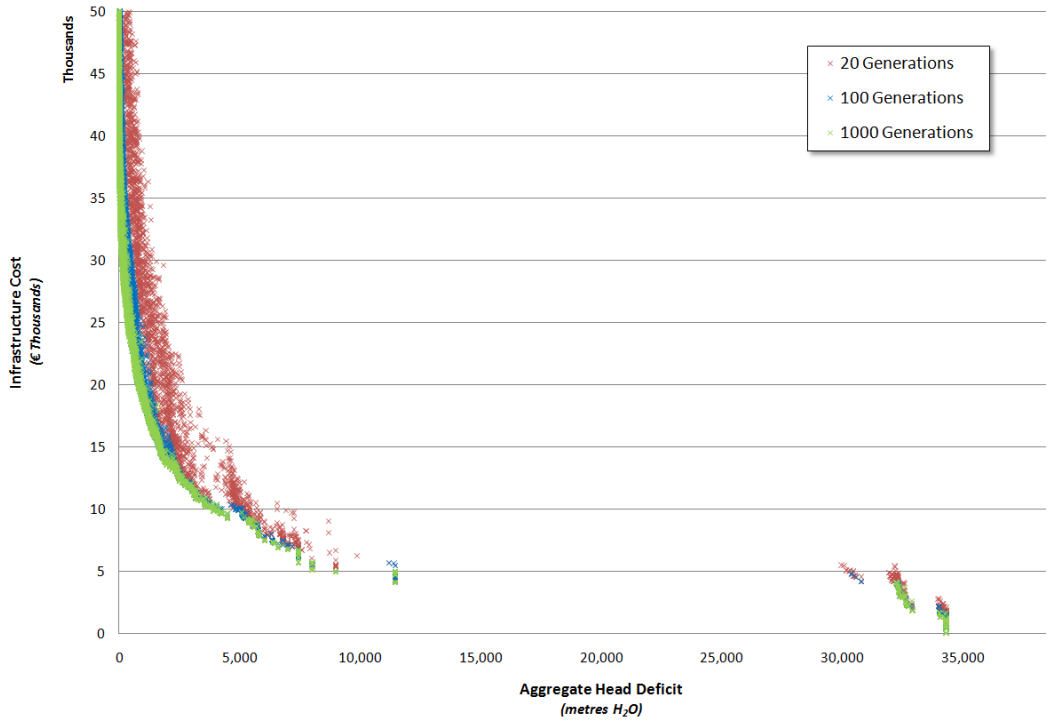

Figure 7-21: Piedemonte San Germano– Multiple Objective Heterozygous Binary String Results

7.4.2.2. Integer

Figure 7-22 illustrates the heterozygous integer results, which appear to be very similar to those obtained with the standard representation. In common with the binary string results, above, the heterozygous version of the integer appears to show better convergence behaviour. Again, none of the runs for this representation managed to identify the best known-solution for the single objective problem formulation.

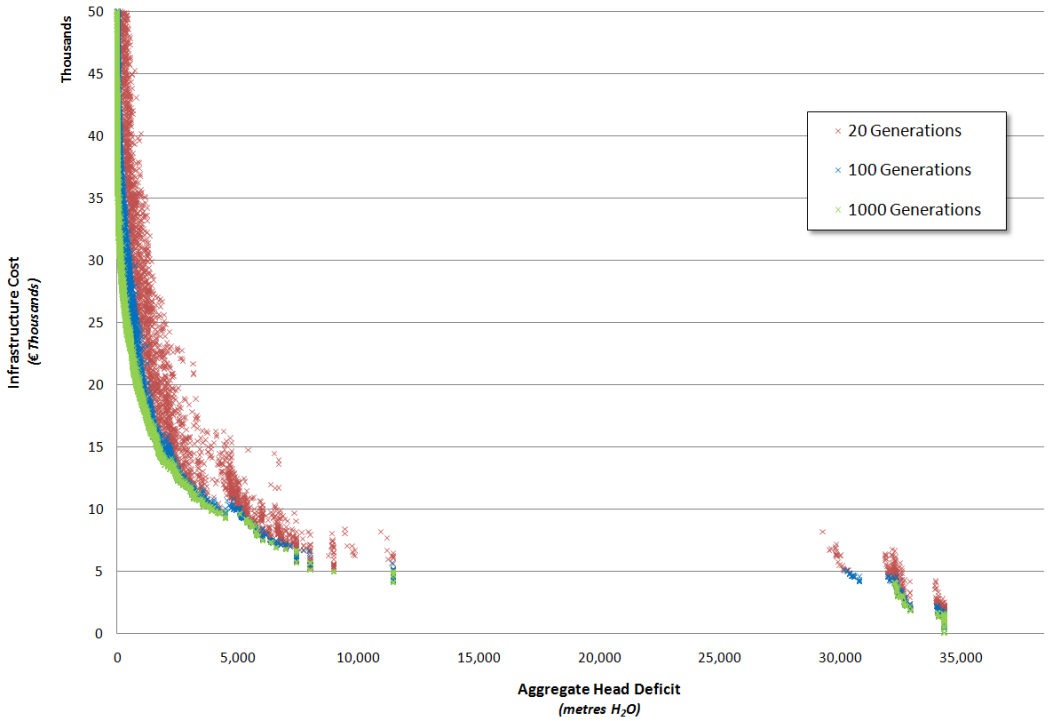

Figure 7-22: Piedemonte San Germano– Multiple Objective Heterozygous Integer Results

7.4.2.3. Hybrid Integer

Figure 7-23 shows the results for the heterozygous hybrid integer representation of the problem showing, once more, more rapid convergence than the prior encoding.

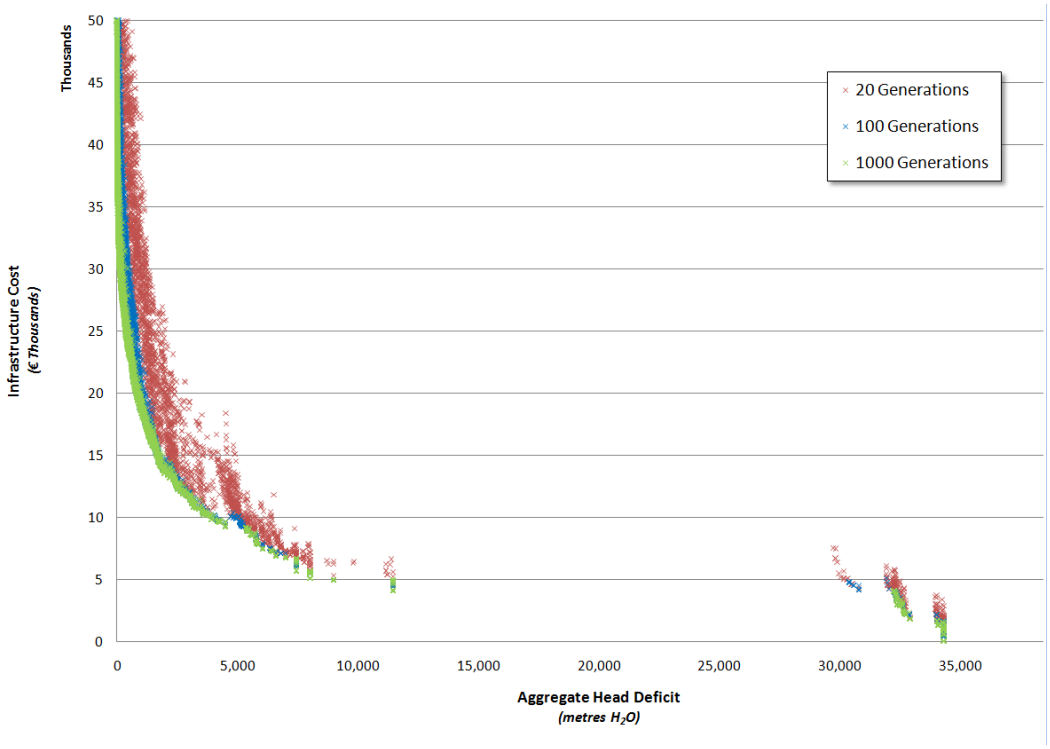

Figure 7-23: Piedemonte San Germano– Multiple Objective Heterozygous Hybrid Integer Results

7.4.2.4. Comparative Analysis

Graphical

Figure 7-24 permits a direct graphical comparison of the algorithm performance for the three representations and two encodings. The more constrained distributions, after 20 and 100 generations, exhibited by the heterozygous binary string and heterozygous hybrid integer (and to a lesser extent the heterozygous integer) representations are apparent in this figure – particularly with reference to the solutions to the right of the head deficit discontinuity.

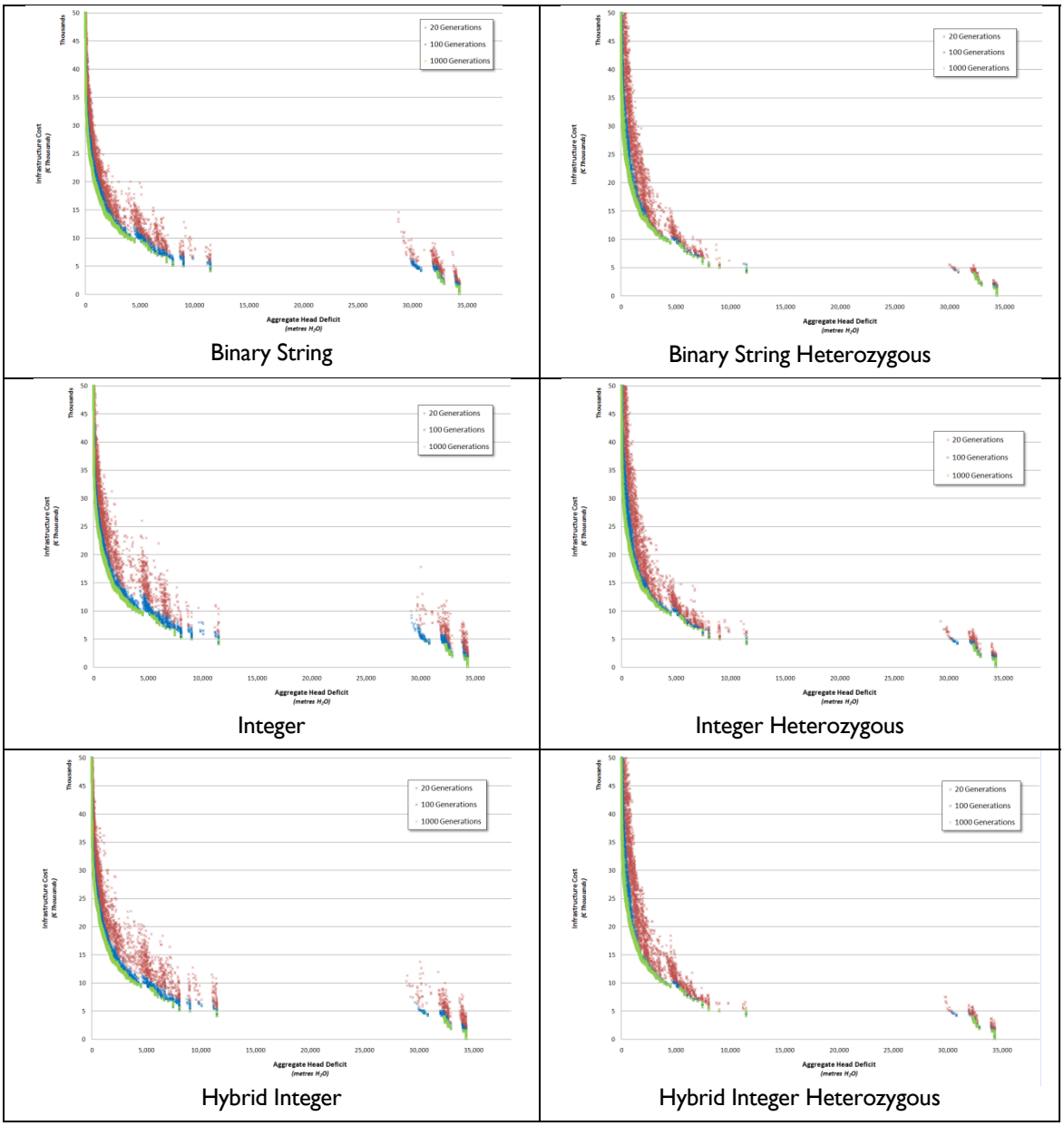

Figure 7-24: Graphical Comparison of Multiple Objective Piedemonte San Germano results

Space Metrics

The *S* metric results for the heterozygous results are presented in Figure 7-25. These results illustrate the better convergence of the binary string representation compared to the other two – with the integer representation again performing the worst of all three. However, as with the normally encoded representation, the superior performance of the hybrid integer representation towards the end of the optimization is demonstrated.

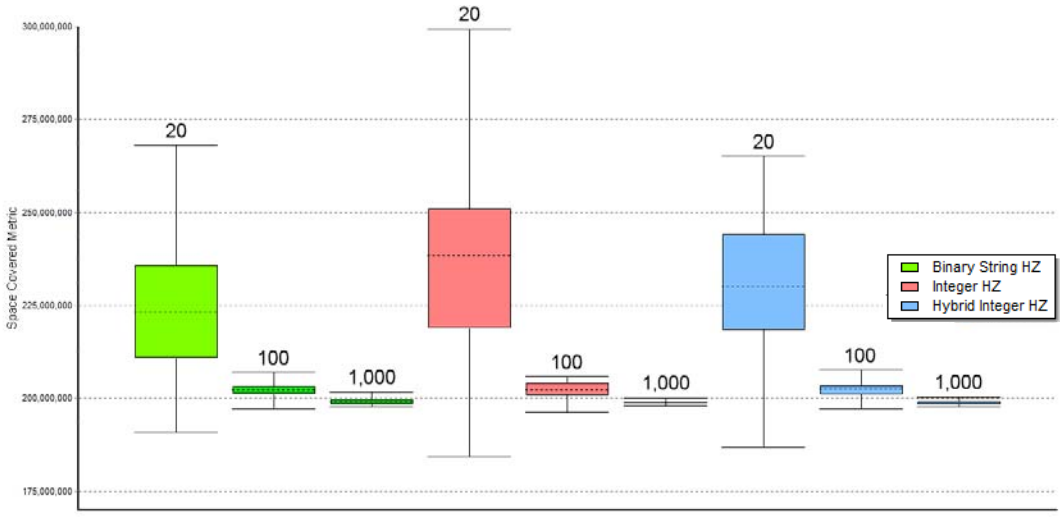

Figure 7-25: Box plots of *S* metric for Multiple Objective Heterozygous Piedemonte San Germano after 20, 100 & 1,000 generations

A comparison between the *S* metrics of both the normally and heterozygous encoded representations is shown in Figure 7-26. In contrast to the results obtained for the simpler New York Tunnels problem (Figure 7-11) the improved performance of the heterozygous encoding relative to the normal encoding, in terms of rapidity of convergence, is clearly seen.

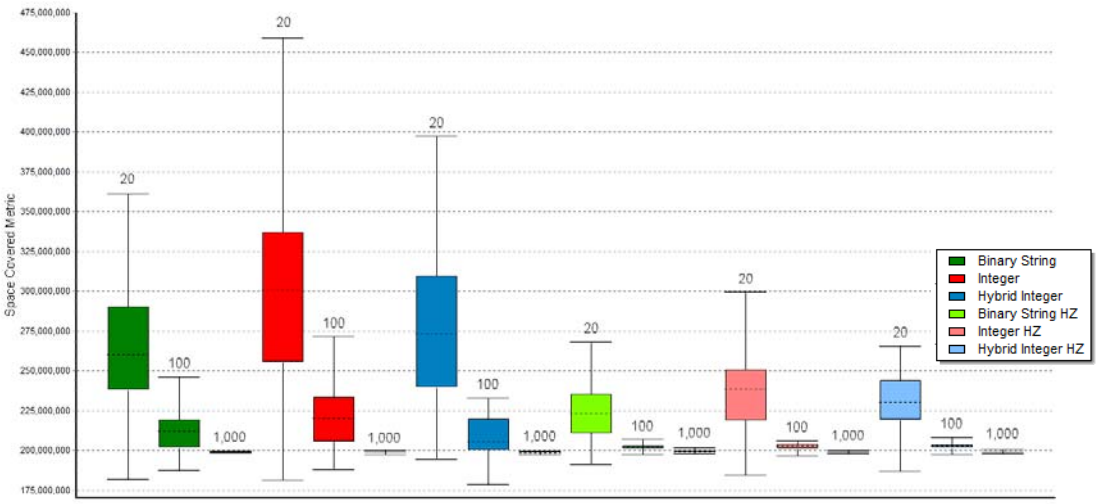

Figure 7-26: Box plots of *S* metric for Multiple Objective Normal and Heterozygous Piedemonte San Germano after 20, 100 & 1,000 generations

Coverage Metrics

In the early stages of the optimization (20 generations) the *C* coverage metrics (Table 7-19) for the normally-encoded binary string show that it is the most effective of the combinations in terms of relative quality of the Pareto fronts obtained. This is at variance with what might be expected from the *S* space metric above (Figure 7-26) in which all of the heterozygous representations appear to outperform the normally encoded binary string.

				Front B					
				Conventional			Heterozygous		
				Binary String	Integer	Hybrid Integer	Binary String	Integer	Hybrid Integer
Front A	Normal	*Binary String*	Min	0.222	0.241	0.193	0.102	0.192	0.109
			Mean	**0.415**	**0.581**	**0.516**	**0.360**	**0.386**	**0.381**
			Max	0.672	0.974	0.868	0.607	0.542	0.732
		Integer	Min	0.000	0.000	0.000	0.082	0.071	0.065
			Mean	**0.300**	**0.439**	**0.371**	**0.317**	**0.315**	**0.341**
			Max	0.567	0.949	0.755	0.589	0.521	0.756
		Hybrid Integer	Min	0.014	0.130	0.053	0.102	0.212	0.077
			Mean	**0.331**	**0.483**	**0.424**	**0.317**	**0.372**	**0.389**
			Max	0.697	0.949	0.906	0.625	0.576	0.732
	Heterozygous	*Binary String*	Min	0.106	0.111	0.053	0.041	0.130	0.026
			Mean	**0.276**	**0.453**	**0.397**	**0.307**	**0.348**	**0.369**
			Max	0.508	0.949	0.906	0.625	0.576	0.732
		Integer	Min	0.152	0.130	0.175	0.041	0.093	0.026
			Mean	**0.284**	**0.272**	**0.296**	**0.346**	**0.285**	**0.355**
			Max	0.460	0.462	0.434	0.564	0.484	0.780

				Front B					
				Conventional			Heterozygous		
				Binary String	Integer	Hybrid Integer	Binary String	Integer	Hybrid Integer
		Hybrid Integer	Min	0.074	0.093	0.158	0.020	0.042	0.026
			Mean	**0.314**	**0.297**	**0.306**	**0.336**	**0.312**	**0.355**
			Max	0.537	0.667	0.566	0.714	0.593	0.780

Table 7-19: *C* metrics for Multiple Objective Heterozygous Piedemonte San Germano after 20 generations

The superior performance trend for the conventionally encoded representations is continued later in the optimization as evidenced by the results presented in Table 7-20. Here their average coverage ratios are improved over Table 7-19 above. The hybrid integer is the strongest performer followed by the binary string representation.

				Front B					
				Conventional			Heterozygous		
				Binary String	Integer	Hybrid Integer	Binary String	Integer	Hybrid Integer
Front A	**Normal**	*Binary String*	Min	0.200	0.320	0.160	0.330	0.350	0.310
			Mean	**0.363**	**0.484**	**0.336**	**0.516**	**0.479**	**0.416**
			Max	0.500	0.707	0.590	0.710	0.610	0.520
		Integer	Min	0.110	0.090	0.050	0.310	0.360	0.310
			Mean	**0.260**	**0.388**	**0.254**	**0.499**	**0.463**	**0.408**
			Max	0.460	0.636	0.470	0.660	0.590	0.530
		Hybrid Integer	Min	0.140	0.190	0.090	0.290	0.390	0.330
			Mean	**0.410**	**0.513**	**0.365**	**0.533**	**0.496**	**0.419**
			Max	0.630	0.747	0.630	0.720	0.590	0.520
	Heterozygous	*Binary String*	Min	0.090	0.160	0.090	0.090	0.110	0.110
			Mean	**0.234**	**0.459**	**0.316**	**0.301**	**0.456**	**0.381**
			Max	0.420	0.747	0.630	0.610	0.590	0.520
		Integer	Min	0.110	0.130	0.060	0.080	0.040	0.080
			Mean	**0.223**	**0.257**	**0.209**	**0.307**	**0.278**	**0.193**
			Max	0.310	0.390	0.340	0.520	0.480	0.310
		Hybrid Integer	Min	0.100	0.170	0.100	0.110	0.150	0.130
			Mean	**0.263**	**0.303**	**0.258**	**0.349**	**0.344**	**0.248**
			Max	0.370	0.420	0.390	0.600	0.600	0.470

Table 7-20: *C* metrics for Multiple Objective Heterozygous Piedemonte San Germano after 100 generations

At the completion of the optimization runs, the *C* coverage metrics (Table 7-21) it can be seen that the Pareto fronts obtained with the conventional encoding tend to dominate those obtained with the heterozygous encoding, with over 50% coverage for all of the genotype representations.

				Front B					
				Conventional			Heterozygous		
				Binary String	Integer	Hybrid Integer	Binary String	Integer	Hybrid Integer
Front A	Normal	Binary String	Min	0.090	0.090	0.100	0.390	0.490	0.250
			Mean	**0.204**	**0.256**	**0.197**	**0.539**	**0.565**	**0.518**
			Max	0.300	0.410	0.310	0.740	0.660	0.680
		Integer	Min	0.070	0.080	0.070	0.340	0.490	0.250
			Mean	**0.191**	**0.230**	**0.180**	**0.524**	**0.547**	**0.501**
			Max	0.330	0.420	0.320	0.740	0.620	0.660
		Hybrid Integer	Min	0.100	0.090	0.090	0.370	0.490	0.240
			Mean	**0.247**	**0.292**	**0.214**	**0.534**	**0.553**	**0.505**
			Max	0.350	0.450	0.360	0.760	0.630	0.680
	Heterozygous	Binary String	Min	0.030	0.080	0.040	0.010	0.080	0.050
			Mean	**0.089**	**0.265**	**0.193**	**0.298**	**0.499**	**0.469**
			Max	0.180	0.450	0.360	0.750	0.630	0.680
		Integer	Min	0.030	0.020	0.000	0.020	0.070	0.030
			Mean	**0.064**	**0.088**	**0.060**	**0.263**	**0.238**	**0.266**
			Max	0.110	0.150	0.130	0.690	0.330	0.630
		Hybrid Integer	Min	0.020	0.020	0.010	0.000	0.010	0.010
			Mean	**0.092**	**0.115**	**0.088**	**0.240**	**0.254**	**0.264**
			Max	0.230	0.280	0.270	0.720	0.450	0.680

Table 7-21: *C* metrics for Multiple Objective Heterozygous Piedemonte San Germano after 1,000 generations

7.4.2.5. Runtime Performance

The more complex genetic structure of the conventional representation cause it to show significantly reduced performance with respect to the heterozygous approach – as seen in Table 7-22. Once more, as with the other analyses, the integer runtime proves to be the quickest representation with the hybrid integer and binary string having roughly equal runtime performance although the hybrid integer representation proves to be marginally quicker for the heterozygous encoding and marginally slower for the conventional encoding.

Chromosome Representation	Conventional		Heterozygous	
	(evaluations per second)	*% of best*	*(evaluations per second)*	*% of best*
Binary String	1,040.43	71.5%	1,380.77	94.9%
Integer	1,145.28	78.7%	1,455.56	100%
Hybrid Integer	1,021.22	70.2%	1,397.16	96.0%

Table 7-22: PSG Multiple Objective Heterozygous Runtime Performance vs. Conventional Performance

7.4.3. Caching

As with the other two networks employed, the caching performance of this multiple objective algorithm is much improved with respect to the single objective formulation of the same problem. Table 7-23 illustrates these results: demonstrating a significant improvement in runtime for the Judy cache. The search effectiveness of the Red-Black binary tree cache is compromised by the length of the chromosome for this problem and is seen to have nearly the same performance as the uncached algorithm – despite the cache being "hit" for 15% of the evaluations.

Cache Size	Runtime performance	Cache Performance		Relative Runtime Performance
	(solutions/second)	*(no. hits)*	*(% of evaluations)*	
None	1,021.22	n/a	n/a	100%
40,000	1,028.60	15,145.2	15.1%	99.3%
Judy (unlimited)	1,098.90	14,605.2	14.6%	92.9%

Table 7-23: Cache results: Multiple Objective Piedemonte San Germano

7.5. Conclusions

Identifying the benefits accrued by the individual representations is more difficult to quantify for a multiple objective optimization because of the nature of the multiple results returned and how best to compare them. The Space-Covered metric, S and Coverage metric, C, of Zitzler and Thiele (1999) are employed to undertake the comparisons of the Pareto-optimal fronts obtained from repeated optimizations using the different combinations of representation.

The most instructive result seen relates to the performance of the binary string representation during the initial stages of the optimization. For all three networks analysed, this representation can be seen to perform best in terms of the convergence of the algorithm – in direct contrast to the results observed for this representation in the single-objective optimizations. To verify the results observed for the binary string representation, each of the experiments were repeated with a version of the hybrid integer, which does not incorporate the Gray coding and, instead, behaves as a conventional binary string. The results obtained for this representation were almost identical to those obtained for the standard binary string – as would be expected. The only variation apparent was owing to the selection of different random seeds for the initialization of each optimization. Despite the more rapid convergence characteristics of the binary string representation, the hybrid integer

representation ultimately produced the best optimization results for all of the problems investigated.

That the hybrid arrangement ultimately outperforms the conventional binary string is not surprising - owing to its superior mutation performance identified in Chapter 4.2.4. The binary string behaviour at the beginning of the optimization should be explored, however. It might be speculated that the addition stochasticity afforded by the non-Gray code binary string representation is beneficial in driving the algorithm to explore the Pareto front in the early stages of the optimization. In this case, a more aggressive crossover regime, such as Uniform Random (Syswerda, 1989) may exert more evolutionary pressure on the other representations and permit them to converge more rapidly. To test this hypothesis, the hybrid integer runs for the multiple objective Piedemonte San Germano network were repeated with such a crossover and the results were seen to be far worse than both the binary string implementation and the original performance of the hybrid integer. A similar test in which the effective mutation rate was increased yielded a slight improvement in the performance of the hybrid integer representation but this remained considerably short of that seen for the binary string. It is recommended that this behaviour be examined in more detail and an opportunity for exploiting the nature of the hybrid integer representation to take account of these results is detailed for further research in Chapter 9.1

The performance of the caching methodology is significantly improved relative to that of the single-objective formulations of the optimization problems. In the reruns of the hybrid integer algorithms outlined above, the proportion of hits of the cache dropped from around 15% to around 3% - more in line with the initial experimentation results seen in Chapter 4.4.3.

Chapter 8. Large Scale Optimization Problems

8.1. Introduction

In order to establish whether the conclusions drawn in Chapter 7 are more widely applicable, some of the techniques introduced are applied to two further optimization problems, which are characterized by an increase in scale and an increase in the computational complexity associated with its hydraulic-related calculations. The first network is analysed in a similar fashion to those in the previous chapter to determine its sensitivity to the different genetic representations under consideration. Moreover, its performance with the caching and the distributed evaluation methodologies is examined. The second network is derived from Piedemonte San Germano model seen previously; in this instance, the problem is reformulated as a hydraulic model with uncertain demands that are required to be simulated over a 24-hour period. This uncertainty is accommodated using sampling techniques that, when coupled to the optimization process, significantly increase the runtimes of the algorithm through repeated, addition hydraulic simulations. This problem is used to evaluate the efficacy of the Non-Repeating Genetic Algorithm introduced in Chapter 4.5 and the distributed evaluation methodology. Owing to the stochasticity introduced as a means of accommodating the uncertainty in the hydraulic model, this problem is not suitable for analysis with respect to its caching performance. This is because, in such problems, there is no longer a one-to-one mapping between a given set of decision variables and the values of the objectives.

8.2. "Real World" Network

8.2.1. Problem Formulation

The "Real World" network was introduced by Savić *et al.* (2000) and is an anonymized network from a UK water company. This model is gravity fed by a single reservoir via 632 pipes to 535 demand nodes – the topology is illustrated in Figure 8-1. Each of these pipes may be implemented as one of 20 potential pipe diameters, which results in a very large search space of $20^{632} = 1.78 \times 10^{822}$. Because of the extended runtimes associated with this problem, a restricted set of four runs was performed with the best Pareto front obtained from each representation used in the following analysis.

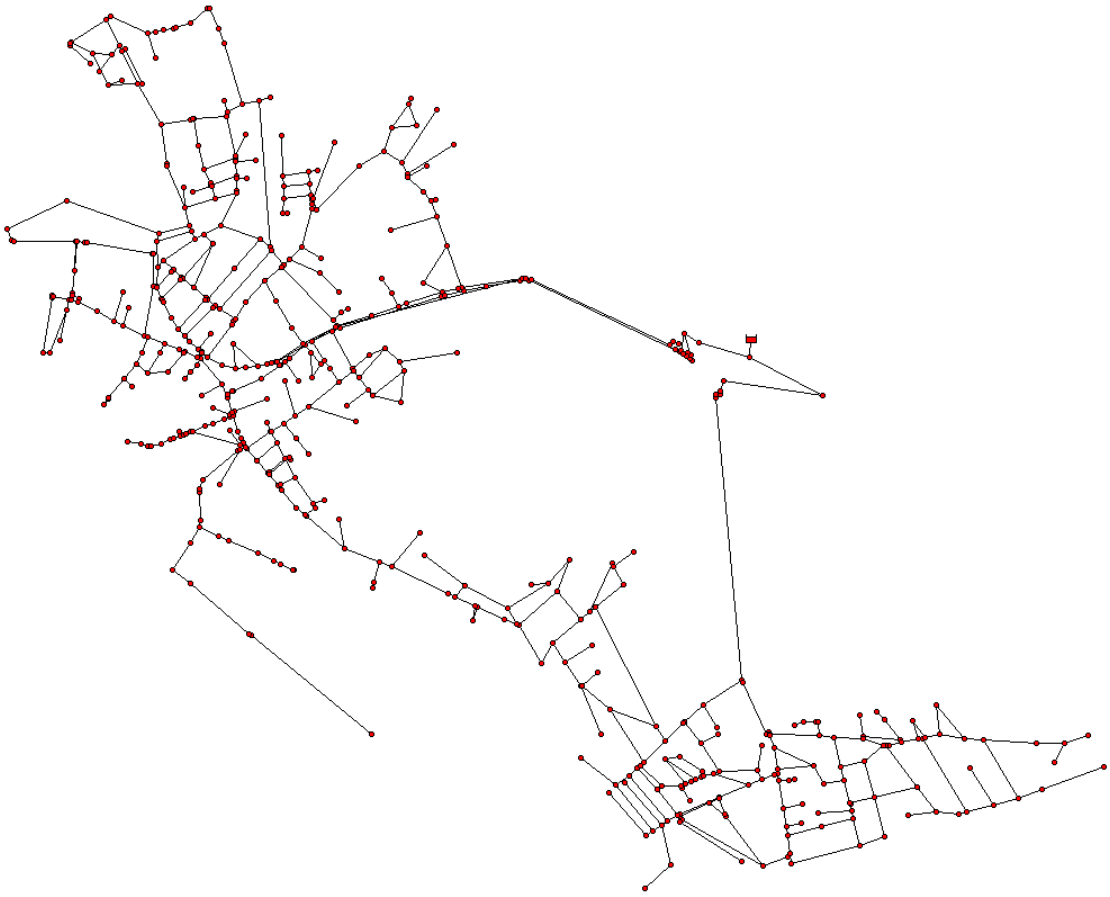

Figure 8-1: "Real World" network topology

In common with the previous multiple objective optimizations, the problem formulation is for the minimization of infrastructure cost and aggregate nodal head deficit and can be represented, thus:

$$Minimize: C_{inf} = f(D_1, .., D_{N_l}) = \sum_{j=1}^{N_l} C(D_j, L_j)$$

xiii)

$$Minimize: T = f(H_1, .., H_{N_n}) = \sum_{i=1}^{N_n} (H_{i,min} - H_i)$$

xiv)

$$D_j \in D \ (j = 1, .., N_d)$$

xv)

where: C_{inf} is the total infrastructure cost, N_l is the number of links in the network for which reinforcement is an option, $C(D_j,L_j)$ is the cost of the j^{th} pipe with diameter D_j (chosen from a

discrete set of available diameters D) and length L_j. T is the total head deficit (negative if a pressure surplus exists), H_i is the pressure head at node i (as computed by the hydraulic solver), $H_{i,min}$ is the minimum pressure head requirement sufficient to fully satisfy the demand at node i and N_n is the number of nodes in the network. N_d is the number of decision variables in the optimization.

8.2.2. Genetic Representation

8.2.2.1. Comparative Analysis

Graphical

The results from the best front produce by runs of each representation are illustrated in Figure 8-2 for the state after 100 generations. As can be seen, the binary string representation performs well, although the integer and hybrid integer arrangements perform appear to perform better at the extremes of the Pareto-optimal front. Outside the range illustrated in Figure 8-2, the standard binary string performs poorly relative to the other representations demonstrated by the results seen in the coverage metrics for this stage of the optimization (Table 8-2).

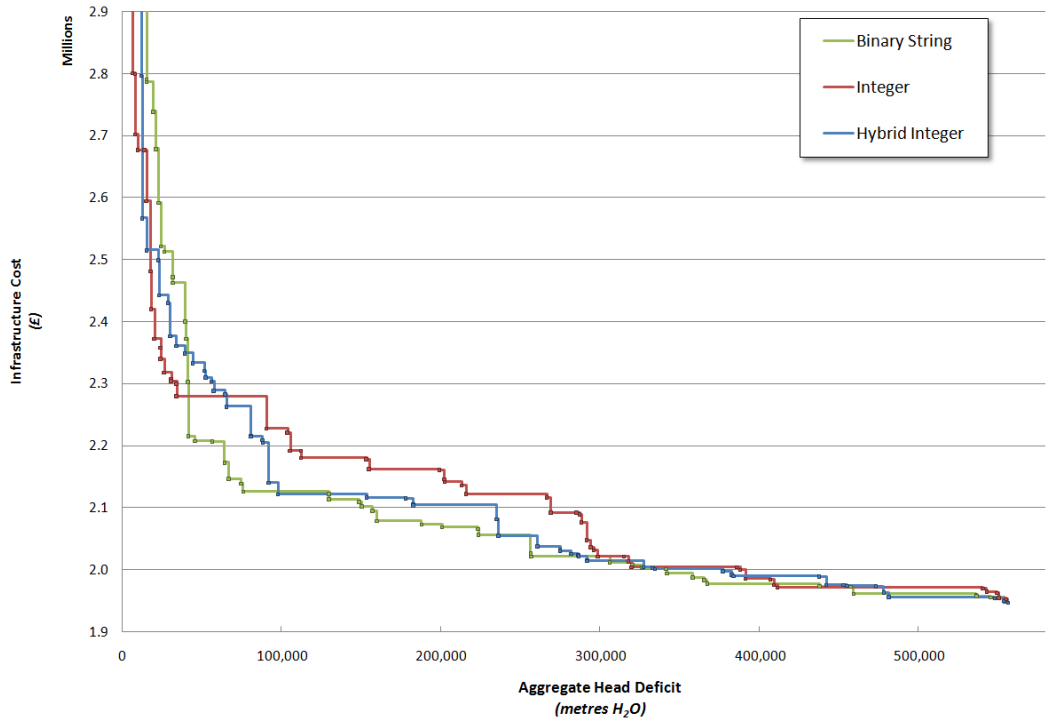

Figure 8-2: Best Pareto fronts for "Real World" problem after 100 generations

After 1,000 generations, the performance of the three representations is much closer, as can be seen from Figure 8-3. In contrast with the earlier stage of the algorithm, however, the hybrid integer variant is performing well, achieving a better spread of solutions.

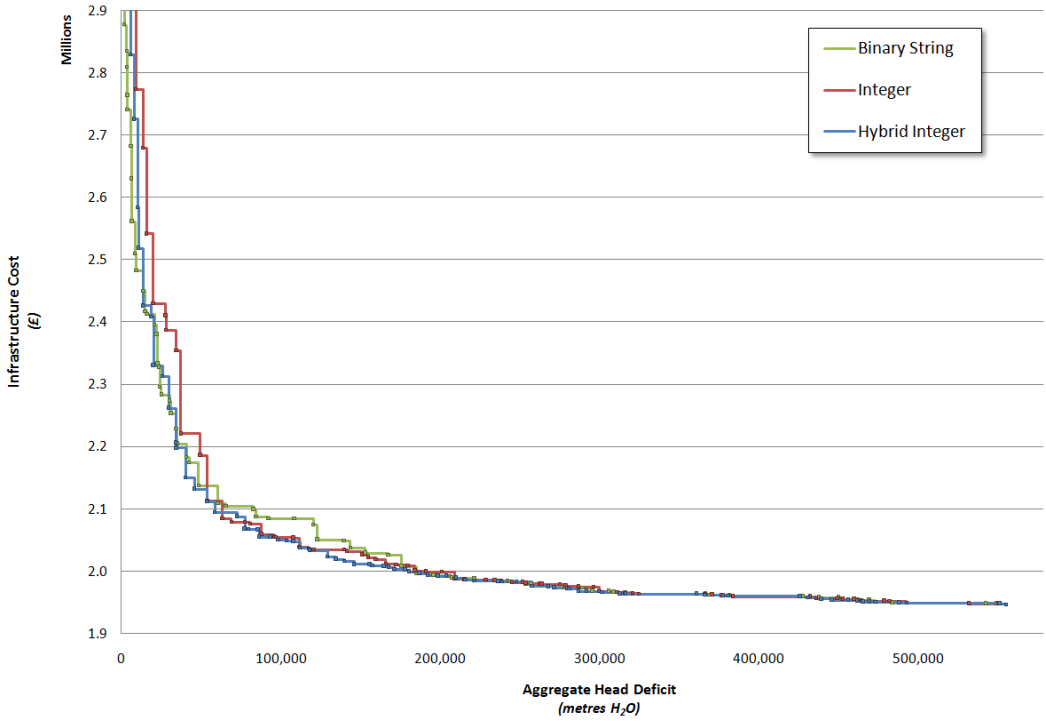

Figure 8-3: Best Pareto fronts for "Real World" problem after 1,000 generations

As for the previous problems explored in Chapter 7, by the end of the optimization the different representations have largely converged to similar Pareto fronts as can be seen in Figure 8-4, below.

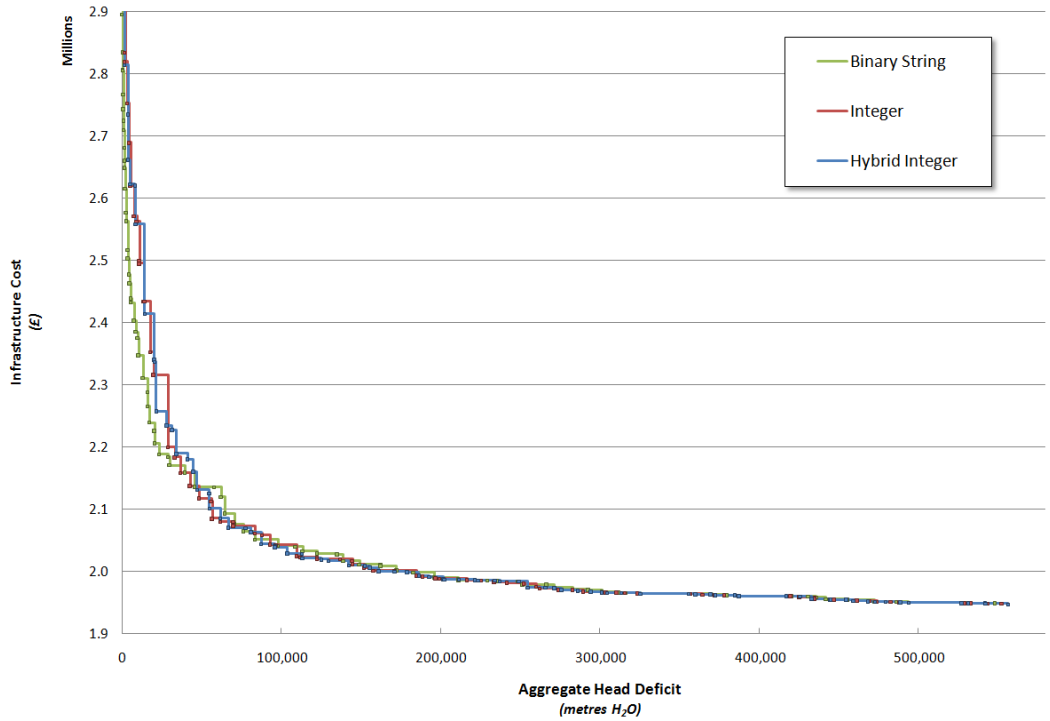

Figure 8-4: Best Pareto fronts for "Real World" problem after 10,000 generations

Space Metric

Table 8-1 shows the space-covered, *S*, metrics for the best runs of each genetic representation after 100, 1,000 and 10,000 generations. These figures show that, as for the prior multiple-objective runs in Chapter 7, the binary string representation converges more quickly – though ultimately the other representations produce similar results.

Generation	Binary String	Integer	Hybrid Integer
100	8.52589×10^{10}	9.28513×10^{10}	8.76128×10^{10}
1,000	4.30702×10^{10}	4.87071×10^{10}	4.20156×10^{10}
10,000	3.38164×10^{10}	3.56136×10^{10}	3.61863×10^{10}

Table 8-1: *S* metrics for Real World network after 100, 1,000 and 10,000 generations

Coverage Metric

The *C* coverage metrics for the obtained fronts are compared for the state of the population after 100, 1,000 and 10,000 generations. In contrast to the *S* metric above, it can be seen from Table 8-2 that, after 100 generations the integer representation can be seen to be the best performing of all, in terms of coverage, followed by the hybrid integer. These results are due to the large number of points identified at the extreme of the Pareto fronts in which the two integer representations dominate the binary string.

		Front B		
		Binary String	*Integer*	*Hybrid Integer*
Front A	**Binary String**		0.31	0.34
	Integer	0.68		0.59
	Hybrid Integer	0.63	0.36	

Table 8-2: *C* metrics for Real World network after 100 generations

Table 8-3 represents the state after 1,000 generations in which it can be seen that the hybrid integer clearly dominates the fronts obtained by the other representations – though the integer representation is now the least well performing in terms of coverage.

		Front B		
		Binary String	*Integer*	*Hybrid Integer*
Front A	**Binary String**		0.47	0.26
	Integer	0.20		0.09
	Hybrid Integer	0.39	0.70	

Table 8-3: *C* metrics for Real World network after 1,000 generations

At the conclusion of the optimization, it is seen, once more, that the optimization has converged to similar Pareto fronts – evidenced by the graphical overlap, the similar *S* metrics and the very similar *C* metrics presented in Table 8-4

		Front B		
		Binary String	*Integer*	*Hybrid Integer*
Front A	**Binary String**		0.29	0.32
	Integer	0.23		0.39
	Hybrid Integer	0.26	0.34	

Table 8-4: *C* metrics for Real World network after 10,000 generations

8.2.2.2. Runtime Performance

The influence of the scale of the network problem is clear from the runtime performance related in Table 8-5. It can be seen that the algorithm manages ~34 solutions per second for this 632 pipe problem versus the next slowest network, Piedemonte San Germano, for which in excess of a thousand solutions per second were obtained.

Chromosome Representation	Average Performance *(evaluations per second)*	% of best performance
Binary String	33.53	96.3%
Integer	34.81	100%
Hybrid Integer	33.88	97.3%

Table 8-5: Real World network Multiple Objective Runtime Performance

The similarity of these performance results reveal that with such an intensive evaluation function, the performance advantages of any particular representation are negated as they become dominated by the runtime of the evaluation function, reinforcing the need to intelligently steer the optimization process.

8.2.3. Caching

The results of the caching for this large-scale problem are presented in Table 8-6. Application of a genetic algorithm to this problem results in a long chromosome of 632 genes being employed. The effect of this on the two caching strategies investigated is profound. The Red-Black binary tree suffers poor search performance owing to the length of the chromosome being used as the search key into the cache. The Judy cache on the other hand can be seen to improve the overall performance of the optimization. A runtime saving of 3.8% may not seem particularly significant but it represents a saving of around 20 minutes on an optimization that took approximately eight hours to complete.

Cache Size	Runtime performance *(solutions/second)*	Cache Performance *(no. hits)*	Cache Performance *(% of evaluations)*	Relative Runtime Performance
None	33.88	n/a	n/a	100%
40,000	25.09	12,613.8	1.26%	135.0%
Judy (unlimited)	35.22	18,191.6	1.82%	96.2%

Table 8-6: Cache results: Multiple Objective Real World problem

8.2.4. Distributed Performance

The Real World problem is by far the most complex problem analysed in this thesis and shows a baseline performance (Table 8-7) comparable to that of the extreme version of the Piedemonte San Germano network employed in Chapter 5.4, which simulates that network for 24 hours at 1-minute intervals.

Computer	Baseline Performance *(evaluations/second)*	Number of Processor Cores	Theoretical Maximum Throughput *(evaluations/second)*
X	33.9	4	135.6
Y	18.2	2	36.4
Z	12.2	2	24.4
Total	**64.3**	**8**	**196.4**

Table 8-7: Theoretical maximum performance for distributed "Real World" problem

For this large, complex network, it can be seen in Table 8-8 that the performance of deEPANET is improved such that the system as a whole has a higher throughput relative to the baseline performance than for all the other models employed previously. The performance reaches some 97.8% of the theoretical maximum derived from the baseline performance. The most complex of the other models, the Piedemonte San Germano problem, achieved 91% of the maximum throughput when evaluated with deEPANET. This can be seen as a direct consequence of the reduced quantity of network traffic allowing the Client X to commit more processor time to its Server threads. However, for this model, this will be offset to some extent by the fact that for such a large model, the volume of data transferred for each evaluation (i.e. the pipe diameter settings and the nodal pressures returned) is significantly greater than for the smaller models. Irrespective of this, given that much of the overhead in network communication, for small amounts of data, is incurred in establishing connections between computers rather than the actual data transfer, this negative impact should be minimized.

Computer	Baseline Performance *(Evaluations/Second)*	Distributed Performance *(Evaluations/Second)*	
X	33.5	31.7	130.7
		33.7	
		34.1	
		31.2	
Y	18.2	17.1	38.9
		19.0	
		2.8	
Z	12.2	10.7	22.4
		9.8	
		1.9	
Totals	**63.9**		**192**

Table 8-8: "Real World" distributed performance results

8.3. Stochastic Piedemonte San Germano

8.3.1. Problem Formulation

Tricarico *et al.* (2005) investigated the demand characteristics of the Piedemonte San Germano network through long-term instrumentation and recording of network flow data and water meter readings. A probabilistic model was then developed to account for the uncertainty associated with the demand at the daily peak hour. A probability density function (PDF) was derived for each node in the network, according to the nature of its consumers and the number of consumers associated with the node. This work was later extended by de Marinis *et al.* (2007b) in which the extended period simulation (EPS) of the network was considered and it was noted that different PDF models, Poisson, Normal and Log Normal, were found to be appropriate for modelling different demand levels during the day. This latter model is used here as a test for the Non-Repeating Genetic Algorithm formulation and to demonstrate the effectiveness of the deEPANET distributed evaluation software introduced in this thesis.

In terms of the complexity of the genetic algorithm, the genetic representation of this problem is identical to the multiple-objective problem presented previously in Chapter 7.4. However, the formulation of the objectives is entirely different. Instead of the twin objectives of infrastructure cost and aggregate head deficit seen in equations xiii) and xiv), the stochastic optimization considers infrastructure cost (equation xvi) and reliability – measured as the probability of demands being met whilst meeting minimum pressure requirements at all nodes (equation xvii), thus:

$$Minimize: C_{inf} = f\left(D_1, .., D_{N_l}\right) = \sum_{j=1}^{N_l} C\left(D_j, L_j\right) \quad \text{xvi)}$$

$$Maximize: R = P\left[H_i \geq H_{i,min}\right] \quad \forall\, i \in \{1, .., N_n\} \quad \text{xvii)}$$

$$D_j \in D \; (j = 1, .., N_d) \quad \text{xviii)}$$

where: C_{inf} is the total infrastructure cost, N_l is the number of links in the network for which reinforcement is an option, $C(D_j,L_j)$ is the cost of the j^{th} pipe with diameter D_j (chosen from a

discrete set of available diameters D) and length L_j. R is the reliability, H_i is the pressure head at node i (as computed by the hydraulic solver), $H_{i,min}$ is the minimum pressure head requirement sufficient to fully satisfy the demand at node i and N_n is the number of nodes in the network. $P[H_i \geq H_{i,min}]$ is the probability that H_i is equal to or exceeds $H_{i,min}$ for all of the nodes in the network simultaneously and is assessed as the ratio of stochastic demand samples (see below) for which this condition is true. N_d is the number of decision variables in the optimization.

The computation complexity of the objective function is significantly greater than that of the steady state, deterministic problem, requiring the hydraulic solver to run 96 times (providing results for 24 hours at 15-minute intervals). Furthermore, in order to accommodate uncertainty in the level of demand at each node, this extended hydraulic solution is performed for each of 20 stochastic samples of demand (applied individually at each node) for each candidate solution developed by the genetic algorithm. The results of this stochastic sampling are aggregated as a number of statistical measures of performance and stored as part of the individual in the population.

8.3.2. Non-Repeating Genetic Algorithm

The Non-Repeating GA (NRGA) introduced in Chapter 4.5 is an amalgam of a conventional GA and the caching technology developed in this thesis, which permits a GA to avoid unnecessary duplication of solutions within a population and, in the case of a robust stochastic GA, Kapelan *et al.* (2003a), to maintain the statistics for solutions that are removed from the population but which are later reintroduced. The methodology operates as an extension to the recombination operations. Following the selection and recombination that results in two new children, the existing population is searched to determine whether an existing chromosome matches that of either child. If a match is identified in the current population then the selection and recombination process is repeated, until two unique children have been identified – thus preventing duplicate solutions from entering the population in the first instance. Having identified these children, an archive is then searched to determine whether the chromosomes of the children have been encountered before. The archive is populated with the individuals that are removed from the population during the optimization process – the archive being implemented with the same Judy technique employed by the caching methodology of Chapter 4.4. If a child is identified in the archive then instead of it starting in the population with null statistics, the statistics from its former

incarnation are used. The net effect of this arrangement is that an individual chromosome may pass in and out of the population as the optimization proceeds without this affecting its aggregated statistics.

In order to evaluate the performance of the NRGA on the stochastic Piedemonte San Germano problem, the stochastic algorithm outlined above has been run using three conditions:

1. The baseline scenario adopts the GA parameterization of Tricarico *et al.* (2005) used for solving the steady state (i.e. non-EPS) rehabilitation problem for the hour of peak water demand. The principal differences between this parameterization and that subsequently employed are the use of integer genes and the adoption of the Uniform Random (Syswerda, 1989) crossover operator. A direct comparison between the results of Tricarico *et al.* (2005) and those presented in this thesis is not possible owing to the significant underperformance of the former resulting from significant shortcomings in the implementation of the basic NSGA-II algorithm.
2. The second parameterization is that employed by de Marinis *et al.* (2007b) for solving the EPS version of the Piedemonte San Germano problem. This form uses the standard one-point crossover and adopts the hybrid integer genes introduced in this thesis in Chapter 4.2.3.
3. The third scenario is identical to that of de Marinis *et al.* (2007b) with the addition of the NRGA operation, which determines whether proposed new solutions are extant in the current population or have been encountered previously, *before* the evaluation process is undertaken.

In each of these cases, the robust NSGA-II of Kapelan *et al.* (2003a) is employed with a minimum chromosome age of 20 generations. That is to say, that the solutions are only reported as being part of the Pareto-optimal front if they have survived for 20 generations or more. Figure 8-5 illustrates the results (i.e., Pareto solutions) obtained for the above three scenarios after the optimization has run for 100 generations (or 10,000 evaluations – given a population size of 100 individuals). A fourth curve in this figure shows the further progress of the first scenario after an additional 300 generations (i.e., 40,000 evaluations in total) have been undertaken.

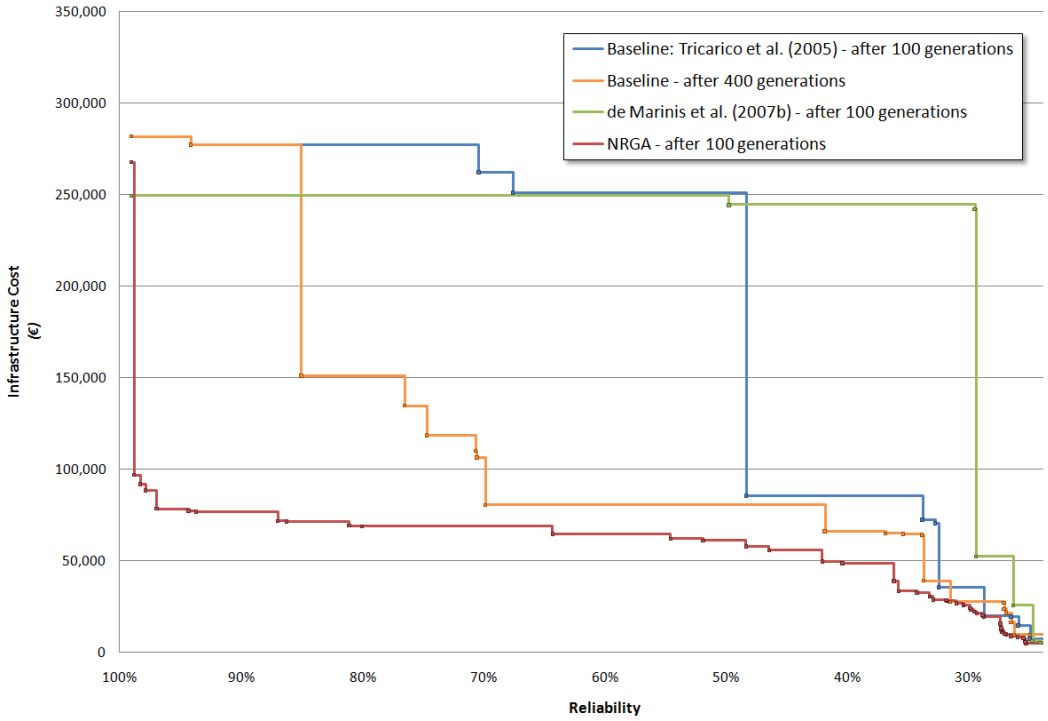

Figure 8-5: Non-Repeating GA performance comparison

The outstanding performance of the NRGA relative to the other scenarios is clearly shown by the red line in Figure 8-5. In this experiment, the NRGA identified 483 individuals (4.8% of the evaluations undertaken) that were either already extant in the population or that had been previously encountered and which were resurrected with their previously obtained statistical data intact. The effect of this is quite dramatic with the NRGA returning 52 solutions along the Pareto-optimal front after 100 generations compared with just 13 and 8 for scenarios (1 - blue) and (2 - green) above. It should be reiterated that the *only* difference in the algorithm applied by de Marinis (2007b) and that of scenario (3 - red) is the addition of the NRGA analysis. The fourth (orange) curve in Figure 8-5 is that of the first scenario after a further 300 generations (30,000 evaluations) have been undertaken. As can be seen, this result, though significantly improved over the 100-generation result with 22 solutions on the Pareto-optimal front, continues to be wholly dominated by the NRGA result obtained with one quarter of the evaluations.

This result appears to be driven, principally by the resurrection of individuals that have previously been encountered in the population but have been removed. In the normal algorithm, as applied in the first two scenarios, when an individual is removed from the population no memory of it is retained – thus if it is later reintroduced by the GA then its age will revert to zero. With the NRGA operative, the age is reset to the age the individual had

when it was removed from the population making it much easier for that solution to, cumulatively, reach the minimum age threshold specified by the optimization. This is evidenced by the five-fold increase in the number of solutions identified on the Pareto-optimal front by the NRGA-enabled algorithm after 100 generations, compared to the other approaches and the clear improvement in solution quality evidenced by the NRGA Pareto front in Figure 8-5.

8.3.3. Distributed Performance

The deEPANET distributed evaluation methodology introduced in this thesis in Chapter 5 was designed specifically to reduce the extensive runtimes associated with the stochastic optimization techniques presented by Kapelan *et al.* (2003a). Accordingly, deEPANET was designed with the capability to offload the generation of the statistical samples to the server computers. For each solution generated by the GA, a PDF describing the demand for each node is transferred to the server computer, which then generates the appropriate number of samples according to the PDF and simulates the hydraulic network for each set of samples.

deEPANET was utilised by de Marinis *et al.* (2007b) to reduce the runtime of this EPS version of the Piedemonte San Germano problem and a reduction in runtime was reported from 17 hours on a single computer to 2 hours using the distributed system.

Computer	Baseline Performance *(evaluations/second)*	Number of Processor Cores	Theoretical Maximum Throughput *(evaluations/second)*
X	78.8	4	315.2
Y	45.7	2	91.4
Z	32.0	2	64.0
Total	**156.5**	**8**	**470.6**

Table 8-9: Theoretical maximum performance for distributed, stochastic Piedemonte San Germano problem

Table 8-10 shows the aggregated performance values obtained while optimizing the stochastic formulation of the Piedemonte San Germano problem. Once more, these results demonstrate good scalability with the total throughput of the algorithm approaching that of the theoretical maximum. In this instance, this is achieved partly by the devolution of the PDF generation to the individual servers and partly by the consequent reduction of the amount and, more importantly, the frequency of the network traffic involved in the optimization.

Computer	Baseline Performance *(Evaluations/Second)*	Distributed Performance *(Evaluations/Second)*	
X	78.8	75.2	305.5
		76.1	
		76.5	
		73.8	
Y	45.7	40.9	85.8
		39.7	
		5.2	
Z	32.0	26.2	54.4
		24.5	
		3.7	
Totals	**156.5**		**445.7**

Table 8-10: Stochastic Piedemonte San Germano distributed performance results

8.4. Conclusions

The introduction of the large scale, "Real World" network demonstrates that a fixation with the runtime performance of individual gene representations is inappropriate when considering the optimization of large-scale hydraulic networks as any advantage thus accrued is dominated by the runtime of the evaluation function. In such situations, however, the caching methodologies proposed might be expected to have a significant impact on runtime – and this is indeed the case with the Judy cache demonstrating improved runtimes for this problem of the order of 4%. Nevertheless, this improvement on its own reinforces the case for streamlining the optimization process as much as possible, reducing unnecessary function evaluations.

The application of the deEPANET distributed evaluation methodology to these two, computationally intensive problems, establishes that the relative performance of the method improves as the complexity of the objective function increases. In the examples given, the performance of the distributed evaluation approaches the theoretical maximum that could be obtained by applying the computers employed separately. These cooperating computers are contributing, however, to a single optimization. This raises the prospect of using genetic algorithm techniques on ever more complex problems, which would previously have been beyond the capabilities of hydroinformatic optimization applications or to apply these techniques to problems in near-real time. This research contributes significantly to realising this application of many processors distributed across many computers to a single optimization at a level of performance approaching the theoretical maximum.

The Non-Repeating Genetic Algorithm introduced in Chapter 4.5, which draws upon the Judy caching technique explored previously, is shown to be highly effective at reducing the number of evaluations required to perform stochastic optimization. Moreover, the application of this method results in significantly better solutions (i.e. Pareto-optimal fronts) This has been achieved by preserving a statistical memory of individuals that are removed from the population and later resurrected and by preventing the GA from considering solutions, which are already extant in the population and thus avoiding unnecessary duplication. Such an improvement in performance may have considerable implications for other optimization applications in hydroinformatics in which stochastic sampling, is performed.

Chapter 9. Conclusions

An extensible, open architecture for the implementation of Genetic Algorithms applied to Water Distribution Systems is presented. Several novel, problem-independent techniques are presented for improving the performance of evolution algorithms within the context of hydroinformatic applications.

The objectives as originally formulated were to:

- Evaluate the effectiveness and relative performance of alternative genetic representations for chromosomes in evolution algorithms with respect to runtime and solution quality considerations.
- Assess the potential for advanced caching and archiving techniques to reduce the runtime of evolution algorithms.
- Determine the value of distributed evaluation of hydraulic network simulation to facilitate the massive parallelization and acceleration of evolution algorithms for the optimization of water distribution networks.

In the fulfilment of these objectives, a number of significant outcomes have been achieved in the following areas:

- Genetic Representation – the development of a hybrid integer gene which combines the algorithmic advantages of the classical binary string representation with the performance advantages afforded by integer gene implementations.
- Solution Caching – Caching techniques have been examined and have been found to accelerate evolution algorithms with computationally intensive objective functions.
- Non-Repeating GA – an extension of the caching methodology is found to radically improve the performance of multiple objective algorithms that employ stochastic sampling to accommodate conditions of uncertainty.
- Adaptive Mutation – the improvement of single objective algorithm convergence performance is demonstrated through the use of a "learning" mutator which identifies, statistically, individual genes which have greater significance to the fitness of an individual.

- Distributed Evaluation – a technique for the parallel evaluation of hydraulic networks is presented and demonstrated to offer effective and scalable acceleration for evolution algorithms.

These contributions are further elaborated in the following sections.

Genetic Representation

A hybridized integer gene representation that offers improved computational performance over the conventional "binary string" representation commonly used in GA applications. Whilst this hybrid seems to perform less efficiently in terms of raw performance (i.e. the proportion of runtime associated directly with the manipulation of genetic material) than the adoption of an integer or floating-point representation, the nature of its binary string-like representation is shown to allow it superior algorithmic performance during recombination whilst outperforming the conventional binary string representation. The case studies performed demonstrated the clear superiority of the algorithmic performance of this representation when applied to the single-objective optimization problems: the progress of the optimization runs was seen to converge more rapidly than the alternative representations and, further, to reach better solutions more consistently. The results from the multiple objective optimizations are less clear-cut, however. In these optimizations, the conventional binary string representation was seen to outperform the other representations in terms of convergence during the initial stages of the optimization. By the conclusion of the optimizations, however, the hybrid integer representation demonstrated, on average, superior results.

The investigation into the algorithmic performance of these representations led to the identification of two results for the case study networks that are believed to be the best-published solutions to those problems. The prior best published solution to the Hanoi network, that is deemed feasible when solved with an unmodified version of the EPANET hydraulic solver (Rossman) is seen to have a capital infrastructure cost of $6,134,015.72 (Zecchin *et al.*, 2006) with a minimum surplus head of 0.29m. The solution found during this analysis has a cost of $6,081,127.54 and a minimum surplus head of 0.01m. Whilst lower-cost solutions have been published, these are achieved by relaxing the values of Hazen-Williams coefficients determining the head-loss in the pipes of the system. This results in higher system pressures and, consequently, makes it easier for the optimization process to identify lower-cost solutions that meet the minimum-head requirement. A new optimal

solution was also identified for the deterministic form of the Piedemonte San Germano problem, €30,082 versus the previous best-published result of €31,002 (Tricarico et al,. 2005).

Caching

Exploiting caching techniques is demonstrated to have a significant improvement in performance on large and complex optimization problems. This caching takes place at many scales throughout the implementation of the algorithm, for example retaining values for binary strings instead of recalculating them on every access as well as retaining objective function results in a solution cache. It is shown that the use of such caches has the potential to improve GA performance by ensuring that processor effort is not expended on solutions that have been encountered previously during the optimization. The development of a solution caching methodology for Genetic Algorithms in this thesis represents a novel technique for enhancing their performance and an approach that is under-explored in the literature. Whilst the applicability of the caching has been clearly demonstrated on smaller problems in reducing optimization runtime, it is noted that the effectiveness of the caching, in terms of runtime performance, is highly dependent both on the parameterization of the optimization itself and on the nature of the solution space being explored – related to the propensity of the algorithm to identify the same solution in the solution space. Thus, it is considered that the case for widespread solution caching is yet to be proven. A highly specialised data structure, the Judy Tree is introduced as a repository for population-based data and is seen to outperform alternative representations for maintaining a solution cache.

Non-Repeating GA

The Non-Repeating GA (NRGA) is presented in Chapter 4.5 as a twin approach to improving the performance of the multiple-objective optimization. Firstly, a small-scale cache is used to identify the current members of the population. On the creation of new candidate solutions, this cache is searched to ensure that the candidate solution does not already exist in the population – preserving genetic diversity and, in the case of the stochastic optimization, avoiding the maintenance of separate statistics for identical individuals. The second improvement is achieved through the deployment of a further Judy tree structure, as used in the solution caching methodology, in order to maintain an archive of previously encountered solutions. In doing so, it permits individual solutions to be removed from the general population whilst retaining their accumulated statistics. This is undertaken in order that, should the individual be encountered again, the statistical aggregation may continue

from the point at which the individual was removed rather than restarting the statistical analysis from scratch. A consequence of this is that it is far easier for individuals to meet the minimum-age criterion used in the robust NSGA-II optimization of Kapelan *et al.* (2003a) as they no longer have to survive for a certain number of *contiguous* generations. Instead, they need only meet the minimum-age criterion by existing in the population for that *total* number of generations.

When applied, in Chapter 8.3, to the stochastic Extended Period Simulation (EPS) extension to the Piedemonte San Germano problem (de Marinis *et al.*, 2007b), the NRGA methodology was shown to accelerate the convergence of the algorithm significantly and demonstrated a five-fold increase in the number of solutions identified along the Pareto-optimal front which met the minimum-age criterion.

These results have demonstrated that better solutions (Pareto fronts) are identified due to better utilisation of the collective system memory introduced through the archiving of the statistical data associated with the individual solutions encountered during the progress of the optimization. This improvement in performance could have considerable implications for other optimization applications where stochastic sampling is employed, for example, groundwater remediation, water resources management, etc.

Adaptive Mutation

Novel modifications to the mutation operator are demonstrated. By determining the genes whose values dominate the results of the objective function, the ability to improve GA performance is demonstrated by concentrating mutation operations on these genes. The operator presented is, however, thought to be constrained significantly by the number of "critical genes" within the decision space and its level of performance is, thus, likely to be highly problem specific.

Distributed Evaluation

The Distributed Evaluation for EPANET (deEPANET) - (Morley *et al.,* 2006) implements a specialized application which offers significant performance improvements to optimization applications by allowing parallelized processing of hydraulic simulations either on a single machine or through cooperating computers connected by a local area network. In contrast to the caching methodology, the Distributed Evaluation arrangement for hydraulic networks is shown to be highly effective, and scalable (in that the performance improvements scale according to the number of computer s committed to the problem), for improving the

performance of applications that require repeated hydraulic solutions. The case studies undertaken demonstrate that as the computational complexity of the optimization undertaken increases, the proportion of the theoretical maximum throughput attained, for the computers utilised, increases. In practice, for the larger problems, upwards of 90% of the theoretical maximum performance was consistently obtained demonstrating the scalability of this technique for reducing runtimes for the most complex of optimization problems – those that will most benefit from the application of this technique.

Commercial Exploitation

The methodologies and software developed in this thesis have been employed in a number of commercial software applications: GAnet (Morley *et al.*, 2000), GAcal (Walters *et al.*, 1998) and the WiLCO software (Engelhardt & Skipworth, 2005). These applications have all used one or more of the components for undertaking the optimization of optimal design and rehabilitation, calibration and whole-life-costs in water distribution and sewer networks, respectively. In addition components of the system have been deployed in other research projects, e.g. Fullerton *et al.* (2002) (storm water flow modelling and optimization), Savić *et al.* (1999) (optimal design of expansion to a large-scale hydraulic network), Engelhardt *et al.* (2002) (whole-life-costing for water distribution network management) and de Marinis *et al.* (2007a,b) (estimation of the economic level of reliability for the rehabilitation of water distribution systems).

9.1. Further Research

A number of avenues for further research are proposed to extend the efficiency and effectiveness of the optimization methodologies and software components introduced and their application to the wider domain of hydroinformatics. In particular, the coupling of the caching technology to the stochastic optimization technique in the Non-Repeating GA methodology deserves careful consideration in its application to other optimization applications where uncertainty is considered. The results of the case studies employed demonstrate that the technique delivers considerably improved Pareto-optimal solutions over existing techniques. As such, the effectiveness of this methodology should be assessed on other hydroinformatics applications that employ stochastic sampling.

Genetic Representation

The performance of the binary string representation in the multiple objective experiments demonstrate that it possesses superior performance in convergence whilst the Gray-coded hybrid integer representation behaves better in the local search towards the end of the optimization. It is proposed, therefore, to investigate a new type of hybrid integer gene, which can behave as a conventionally coded and Gray-coded value simultaneously and evaluate its performance relative to the original implementations. Thus for recombination, which dominates the early phase of the optimization, the genes could behave as conventional binary strings as they have shown good performance in these tests. For mutation, which dominates the local search phase towards the end of the optimization, the genes could behave as Gray-coded binary strings. This modification could be accomplished with minimal overhead in performance over and above that of the normal Gray-coded hybrid integer gene. This variation in representation would be a unique attribute of the new genotype and one that would not be possible – or would be computationally undesirable – when using conventional binary string representations.

Caching techniques

Whilst the case for the use of caching as an integral part of evolution algorithms is not conclusive, the methodologies developed to facilitate it may have other application in the field. Given the apparent sensitivity of cache performance to the parameters of the GA – particularly to mutation – it may be possible to apply the cache as an aid to dynamically tune the performance of the algorithm to ensure that it continues to investigate solutions that have not previously been encountered. For example, a methodology could be envisaged where the rate of cache "hits" is continuously monitored and, should that rate vary outside of predefined bounds, the mutation rate – or other operator parameter – could be dynamically changed in an attempt to return the level of cache hits to the accepted bounds.

At present, the caching methodology cannot, reliably, be applied to heterozygous chromosomes. This restriction results from the nature of these chromosomes which are used for the search terms in the cache. Given that two sets of decision variables that implement the same decisions may be defined in the chromosome in different orders, the caching techniques as they stand would be unable to identify this circumstance. Accordingly, it is proposed to investigate mechanisms that will arrange the elements of a heterozygous chromosome in a predetermined order that will allow the caching methodology to correctly identify identical sets of decision variables.

Non-Repeating GA

The NRGA assists in preventing the re-evaluation of previously encountered solutions for stochastic optimization in both single and multiple objective scenarios and is shown to improve dramatically the performance of such optimization. However, the likely members of the solution space that will exhibit the behaviour of moving in and out of the population in the multi-objective circumstance are those that will lie along the periphery of the Pareto front. As such, they may be better accommodated by an archiving multi-objective optimization technique such as emerging optimization techniques, which include some form of archiving of a more restricted set of solutions, such as the ε-NSGA-II algorithm of Reed *et al.* (2005) or the archiving trees of Fieldsend *et al.* (2003). A comparison of such techniques – and their applicability to stochastic optimization, in particular, should be undertaken with a view to identifying the approach that delivers the best algorithmic and runtime performance.

Further Applications

Distributed Evaluation

At present, for evolutionary optimization purposes, deEPANET, transfers data across the network in phenotypic terms – that is in terms of the network element attributes that are to be changed, rather than in genotypic terms – the native genetic representation of the network element attributes being optimized. It would be instructive to examine the effect of offloading further processing onto the server computers by passing the genotypic information from the optimization algorithm directly to the servers in a similar fashion to that, which has been undertaken with the devolving of the stochastic sample generation to the servers. That said, one of the attractions of deEPANET is that the server-side logic is very simple and can be used for many applications without the need for specialization.

Distributed Evaluation for Wastewater Networks

The distributed evaluation methodology presented herein has concentrated on the provision of tools for the modelling of water distribution systems – using the EPANET pressurized hydraulic solver (Rossman, 2000) allied to the OpenNet network modelling architecture described in the Appendices. The range of applications for which this methodology could be brought to bear would be greatly enhanced by the addition of a solver for mixed open-channel/pressurized flow such as SWMM (Rossman, 2005) developed by the United States Environment Protection Agency. Like EPANET, this software is in the public domain and

the source-code can be freely obtained. However, in addition to obvious differences in the modelling approach employed in the two models, the SWMM software differs from EPANET in that it does not expose a coherent Application Programming Interface (API) for controlling the model in the fashion that would be required to integrate it with the existing methodology. Although an API is available, it does not include, for example, the necessary functions for interactively interrogating or condition of the network or for modifying its constituent attributes. This falls far short of the level of control that would be required to marry it to an optimization algorithm in the fashion that EPANET has been used hitherto.

It is proposed develop a suite of "hooks" within the SWMM library to enable the layering of an additional API onto the model to provide the functionality to facilitate the interactive control and interrogation of the network condition. With this in place, the SWMM components can be mapped directly onto their analogues in the OpenNet modelling library – the sewer representations already being extant – and a similar level of optimization functionality as has been achieved with EPANET will be available.

Owing to the extended runtimes associated with sewer modelling, and associated components, it is anticipated that the application of similar distributed evaluation techniques as those applied with deEPANET in Chapter 5 will result in significant shortening of optimization algorithm runtimes. The additional complexity of the model solution would benefit the distributed evaluation technology because of the reduced time overhead associated with network latency relative to the time taken to perform a solution – as has been seen by devolving the stochastic sampling in deEPANET to the server computers.

OpenMI Connectivity

OpenMI is an effort to harmonize the interfacing of related models – particularly those from the hydrologic domain (Blind & Gregersen, 2004). The OpenMI interface specification imposes a number of requirements on the implementation of "models" that conform to it. These constraints arise, principally, as a result of the decision to forego any centralized control module for OpenMI – instead "models" freely interact, synchronously, on a peer-to-peer basis. Thus to proceed to integrate these models without any formal direction from a controller, it is necessary to tightly prescribe the operations of the individual components to ensure their correct interoperation.

One of the key concepts in understanding the operation of the OpenMI is that the system operates entirely synchronously – i.e. in a single thread. This is a sensible approach as it avoids having to accommodate thread-contention issues. However, there is a measure of inflexibility that is the price to pay for this approach – though there appears to be no impediment in adopting an asynchronous approach, provided that it is wholly-contained within components. Whether this would offer sufficient flexibility is a matter for debate and, indeed, depends largely on the granularity of the components to be used in "models" in the first instance.

The potential for adding OpenMI compliant extensions to the optimization software developed in this thesis should be investigated, in order that the optimization techniques herein may applied, without additional programming, to OpenMI-based models.

Appendix A Network Infrastructure Modelling: OpenNet

A.1 Introduction

A recurring necessity in developing the optimization methodologies for this thesis has been for an abstracted network model representation to assist in interfacing between the optimization software developed using the methodologies, e.g. GAnet (Morley *et al.,* 2001) and the hydraulic solvers and other data sources that contribute to the optimization, e.g. StruMap GIS and network modeller (Structural Technologies Ltd., 1996), EPANET (Rossman, 2000) and MapInfo (MapInfo Corporation, 1998). Such an abstracted model is necessary to decouple the optimization from the other elements of the software, in order that dependencies are minimized and flexibility is maintained to integrate additional components as required.

A further fundamental issue with the optimization of water networks has been accommodating the plethora of different hydraulic simulation software packages that might be used as a data source. This has highlighted the absence of an agreed standard for representing water network infrastructure and operating conditions in software. Commonly, in order to optimize commercial networks it has been necessary to translate the network infrastructure from the clients' network-modelling software into a form that can be understood by the used by the optimization software – particularly the hydraulic solver. This process is hampered by differing conventions for representing different hydraulic elements such as valves, pumps etc.

To address these problems, an object-oriented class-library developed in C++ (Stroustrup, 1997), called OpenNet (Morley *et al.*, 2000), has been developed as an abstraction to hide the inner workings of the network solver from the optimization software. This class library is allied to an XML (W3C, 2000) metafile representation of the network, which is designed to facilitate easier dissemination of network infrastructure data – the definitions for which can be found in Appendix B. XML is employed to provide an extensible and transparent means to create network definitions in a file format that is independent of any particular hydraulic solver and which can be adopted as a "halfway-house" between specific modelling representations. OpenNet is equipped with a suite of translators which can read and write not only its own XML format but also the import/export formats of many popular hydraulic modelling packages including SynerGEE (née Stoner Workstation - Advantica Inc.,

2003), InfoWorks WS (Wallingford Software, 2005), Aquis (7-Technologies, 2002) and EPANET (Rossman, 2000).

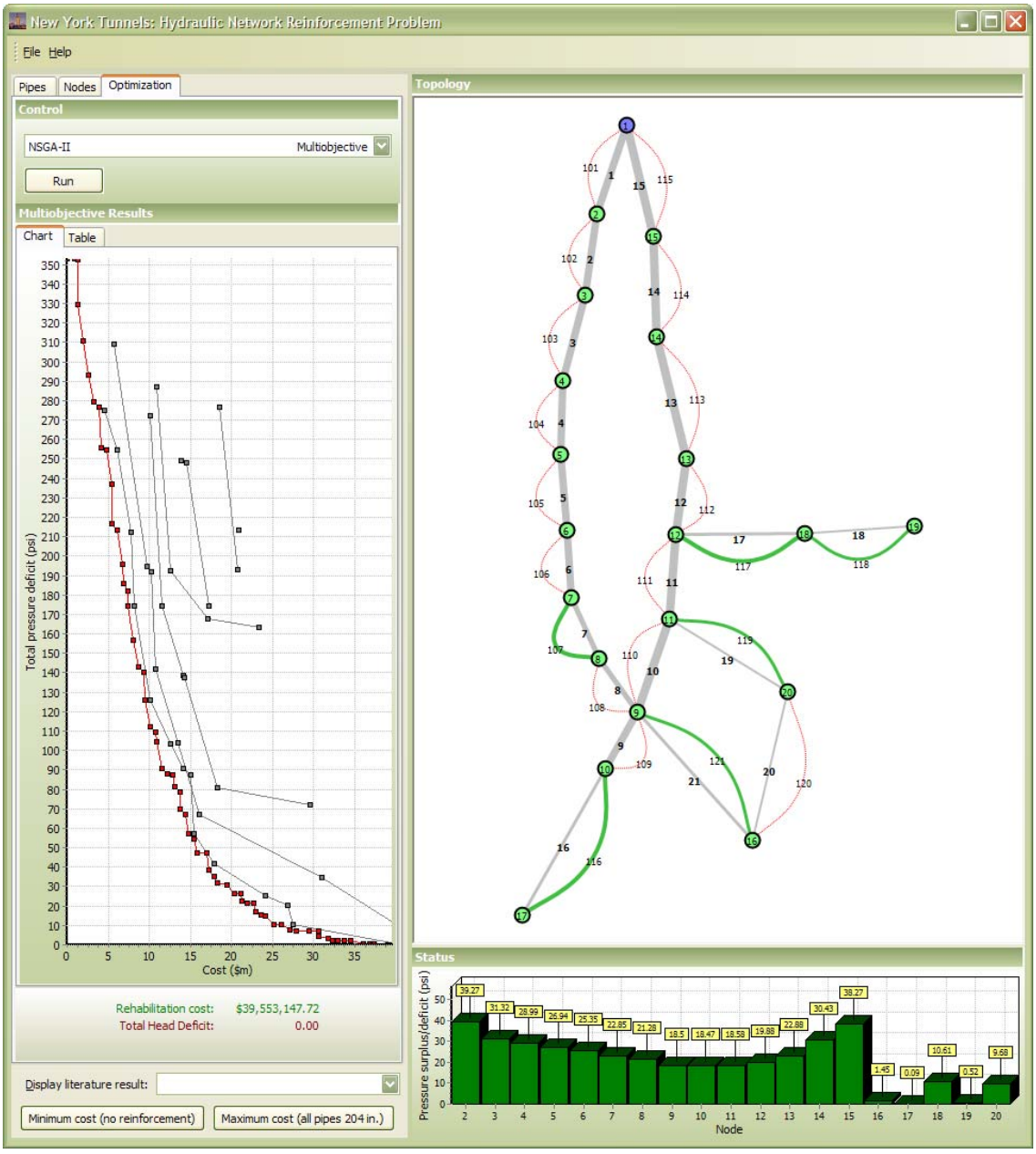

Figure A-1: New York Tunnels-specific version of GAnet with OpenNet visualization component

The class library and XML document structure integrate directly with visualisation routines that can be used to interact with the network model – as seen in Figure A-1. These routines allow the structure and behaviour of the network to be viewed without recourse to an external GIS module or that of a specific modelling package - although data exchange is supported with common desktop GIS applications, including MapInfo (MapInfo Corp., 1998).

A.2 Implementation

The OpenNet abstract network-model class-hierarchy is implemented as a collection of nodes and links and has similarities to other hierarchies, such as that proposed by Solomatine (1996) and subsequently implemented in OOTEN (van Zyl *et al.*, 2003), in that it has an implicit geographic framework as shown in Figure A-2. This allows the network model, which is not limited to hydraulic applications, to link seamlessly with a GIS application, which represents data in a similar fashion. Whereas Solomatine (1996) uses a single registration structure for all hydraulic elements, the OpenNet library implicitly divides all hydraulic elements into node (point), link (line) and area (polygon) elements. Each element type is stored in an independent *ONElementStore* object. This simplifies the maintenance of the referential integrity of the network, facilitates implicit links to objects stored in external GIS applications and exposes a straightforward user interface to third-party developers. To ensure equivalent functionality to Solomatine's single registration store, iterator functions are provided to facilitate easy access to all hydraulic elements.

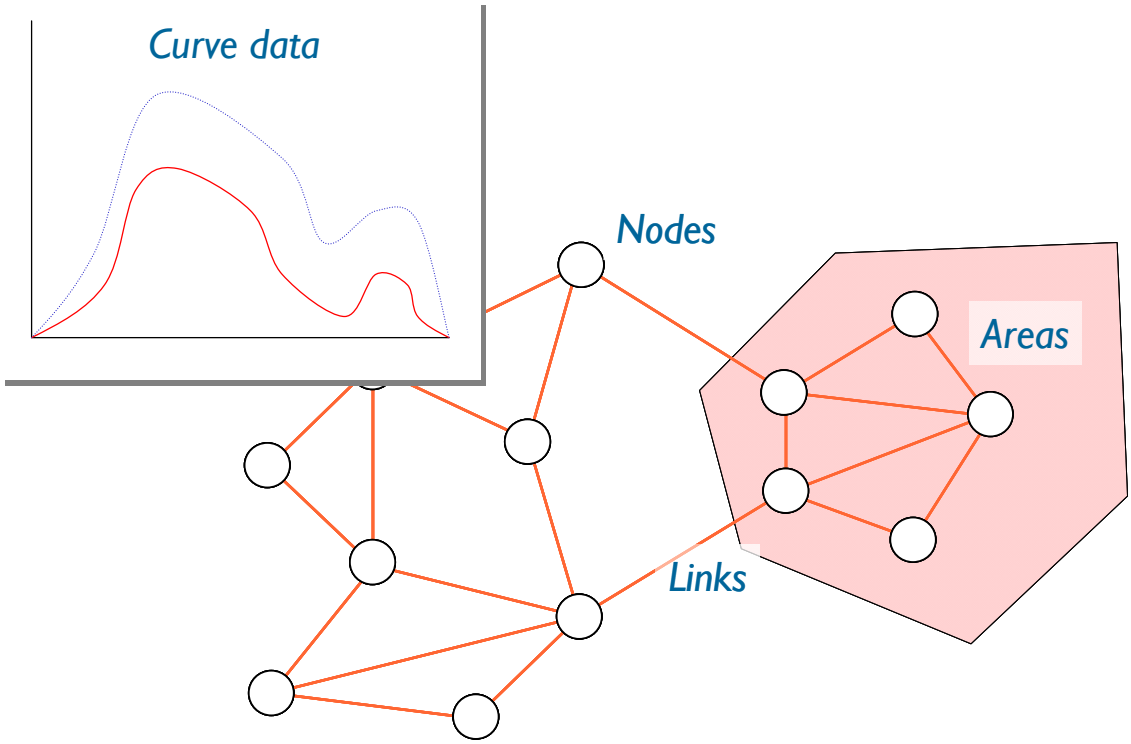

Figure A-2: Constituents of a network representation

As with the Population-based Optimization library before it, OpenNet was initially conceived as a ObjectPascal-based library in Delphi (Borland International, 1997). However, in order to overcome severe performance constraints when handling large networks, the

library was ported to C++ to leverage the more efficient data structures incorporated in its Standard Template Library. The resulting class hierarchy can be seen in Figure A-3.

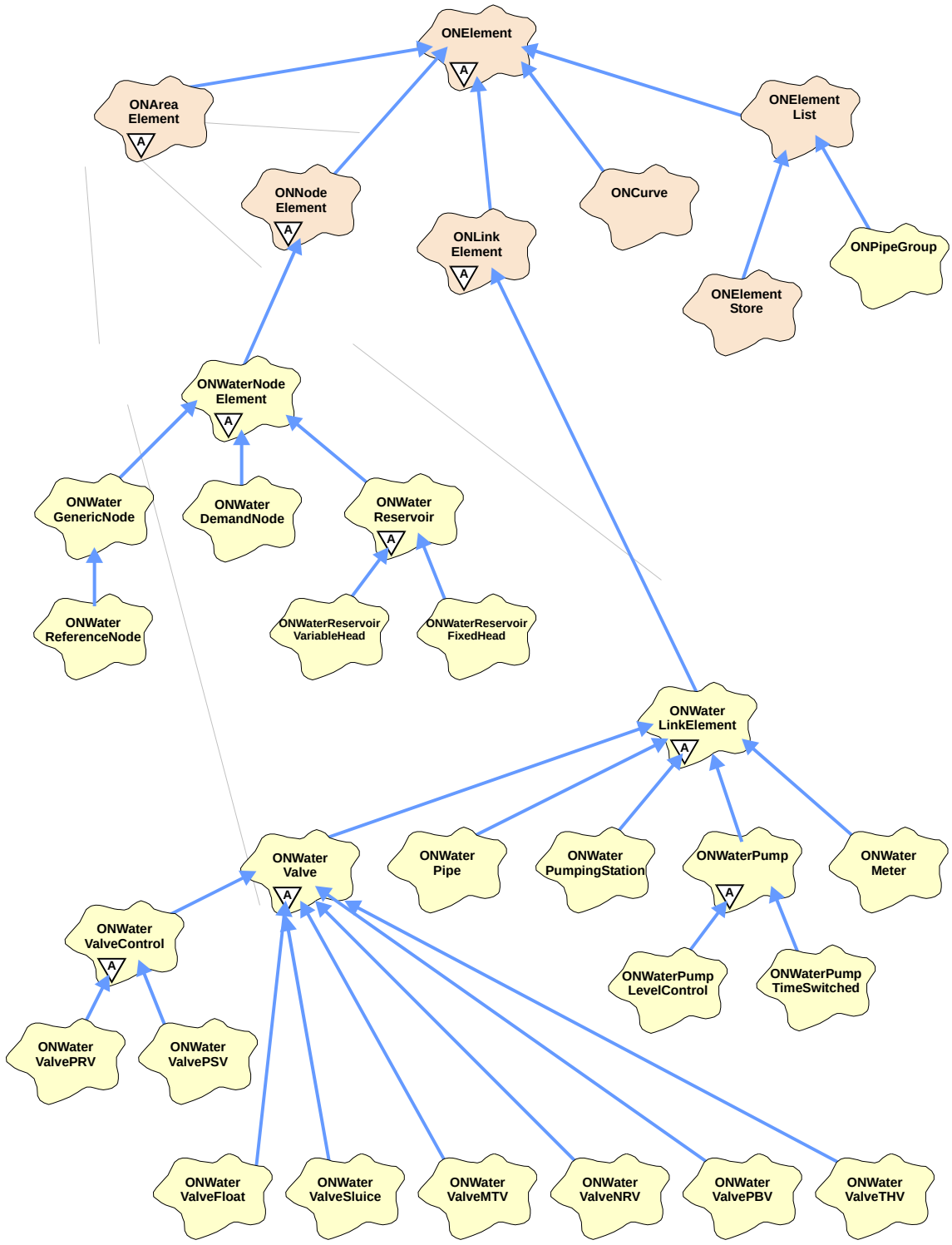

Figure A-3: OpenNet class hierarchy (partial)

A.3 Network Constituents

Each network is comprised of a number of collections of elements that represent the physical infrastructure of the network. These basic elements are wholly abstract and can be extended to implement any form of network desired - indeed, classes derived from this base have been used to represent fracture patterns in impermeable rock masses and for use in the simulation of Air Traffic Control patterns. In this form, they are generalised and need to be derived from before any useful functionality is achieved.

A.3.1 Elements

Elements are the fundamental units of the object hierarchy that implements the network representation. The *ONElement* concept is abstract - though it is possible to instantiate objects of this class. Each of the hydraulic components of the network ultimately derives from this class. However, this and the other basic classes are not specialised in any fashion and can consequently be used for any network representation that has a node/link/area configuration.

The basic characteristics of an *ONElement* include:

- Unique identifier – a common prerequisite for efficient data handling is the provision of a unique key to reference an individual.
- Description
- Element status – whether an element is available (open) or not (closed).
- Assignment
- XML and stream handling

Foremost amongst these is the ability to assign *ONElement* derivatives to each other. This is analogous to overriding the assignment (=) operator in C++ (which, in the C++ version of the library is also used). The assign method copies the contents of a specified *ONElement* object into the object, which calls the method. For example, the statement:

```
anElement.assign(anotherElement);
```

copies the contents of anotherElement into anElement. The assign method implemented by *ONElement* merely calls the method assignTo on the specified object thus:

```
anotherElement.assignTo(anElement);
```

The *assignTo* method propagates up through the class hierarchy copying across each data member that can be copied by checking the class type of the object to receive the data members.

In this way, it is possible to ensure that the objects copy only the data members that they have in common. Ordinarily it is only necessary for a derived class to supply a new overridden *assignTo* method when a derived class introduces a new data member. The assign method may be overridden if special processing is required after the data members are copied. However, the widespread use of properties within the library (or correctly defined "getter" and "setter" methods in the C++ version) makes this facility largely obsolete since properties can be used to centralise 'intelligent' processing of this sort.

Closely associated with the object assignment facilities are those of object stream handling. Streams are a concept familiar to C++ programmers, though there is no implicit analogue present in the ObjectPascal language. Streams are used to store sequences of objects - be it in memory, or more commonly to disk or other storage medium. A disk stream is implemented as a conventional binary file. An object is written to a stream, using the write method, by first committing a unique identifier to the stream - in this case the object class name - followed by the data members that make up the class.

Resurrecting an object from the stream is performed by the read method, although this is made more difficult without specific language support. Fortunately, ObjectPascal provides the ability to construct programmatically an object from a class name. Conventionally you would create an object thus:

```
anObject= new aClass();
```

The alternative representation allows the construction from a String variable using the class function GetClassType inherited from *TObject*, the ultimate ancestor of all objects in C++Builder and Delphi:

```
aString= "aClass";
anObject= new (GetClassType(aString))();
```

Both read and write methods are to be overridden in descendent classes wherever a new data member is introduced. It is vital, during development, however to ensure that the order of reading and writing the objects to the stream is not compromised. Both methods operate recursively across *ONElement* derivatives, thus it is possible to save an entire OpenNet network configuration simply by executing the write method on an *ONNetwork*

object. The same basic mechanism is used to implement the writing and reading of objects to XML metafiles.

The implementation of the C++ version differs slightly in that it is not possible in standard C++ to create an object by supplying its name as a string. Instead, a generic *Create* method is provided in the C++ version and must be overridden by any descendents to return a new instance of that class type. Since OpenNet classes are centrally registered in C++ applications they can be associated with an identifier which can be written to streams and examined when recreating objects.

A.3.2 Element Lists and Stores

Element lists and stores are specialised container classes for storing instances of elements. They differ in that *ONElementList* stores only references to the objects whilst *ONElementStore* contains the objects themselves and is responsible for their safe destruction.

Both classes are implemented using C++ Standard Template Library (STL) underpinnings and there are two variants of each – optimized for small-scale and large-scale networks respectively:

- The first representation uses the STL vector class. The vector class is the functional equivalent of an array – allowing rapid, random access to the contents both by iteration and through the index number of a given element. This representation is better suited to smaller networks – minimizing the scale of the vector resizing necessary as well as the scope of any sorting operations that take place over the lifetime of the vector.
- Based on the STL *map* class, the second, and preferred, representation is implemented as red/black binary tree structure in a similar fashion to the caching data structure discussed in Chapter 4. The *map* implementation facilitates uncomplicated access to the contents through the use of iterators. Access through indices is, however, computationally intensive as the *map* obliges the use of an iterator to traverse the tree to the required index. For this reason, the algorithms that use this representation need a little more forethought as to their design.

A.3.2.1 Addition

In the case of the vector representation of a list or store the addition operation pushes the new element to the end of the vector – necessitating the extension of the contiguous memory

required by the array. This expansion is potentially time-consuming and is to be avoided if possible. To this end, the length of the vector can be set manually at the beginning of a series of addition operations to prevent a sequence of expansion operations. Following the addition, the contents of the vector must be sorted to ensure that the vector remains in element ID order.

The advantages of the map representation for large-scale networks are clear when it comes to implementing the addition operation. When adding an element to the map, the map is traversed and the element inserted at the appropriate point, without the requirement of an additional sorting procedure. Because the data structure is implemented as a number of discrete data items, rather than an array, there is none of the overhead of the array management as seen with the vector representation other than those related to allocating memory for the pointer to the new element, which is common to both representations.

A.3.2.2 Deletion

Removing elements from either data structure is straightforward. Again, however, the efficiency of the map representation over that of the vector is apparent when considering large networks. Deleting the element from the vector involves remapping the remainder of the vector to begin at the point of the deleted element – or, alternatively, by maintaining a list of the "deleted" elements so that they may be ignored when iterating over the structure. If the underlying implementation of a vector were that of a linked list, this remapping can be made with minimal disruption. However, as the implementation is as an array, the remapping requires the copying of the remainder of the vector in memory.

A.3.2.3 Searching

The *ONElementList* and *ONElementStore* classes introduce support for retrieving their contents in various fashions. The *index* method returns the array index of a given object or element ID. This function works with both representations – despite index access being prohibitively time-consuming under the map representation. This allows the use of source code using either representation to run unchanged if the underlying representation is changed. Similarly, the find method returns an object with a given element ID or array index. An array property, Element, allows random access to any element in the list. This property is supported by both map and vector implementations although for the map implementation this function is expensive in performance terms.

For optimal performance when searching, the map based *ONElementList* class implements tree balancing, triggered manually, which traverses the map and attempts to construct tree branches to approximately the same depth – minimizing the average search time for elements in the map.

The time complexity for searching for a given element among *n* items in the vector representation is O(*n)* whilst for the map representation it is O(log *n*).

A.3.2.4 Other functionality

The assignment functions of the *ONElement* class are also represented in the *OnElementList* and *ONElementStore* classes, as would be expected as they are derived from the basic *ONElement* class.

Assignment for classes that contain other objects is, however, more complicated, as there are two possible interpretations of the assignment:

- A shallow copied object – equivalent to a binary copy of the original object.
- "Deep" copying, by contrast, ensures that the resulting object is fully independent of the original object. It achieves this by creating new, equivalent objects of those within the container.

By default, the List class implements shallow copying whilst the Store class, since it is intended to maintain ownership of any contained objects, implements deep level copying.

A.3.3 Network

In order to maintain the referential integrity of the connectivity information in the network, the *ONNetwork* object is responsible for managing all additions and deletions to/from the network. To achieve this, the *ONNetwork* class uses three *ONElementStore* objects to contain all of the objects that fall into the categories of Nodes, Links and Areas. Further specialization is possible by using ONElementList objects to maintain lists of specific types of Node, Link or Area – as seen in the hydraulic specialization of OpenNet.

A.3.4 Node Elements

ONNodeElement objects are the points on a network – representing demand points on a water network, junctions, substations or other such elements. Each node maintains a list of all the links that are connected to it as well as a list of other elements that refer to it. This is useful for determining relationships between nodes and other objects, which are not strictly defined by connectivity – such as a valve pressure-setting point.

Creating a node element entails specifying the geographic location of that node as either a two or three-dimensional Cartesian coordinate. If a node is removed from the system then all its associated links and references, such as valve controls, are also removed from the system or disabled automatically.

A.3.5 Link Elements

ONLinkElement link elements represent the linear constituents of the network. They may represent water pipes, telephone lines, electricity transmission wires or other elements that connect nodes. These elements have a "from" node and a "to" node along with, an optional, series of points which represent waypoints along the link. Waypoints can be expressed as either two or three-dimensional coordinates. Any two-dimensional waypoints in the link are resolved to their three dimensional equivalents by interpolating between points with known elevations (if any).

In order for a link (*ONLinkElement*) to be added to the network, it must supply the identities of two node objects (*ONNodeElement*) that it links. When a link is added to the network, it requests that these "from" and "to" nodes add it to the connection lists they themselves maintain. Adding a link to a network ordinarily requires the specification of the "from" and "to" nodes that the link is to connect – thus requiring that node elements be created before link elements. However, in some circumstances, this is inappropriate and this linking can be deferred by supplying the unique IDs of the "from" and "to" nodes instead. The linking process can then take place following the construction of the rest of the network. This deferment is particularly important when dealing with translation from other network representations – particularly those stored in text files where it would be highly inefficient to retrieve data from different locations within the source file. Deleting a link element from a network removes the references from its "from" and "to" nodes.

A.3.6 Element Type Registration

Each element class is registered with the OpenNet system so that related libraries can determine attributes of the element. The registration process can also optionally nominate a user-interface component, derived from a specialised dialog box class - to be used to edit the properties of the element. Figure A-4 illustrates the edit for a pipe object. Discrete panels on the editor are accessible by using the tabs at the top of the dialog. Derived classes can introduce further tabs to add custom extensions to the dialog if necessary.

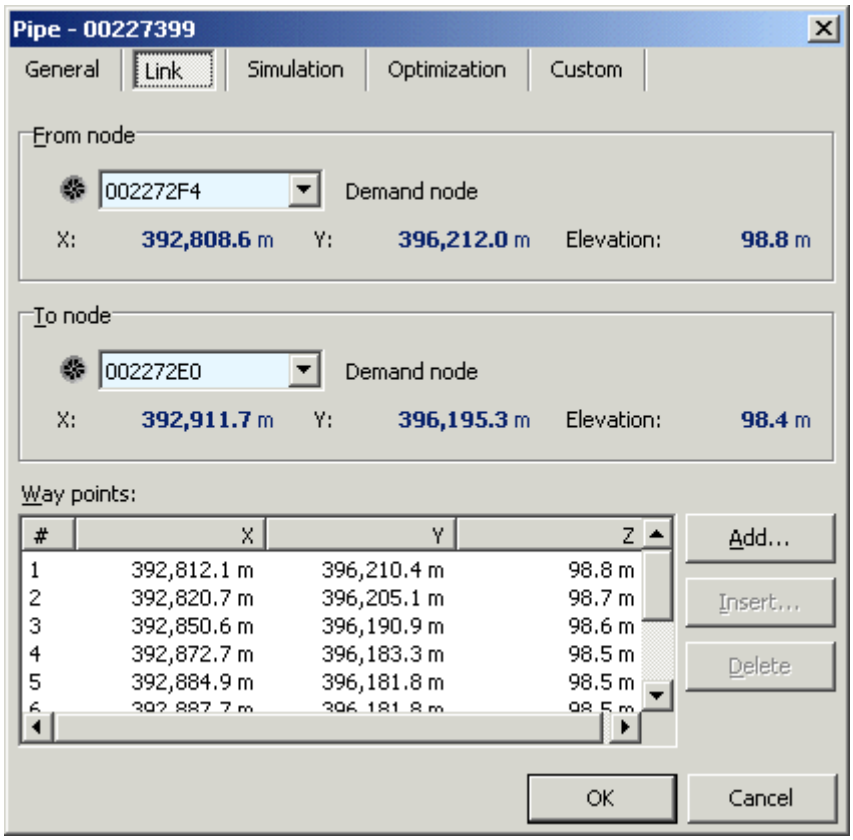

Figure A-4: OpenNet pipe properties dialog box

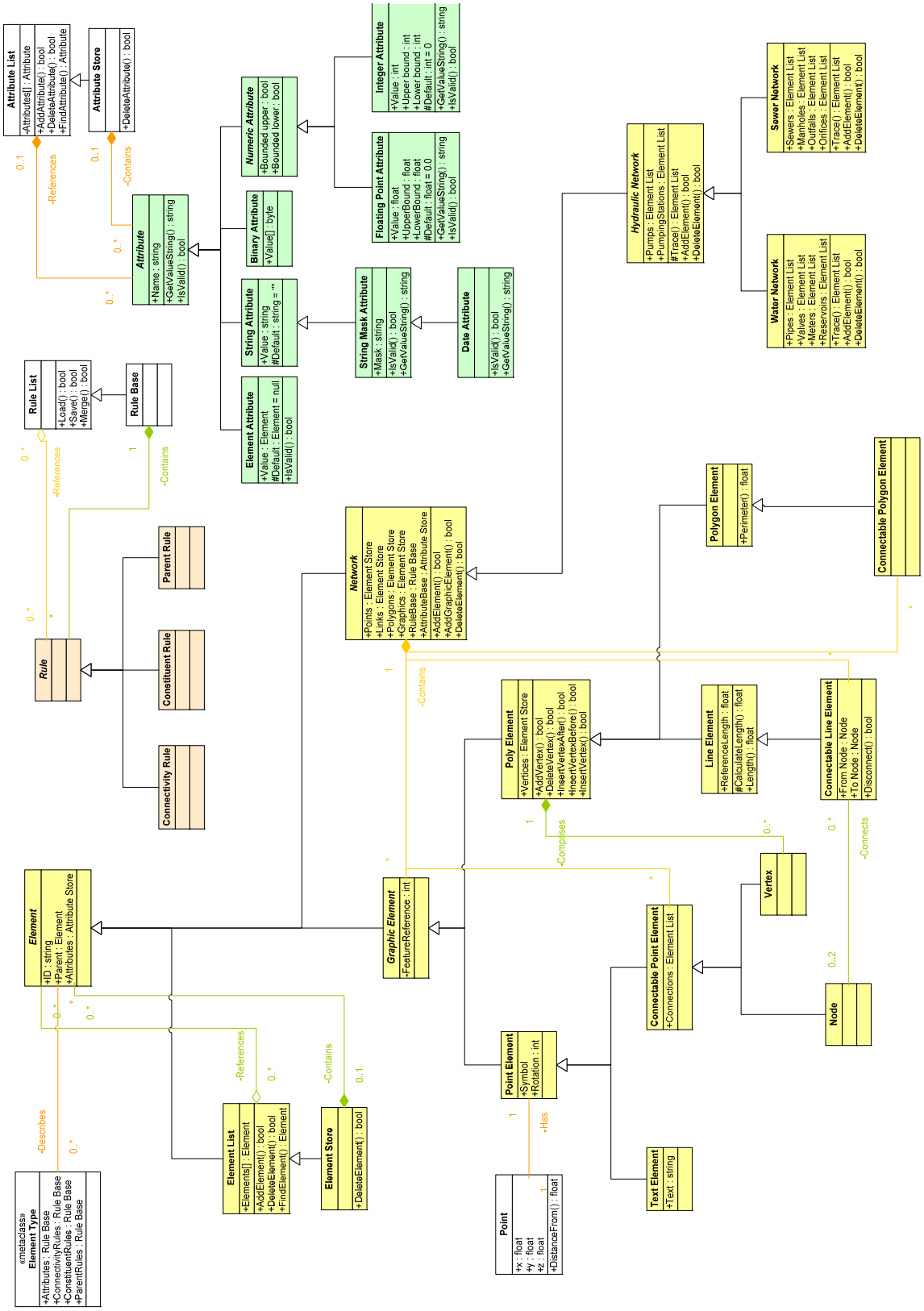

Figure A-5: High level class hierarchy of OpenNet implementation

A.4 Hydraulic specialisation

From the outset, the hydraulic network-model has been designed to interface with GIS implementations and relational databases as well as specific hydraulic solver implementations. Independence from the hydraulic solver is achieved through an abstract *ONNetworkSolver* object from which all solver implementations are derived.

The various commercially available hydraulic modelling packages elect to use quite distinct strategies to model specialist hydraulic components of the network such as valves, pumping stations etc. OpenNet was originally conceived as an abstraction layer for an EPANET solver and accordingly, it adopts many of the representations and conventions used by that hydraulic solver.

Generic nodes

The simplest node class, from which all of the hydraulic nodes are derived, is the generic node. This adds to the basic properties of an *ONNodeElement* object the ability to store nodal hydraulic results, for example, pressure, total demand etc. for a number of time intervals.

Demand nodes

Demand nodes are elements of the network where water is extracted. The concept of demand nodes is somewhat contrived: demand nodes are a geographic aggregation of the real demands made of the system by domestic and industrial users. The demand on the network at a given node can be specified either in absolute terms or as a factor to be applied to a demand curve – of which many may be applied to a single node. OpenNet can accommodate an unlimited number of demand types per node.

Demand curves are used to describe the pattern of water usage for a particular demand type, such as domestic or industrial usage. The example in Figure A-6 illustrates a typical demand curve for domestic consumption – note the peaks around the morning and early evening.

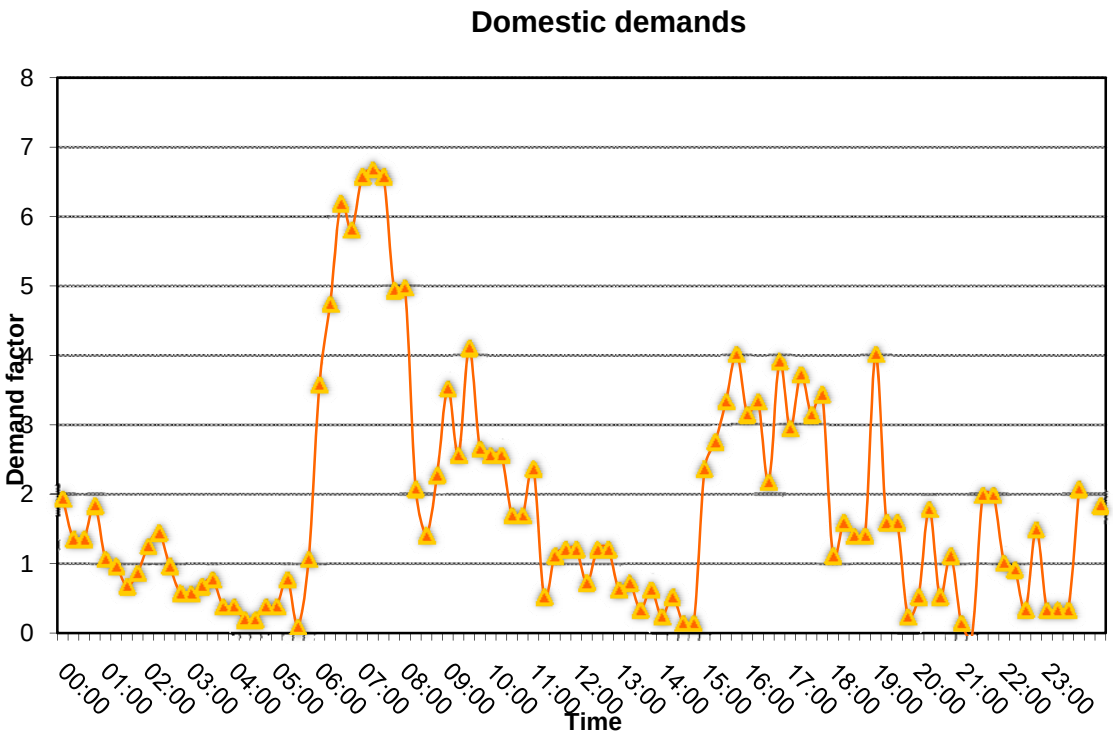

Figure A-6: Typical 24 hour domestic demand curve

Transfer nodes

Transfer nodes are a specialised case of demand node that are used to model outflow and inflow into a network from an adjoining network. OpenNet models inflows by assigning a negative demand to a conventional demand node. This type of nodes is useful for modelling distinct District Metered Areas (DMAs) which might make up a larger water supply area.

Reservoirs

OpenNet models two forms of reservoir object in addition to negative demand nodes.

A fixed-head reservoir provides a constant head irrespective of the amount of water drawn down from it. The head can be varied through the lifetime of the hydraulic simulation by applying a total-head setting curve to an object of this class. This type of reservoir is often used for modelling purposes as an alternative to transfer nodes to represent fixed pressure inflows or outflows from adjacent networks.

A variable-head reservoir behaves differently in that the shape and dimensions of a "tank" are specified allowing the computation of the pressure at the reservoir outlet. As an alternative to specifying dimensions, this type of reservoir is commonly associated with a

Level/Volume curve which defines the reservoir level (and hence outlet pressure) associated with a given volume of water in the reservoir.

Generic Water Links

The base class for links in the water specialisation of OpenNet is the *ONWaterLinkElement*. This class adds the properties that are common to all link elements in water networks, such as the concepts of diameter and hydraulic parameters. This class also introduces data structures for storing the link-related results of a hydraulic simulation, such as flow and headloss, for a number of time-steps.

Pipes

The basic pipe class introduces properties for recording the material, lining and age of a pipe.

Derivatives of this pipe class are used in optimization applications. These descendants include specialized attributes for storing penalty and costing information.

Valves

OpenNet implements a number of valve types. The basic *ONWaterValve* object describes the basic geometry and the hydraulics of the valve.

Other valve types implemented are:

- Float valves (FLV) – used for regulating reservoir levels.
- Throttle Valves (THV)
- Motorised Throttle Valves (MTV)
- Non-Return Valves (NRV)
- Sluice Valves – simple valves that can either be open or closed.
- Control valves – control valves belong to a derived class, which include a reference to a node that is used in some fashion to control the valve setting.
- Pressure Reducing Valve (PRV) – closes to reduce pressure downstream.
- Pressure Sustaining Valve (PSV) – opens to maintain pressure downstream
- Pressure Break Valve (PBV)
- Remote Control Valve (RCV)

Pumps

Pumps are elements that introduce hydraulic pressure into the network. The pumping station class can be used to aggregate pump objects. This is implemented as a link element and all pumps belonging to the station share the same connectivity. Such an abstraction is useful in optimization applications, which can size pumping stations to match minimum-pressure requirements.

Curves

In order to perform a hydraulic simulation it is normally required to introduce some element of time-dependent data to the network, be it nodal demands or reservoir levels etc. Other data used by the hydraulic model is also commonly represented as curves such as level vs. volume curves for variable head reservoirs.

The *ONCurve* class offers facilities for:

- interpolating missing values.
- aggregating curves – including those with different timebases.
- applying multipliers and offsets to timebase (or x value) and y values.

Water Network

The water network class, *ONWaterNetwork*, implements water specific behaviour over and above that of the *ONNetwork*. Nine additional referential lists and their associated management functions are introduced to maintain collections of pipes, generic nodes, demand nodes, valves, pumps, pumping stations, meters, reservoirs and curve data.

A.5 Network analysis

A.5.1 Connectivity

Conventional relational database management systems (RDBMS) do not allow for the easy implementation of databases representing connective structures like networks. Ironically, the outmoded hierarchical database architecture was much better suited to such applications. Riggs (1994) relates the variety of information that needs to be stored in a conventional RDBMS and GIS to model networks on a basic level. By considering the application of some object-oriented techniques, it is possible, however, to implement rules and constraints that are used to define network connectivity in a much more natural fashion.

Meaningful analyses of the network may be achieved with very limited GIS analysis tools. As found previously most commercially available GIS applications have little or no connectivity functionality, let alone network analysis capabilities. Tsakiris & Salahoris (1993) describe a possible vector based representation of a water distribution network. The implementation they suggest is quite simple and does not address connectivity issues - although the authors do note that this is an important concern in any GIS that aspires to be anything other than a straightforward inventory of the infrastructure components.

The venerable Arc/INFO GIS (ESRI, 1999) stands apart from other PC based GIS implementations by offering a connectivity analysis module, which can be used to good effect to develop the data provided by RDBMS to implement rudimentary network analysis functionality. Examples of these are the 'ROUTE' and 'TRACE' procedures from the standard 'NETWORK' module, used by Taher & Labadie (1996) to determine least-cost routing and resource allocation. The background to these networking implementations is covered in Lupien *et al.* (1987) and Djokic & Maidment (1993). The former, working for the Environmental Systems Research Institute - Arc/INFO's publishers - perhaps unsurprisingly, espouse the inclusion of these facilities in GIS systems. They do not enter into a detailed description of the techniques than can be used although a common implemented facility is Dijkstra's algorithm (1959), which is concerned with shortest-path routes with respect to some concept of 'cost'. The NETWORK implementation of Arc/INFO is based on this work and it has been implemented in the abstract network model allowing the use of GIS applications without inherent connectivity functionality such as the low-end MapInfo package.

A.5.2 Network Traversal

The concordant connectivity information maintained within OpenNet's node and link structure allows for the development of powerful algorithms for analysing the network.

A.5.2.1 Basic functions

The OpenNet Network class implements a number of core routines to facilitate traversing the network. Most of the routines are of a recursive nature – such routines operate by calling themselves with new parameters until an end condition is satisfied. The most common illustration of a recursive routine is used to calculate factorials. The non-recursive form of the function is:

```
int factorial(int x)
{
 int Result= 1;
 for (i=2; i<=x; i++)
  Result= Result * i;
 return Result;
}

int factorial(int x)
{
 if (x= 0)
  return 1;
 else
  return x * factorial(x-1);
}
```

One of the considerations of implementing recursive functions is that of stack usage. Every time a procedure, function or object method is called its return address and any parameters are pushed onto the application's memory stack. With large-scale recursion, the depth of these stack calls can become critical as a limited amount of memory is given over to the application stack. Under modern 32 and 64-bit operating systems this limitation is largely irrelevant but remains important under more dated operating systems where a 64 kilobyte limitation on stack memory size was common.

The most important of the basic tree tracing functions is *recurseSubtree*. The operation of this function is illustrated in Figure A-7 and Figure A-8. Given a starting node, *startNode*, and an initial link to traverse, *startLink*, the function performs the following algorithm:

```
procedure recurseSubtree(startNode,startLink)
begin
 if startLink is not Selected then
 begin
  set startLink Selected to true
  set endNode to Node at other end of startLink from startNode
  for each link connected to endNode
   recurseSubtree(endNode,link)
 end
end
```

Two further base functions, *isSubtreeBranch* and *isSubtreeLoop* determine whether a part of a network is purely dendritic (tree-like) or whether it contains loops. This determination is made by starting a recursion from a given point on the network down a specific link – if the sub-tree is purely dendritic then the recursion will never encounter a link that it has visited before – otherwise the sub-tree contains one or more loops.

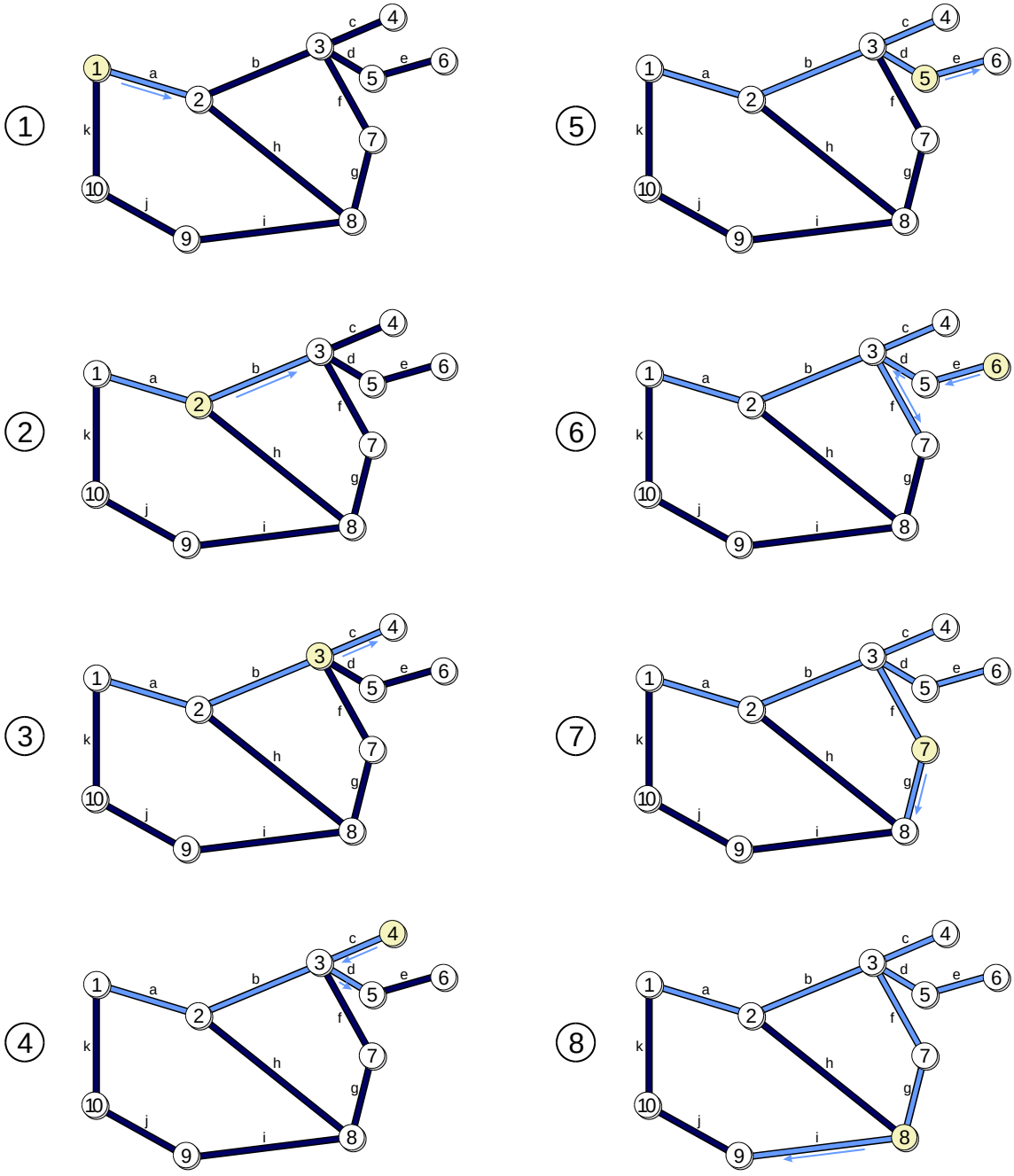

Figure A-7: Recursive network traversal

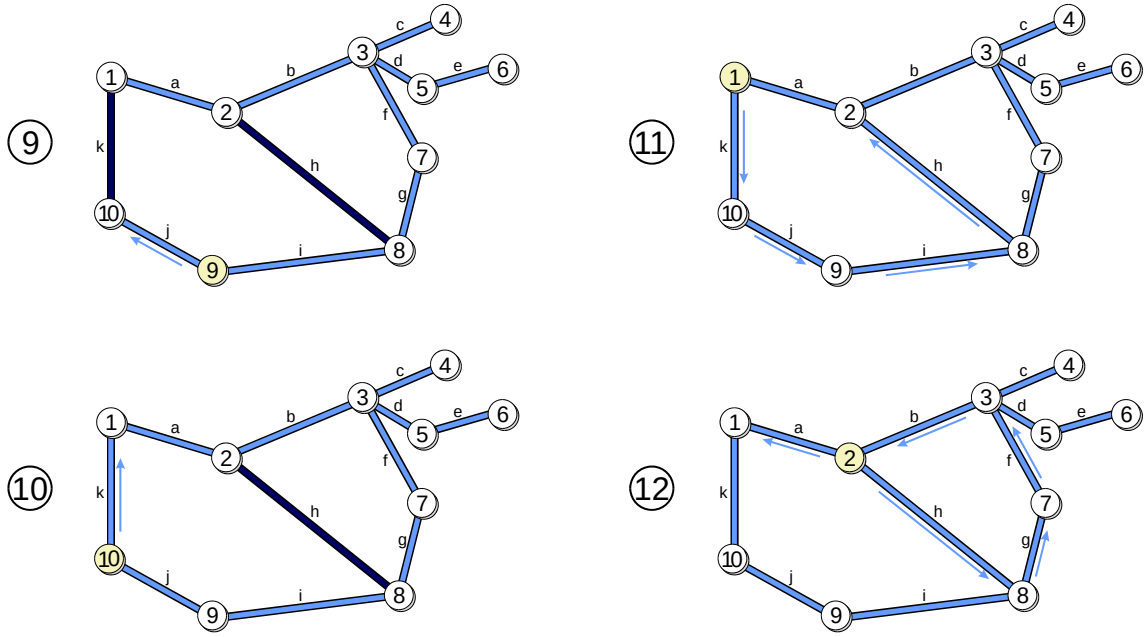

Figure A-8: Recursive network traversal (continued)

A.5.2.2 *Network Simplification*

To assist in the automated construction of efficient genomes for evolution algorithms a number of network simplification routines have also been developed. These are particularly relevant to network calibration applications. These routines include the grouping of pipes to user-specified criteria. These groupings can be used to reduce the length of the genome and thus increase algorithm performance.

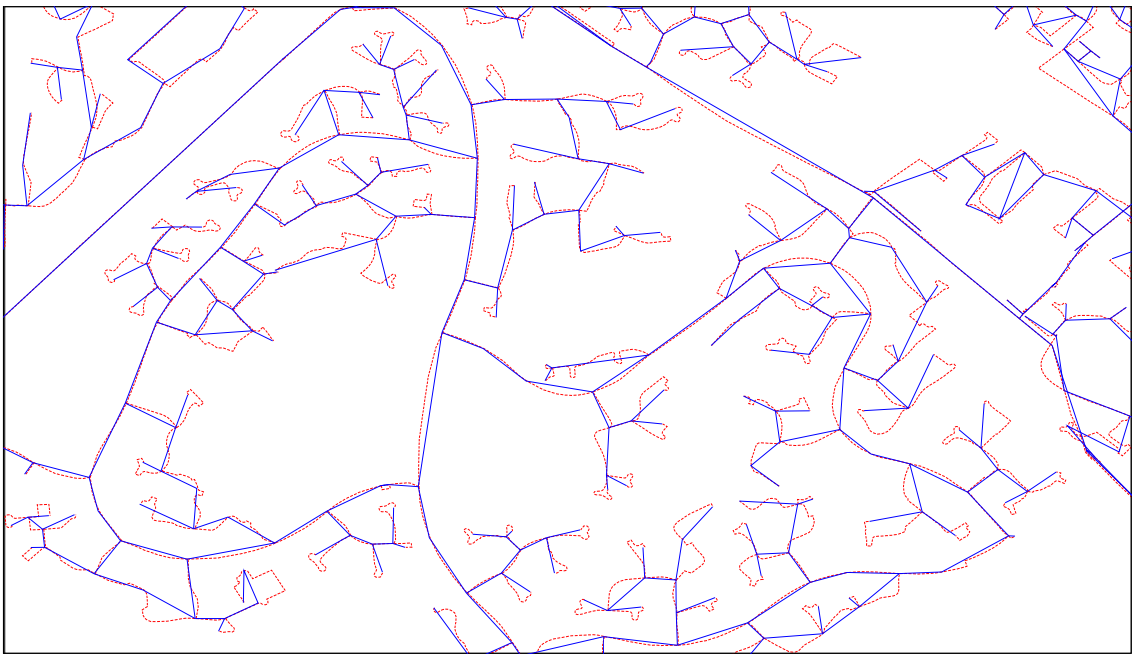

Figure A-9: Network schematic simplification with OpenNet

The automated pruning of selected loops or dendritic structures that play no part in a calibrated solution realises a similar objective. It should be noted that these routines do not modify the components used in the hydraulic solution, merely the information on which the evolution algorithm operates. Figure A-9 illustrates one of these applications in which unnecessary intermediate nodes have been removed from the hydraulic network model after translating it from a GIS-sourced representation.

A.6 Hydraulic Evaluation

Foremost amongst the aims of modelling the infrastructure of a hydraulic network is the evaluation of its hydraulic performance. This involves determining the pressure at each node of the network and the flows in the intermediate pipes – often for multiple time steps (Extended Period Simulation – EPS). The specialised OpenNet hydraulic classes establish a connection to the EPANET2 hydraulic solver (Rossman, 2000) for the purposes of providing hydraulic solutions. The interface to the solver is abstracted so that alternative solvers may be substituted if available. Direct control over the hydraulic solver is offered through the abstraction, including the ability to pause after intermediate time-steps to retrieve results from the network – crucial when undertaking an optimization that operates in EPS mode.

A.6.1 Pressure Driven Demand

Predominantly, Demand-Driven hydraulic simulators such as EPANET used in optimization processes are configured to deliver water even when there is insufficient pressure to do so – Demand-Driven network solver (as in EPANET – Rossman, 2000). In the analysis of structurally inadequate systems, however, studies (Germanopoulos, 1985; Fujiwara & Li, 1998; Ang & Jowitt, 2006), have highlighted limitations related to the use of such demand-driven solvers.

A PDD extension for EPANET has been developed (Morley & Tricarico, 2008) in order to be able to determine more accurately the non-revenue water unsupplied in a pressure-deficient network in order to better estimate a network's Economic Level of Reliability (Tricarico *et al.*, 2006). A logical extension of that work required that the PDD simulator should also be able to operate in an EPS mode. The EPANETpdd extension has been derived from two existing modifications to the core EPANET library: OOTEN (Object Oriented Toolkit for EPANET) (van Zyl *et al.,* 2003), provided by the University of Johannesburg and a revised PDD version of EPANET obtained from its author, Lewis

Rossman. The functionality of the EPANETpdd code has been further extended to incorporate Extended Period Simulation – something that neither the OOTEN nor revised EPANET algorithms facilitated. This is accommodated through the dynamic computation of demand ahead of each timestep in OpenNet. The nodal demand is then converted into an appropriate emitter coefficient and applied to the node in EPANET accordingly and transparently.

A.7 Extensions

A.7.1 Tracing

A simple hydraulic-flow tracing algorithm is integrated into OpenNet. This module allows an individual node to determine the proportion of source waters that it receives via other points on the network. This functionality has been used to determine the probability of upstream contaminants reaching distal nodes.

A.7.2 Generalized Attributes

OpenNet implements a scheme of generalized attributes that can be used for representing user-defined data within the network.

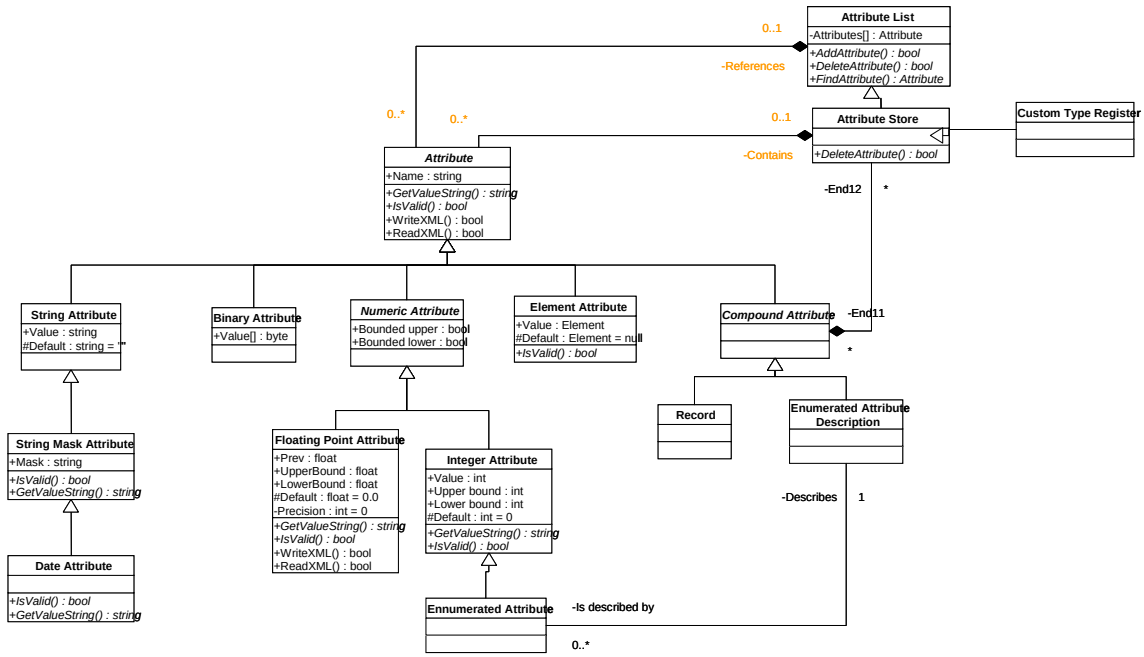

Figure A-10: UML Class Hierarchy for generalized attributes

Whilst other modules also include this functionality, it is put to most use in the network modelling component as it can be used to store data obtained from other data

sources, such as GIS tables directly with the network elements that they refer to. For example, pipe-burst data obtained in a spatial form from a GIS dataset may be directly attached to the pipes in the hydraulic model that they relate to – allowing the development of analysis techniques that operate on the hydraulic model that are then able to consider this data. Figure A-10 illustrates the hierarchy of customized attributes that may be attached to any network element.

A.8 Network model translation

Hydraulic modelling software used within the water industry is dominated by the products of a small number of commercial vendors. Writing software to directly interface with these applications is often either difficult - for example no publicly available interface specification – or impossible – where there is no interface at all. Without the intervention of the commercial entities developing the software, it is therefore necessary to operate on the raw data that can be exported from these applications. Most if not all of the software in this market allows the topology and operating characteristics of the network to be exported in some form, often an ASCII text file. A number of the packages, SynerGee (Advantica, 2003) and InfoWorks (Wallingford Software, 2005) natively store their data in the form of an Access database that can be operated upon by third-party applications using a standard ODBC driver.

A.8.1 Translation suite

To promote the ease of using disparate hydraulic modelling systems with third party applications developed with the optimization methodologies presented in this thesis, a suite of translator utilities has been developed which permit the exchange of network-model information between a variety of third-party hydraulic modelling software using OpenNet as an intermediate, representation-independent format for describing the network. In addition to the translation suite, OpenNet also has the ability to store and load models that it has imported in its own native XML representation – thus once an imported model has been validated satisfactorily, the resulting model can be stored in the native XML format for reuse. Figure A-11 illustrates the import and export translators that are available to OpenNet at the time of writing:

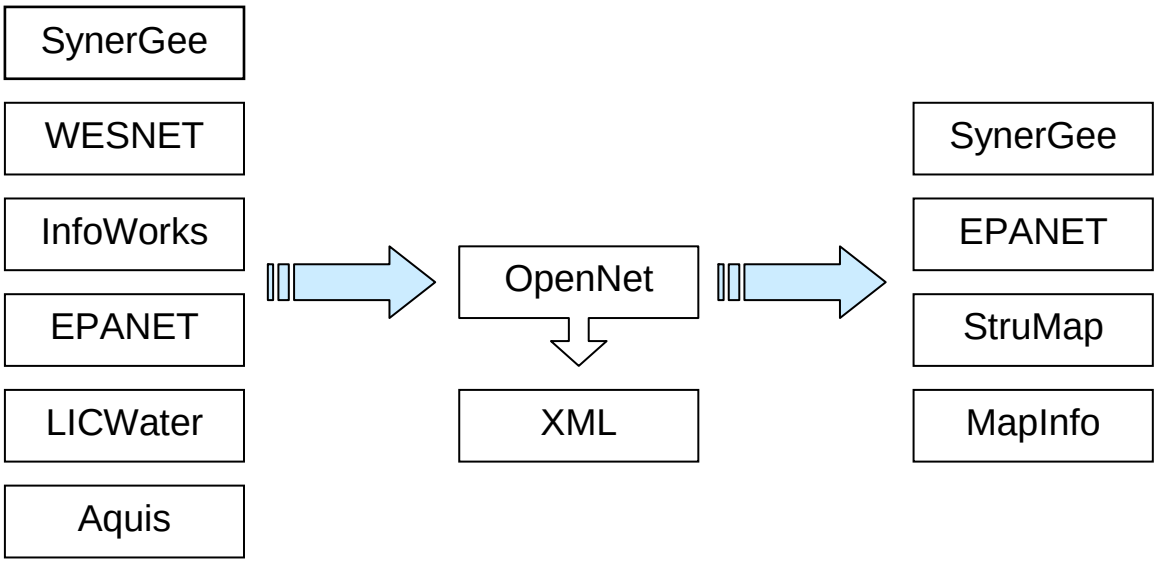

Figure A-11: Translation options available through OpenNet

EPANET (Rossman, 2000) has emerged as a de-facto standard amongst the research community – partly because of its public domain licensing but also because the source-code is freely available and can be easily modified to extend the functionality of the software. It is perhaps unsurprising, then, that many of the conventions used within EPANET are reflected in the underlying structure of OpenNet. The widespread use of EPANET in research is also the reason for this being one of the few packages supported with both input and output translators.

A.8.2 Translator structure

OpenNet provides and abstracted translator class *ONTranslator* which provides a basic framework for implementing translators – including robust file-handling for ASCII files. Two subsidiary classes *ONTranslatorImport* and *ONTranslatorExport* are provided which handle, respectively, loading a network whilst ensuring a concordant OpenNet representation and saving a network. The individual translator implementations then need only provide functions to import/export the individual elements from/to the file.

A.8.3 User interface support

Owing to the fact that there is a degree of uncertainty involved in the reliability of the translation process, the OpenNet library provides interactive feedback on the process of the translation for the end-user. This dialog, an example of which is shown in Figure A-12, alerts the user to any incompatibilities or uncertainties, which may have been encountered in the translation process for the purposes of manually fixing-up the model at a later stage.

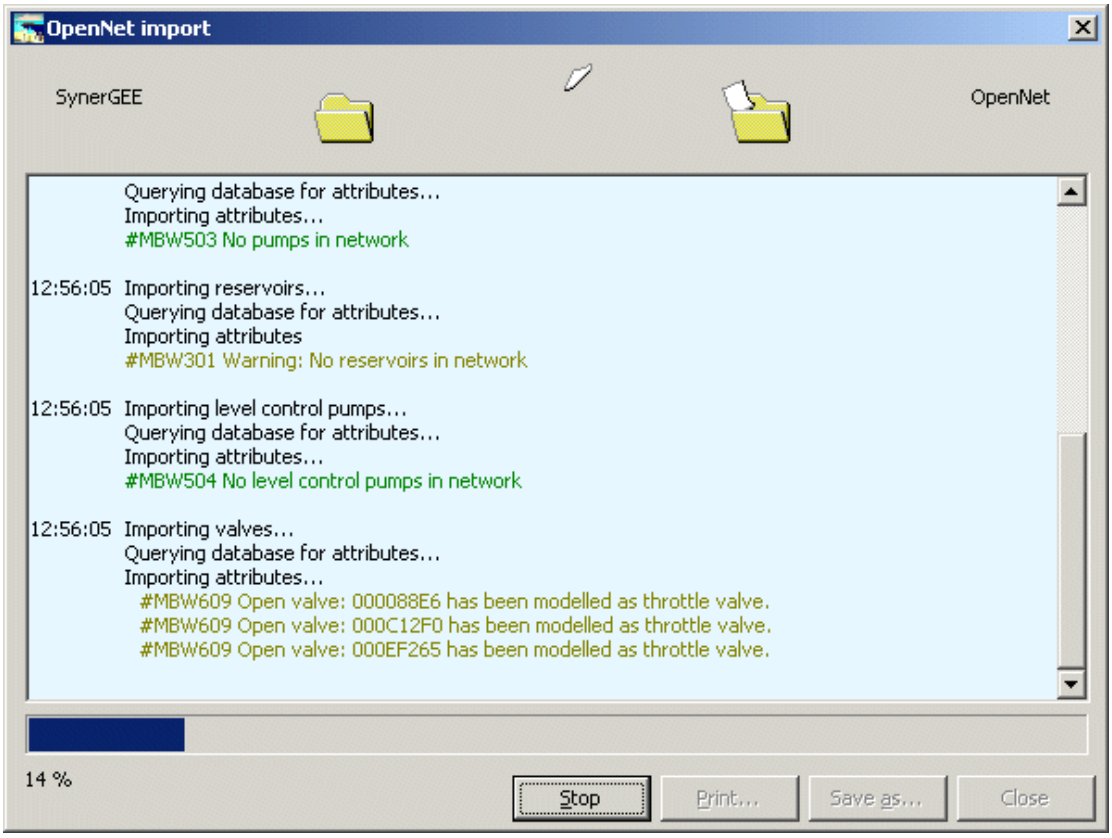

Figure A-12: Translator progress window showing a SynerGEE model being imported into OpenNet.

A.8.4 Difficulties

Table A-1 illustrates one of the major issues that has to be accommodated in handling translations from one hydraulic modelling package to another: the varying representations of basic infrastructure elements.

Element	OpenNet	SynerGee	WesNet	StruMap	EPANET	Aquis
Valve	Link	Link	Node	Node	Link	Link
Pump	Link	Link	Link	Node	Link	Link
Reservoir	Node	Link	Node	Node	Node	Node
Pumping Sta.	Link	n/a	Link	n/a	n/a	n/a
Meter	Link	Link	Node	Node	Link	n/a

Table A-1: Differences in network element representation between common hydraulic modelling packages

Not only do the different modelling packages have different representations of common elements (for example representing a valve as a link element rather than a node) but also in the specification of the more complex network devices. For example, SynerGee (*née Stoner*) offers an unrivalled number of ways to specify the performance of a pump. The majority of these specifications are unavailable in any of the other packages – making such a

translation dependent on the end-user applying expert judgement to the translated model to obtain an appropriate translation.

Accounting for differing node/link representations is rather more straightforward with the base *ONTranslator* class providing functions to generate automatically dummy nodes and links, as appropriate, to maintain the correct representation and to ensure that a consistent hydraulic performance is retained. Information regarding these dummy nodes is nominally stored in any exported models so that, if encountered again by the OpenNet importer, the dummy elements can be safely removed from the model. Otherwise, repeated use of the OpenNet translator system moving from one modelling scheme to another would lead to ever increasingly complex models.

Other problematic elements include "remote controlled valves" where the valve setting is controlled by the pressure state at a node elsewhere in the network. Whilst all modelling packages provide PRV (Pressure Reducing Valves) and PSV (Pressure Sustaining Valves) which operate on a point immediately downstream of the valve, few of them allow this control point to be elsewhere in the network.

A.9 Linking hydraulic models to GIS applications

Atkinson *et al.* (1998) illustrate some of the advantages of a close-coupled integration between a GIS and a genetic algorithm solver - including the speed of processing and a common user interface. The GA*net* user interface of Morley *et al.* (2001) takes this integration a step further and integrates the interface of the GIS application into its own. Initially, this was designed to integrate the StruMap GIS application but in addition, the interface can also act as an OLE container, or client, for OLE automation servers. One such server is the GIS application, MapInfo (MapInfo Corporation, 1998) which can also be instantiated as a COM object. The modular nature of the design allows it to be comprehensively extended without the need for recompilation – making it ideal for distributing as an end-user product. Functionality can be extended using "plug-ins", as can the definitions of the genetic algorithms themselves.

The algorithm control window is extensible and can be modified by any organism to show application specific information – again without recompilation. The simplest mechanism for linking GIS information into a GA application is by reading the data in through some common file format. It is often desirable, however, for the geographic information to remain in the GIS in order to make geographic queries against it. Both of the

approaches taken by OpenNet and GA*net* allow this, the latter effectively acting as a plug-in to the StruMap GIS. Integrating the GIS as a COM object provides a unique framework with easy and complete manipulation of the data in the GIS coupled with the complete integration and customization of the user interface. Through the use of Microsoft's COM technology it is possible to embed application objects within other applications. Both MapInfo and Arc/INFO have COM or OLE2 variants that can be used by third-party developers. The use of both these technologies affords a hitherto unavailable level of integration for applications. Future development of Distributed COM objects or CORBA communications raises the potential for offloading the GIS querying functionality to a dedicated server to improve performance, along with the possibility of using more powerful workstations to perform the hydraulic network solutions.

Water companies often have two sources of data with respect to their pipeline networks. As well as hydraulic models of their networks, they will commonly use a GIS-based system for asset management purposes.

The hydraulic model is often an idealized, simplified version of the all-mains model contained within the asset management system. Simplification of the network model can improve the performance of hydraulic evaluation through having fewer pipes and nodes to evaluate. For example, one simplification commonly performed is the aggregation of contiguous pipe assets with identical characteristics such as diameter and age and the sharing of intermediate nodal demands between the end nodes of the aggregated pipe.

Given that hydraulic models rarely contain data other than that directly associated with the hydraulic performance of the network it is often necessary in optimization applications to have access to other data elements such as pipe age, material, burst history etc. This information will usually be contained within the asset management system. Relating these two sources of data is often time-consuming and difficult – not least because individual water companies maintain their own policies with regard to the form that the data is stored and, indeed, *what* data is retained.

Owing to the fact that the development of hydraulic models is often a completely segregated process from the maintenance of the asset management system, it is common for there to be no direct means for associating records from the asset management system with their counterparts in the hydraulic model. Asset management records should have a unique identifier for the individual pipes across an entire Water Supply Zone. Elements in a hydraulic model are much less likely to maintain unique identifiers even if the model has been

derived directly from the asset management data owing to the simplification processes employed in generating the hydraulic model.

As part of a research contract with a UK Water Company, the candidate devised a system for attempting to produce a concordant data set, which could be used by analytical processes including optimization applications. Identifying co-located geographic features in the two datasets was the principal technique used in this analysis – allowing for the direct matching of pipes from one model to another. Pipes that remain unmatched after a geographic analysis are then permuted in aggregations with their neighbours, pipe grouping, and then resubmitted to the matching process.

A.9.1 Pipe matching

In performing the pipe matching, it is assumed that both networks contain some form of geographic referencing for the data elements. The matching process operates as follows:

A.9.1.1 Import Hydraulic Network

As a preliminary step, the hydraulic model is loaded into the OpenNet generic network modelling through one of a number of translators described earlier in this appendix. The translated network is output to a MapInfo-compatible dataset.

Early test networks imported using this procedure highlighted a number of issues with the geographic information associated with the source data. In some cases, the coordinate system applied to the model appeared to be entirely arbitrary: geographic information is not necessary for the successful operation of a hydraulic model where only the elevations of network nodes and lengths of pipes need to be accurately described. Other networks illustrated significant displacements or scaling issues, which necessitated modifications to the OpenNet import translators to apply user-defined offsets and scaling factors. Other hydraulic models required rotational corrections as they had been recorded in either magnetic north or true north orientations whereas the Asset Management System used OSGB (Ordnance Survey Great Britain) grid north. One network, which had been composited from two separate District Metered Areas (DMAs) managed to contain both orientation errors in the different sections of the network.

A.9.1.2 Import Asset Management Database

The asset management GIS can be translated from an ArcInfo-compatible dataset or opened natively from a MapInfo compatible or SpatialWare-wrapped dataset. Owing to the size of

such databases, opening the dataset natively is much preferred as the dataset can remain on a remote server and utilise the remote querying functionality of the server to accelerate performance. However, due to the remote location of the Water Company involved in this research and the security concerns regarding access to their asset management system, a full copy of the database was provided to the candidate on DVD-ROM.

Once access to the dataset has been established, if a geographic representation of the network is not already available then a new geographic dataset is created in a MapInfo compatible format.

A.9.1.3 ID Matching

Although much of the network model stock held by the Water Company predates their asset management system, some more recent models have been generated directly from this data. Many of the elements in these models are transferred unchanged – although some are aggregated during the transfer process. The unchanged elements retain their unique identifier from the AMS in the hydraulic model and, consequently, as a first step in the matching process, the software determines whether any elements in the hydraulic network match the unique identifiers in the AMS. Due to a constraint in the length of identifier that the hydraulic modelling software used by the Water Company can handle, the 11-digit numeric identifiers used in the AMS were encoded into base 24 in the network model in order to allow them to fit the capacity allotted of 8 ASCII characters.

A.9.1.4 Geographic matching

The geographic matching technique is initially applied to the link elements of a network as they define the unique connectivity and geographic arrangement of the network.

Each link in the hydraulic network is analyzed to determine how well it fits a number of criteria. Firstly, a short-list of candidate links from the AMS is drawn-up by identifying all links in the AMS that fall within a user-defined distance (a "buffer") of the link in question. Then four criteria are analyzed for each of the nodes located within the buffer.

- **End node location.** A geographic "buffer" is erected around each end of the link (with a user-defined diameter). Each link in the shortlist, which falls into one – or preferably both – of the buffer areas, is scored according to how closely the end points match. Experiments have determined that an upper tolerance of 20 metres for the link endpoints is sufficient to accommodate most hydraulic models that have not been derived from the Asset Management System.

- **Link lengths.** The length of the link is compared against each of those in the shortlist. An upper tolerance limit of 20% has been found to work well. The end-user may define a cut-off limit for the pipe length below which this analysis is not performed. This is to prevent very short pipes of circa 15 metres or less being penalised excessively.
- **Pipe centroids. T**he centroids of the curve described by the link elements are compared. This can be considered to being similar to the concept of "a centre of gravity" for a three dimensional form. An upper tolerance of 30 metres has been found to be most appropriate for this option. This technique appears to be more accurate at correctly identifying links that share the same start/end nodes (common in loops) than analyzing the locations of intermediate waypoints along the link.
- **Pipe trends.** Compares the angles on the ground of the intermediate elements of a link – for those defined with one or more intermediate waypoints. A upper tolerance of 35% is considered a good delimiter for correctly identifying pipes.

A confidence score is generated for each of the short-listed elements analyzed, according to how well it meets each of the geographic matching criteria. The shortlisted pipe with the highest overall score is associated with that in the hydraulic model. The weightings for each criterion can be modified by the end-user to suit the type of network under consideration. For example, for large-scale rural networks, the end node locations were found to be a less important discriminator than the overall link length – the reverse of the situation seen in the urban environment with a higher density of nodes.

A.9.1.5 Pipe grouping

Following the geographic matching process outlined above, the algorithm then seeks to improve the solution obtained by aggregating adjacent pipes in the AMS shortlist with similar characteristics and repeating the geographic matching procedure. Aggregates that score higher than previous results will be matched instead. An example of an aggregated, matched pipe-group can be seen highlighted in Figure A-13.

A.9.1.6 Interactive matching

The final stage of the pipe matching process is an optional, interactive matching arrangement where the end-user is presented with the two models in side-by-side windows and can fine-tune the matches that the automated processes have determined as well as associating any remaining unmatched links.

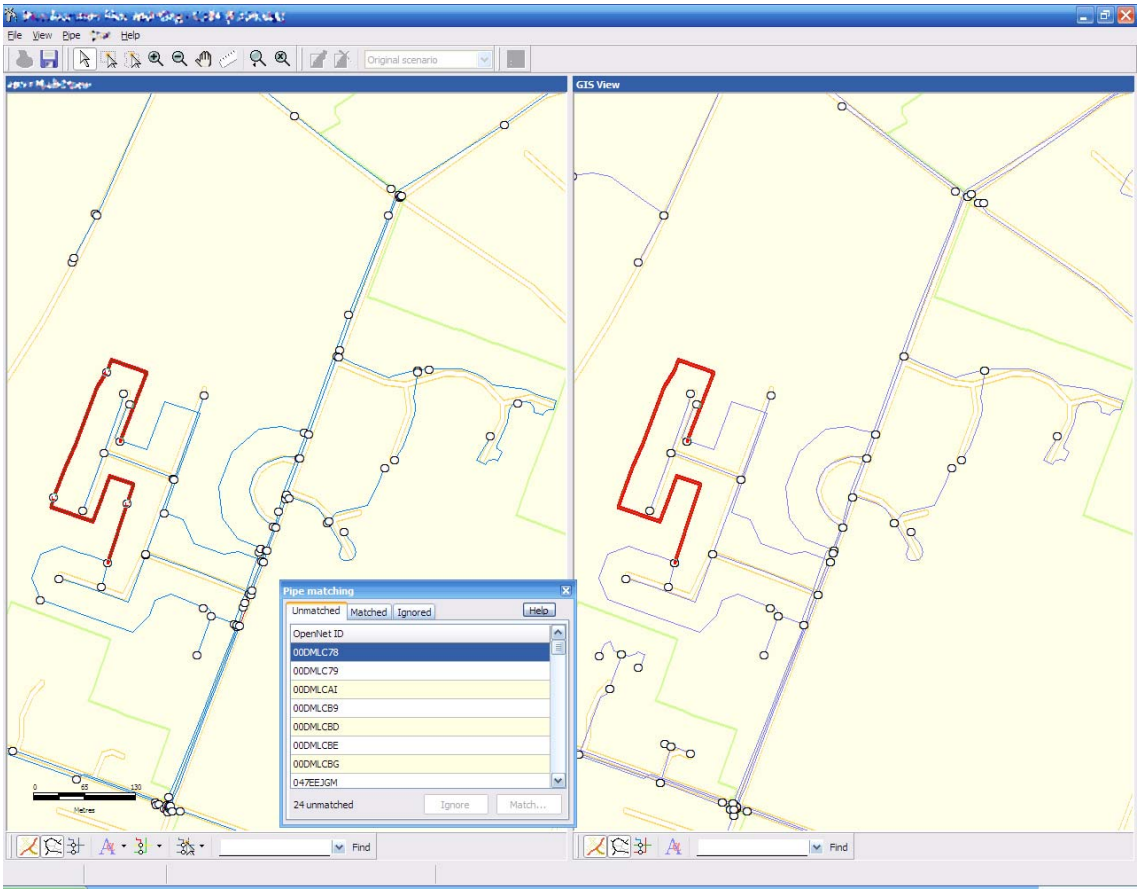

Figure A-13: Pipe matching application

Figure A-13 illustrates the interactive matching mode available to the algorithm. The right hand map pane represents the data extracted from the Asset Management System. That on the left is from the imported hydraulic model. As can be seen, there is a good concordance between the topological arrangement of the pipes in this model as this is an example where the hydraulic model was originally derived directly from the AMS dataset. However, there are, unusually, a number of additional nodes in the hydraulic model that have been inserted for the purposes of attaching demands. Often, the simplification is the reverse with more nodes being present in the AMS dataset that are then removed as part of a simplification process. Highlighted is a single pipe from the AMS data, which corresponds to four separate pipe elements in the hydraulic model. The modeless dialog seen at the bottom of Figure A-13 allows the user to locate any unmatched pipes and to select interactively their analogues from the AMS dataset. This interface display can also be configured to show a thematic map that colours each pipe in accordance with the confidence level ascribed to the match by the automated matching process. This allows the end user to identify quickly any pipes that may have been incorrectly matched by the process.

When the operator is confident in the quality of the matched models, the results can be exported back to the Asset Management System to preserve the link between the two datasets.

A.9.1.7 Results

The accuracy of the pipe-matching software is generally very good and nominally exceeds 95% for GIS derived datasets. The principal problems with this algorithm arise in the attachment of service reservoirs to the network under consideration. Network modellers use non-existent pipes to connect reservoirs to the network as each DMA is usually fed from another DMA or through a large number of pipes. This simplification means that the algorithm cannot match these pipes (since they have no basis in reality) and these anomalies constitute most of the 5% of pipes in a network that cannot be automatically matched.

Once the pipe matching has been completed, it is then possible to use the association found with the AMS to extract pertinent data that can be used in an application as well as providing an accurate geo-referencing which can be used for, amongst other things, for associating leakage complaints etc. directly with the hydraulic model.

For a network comprising around 2,000 pipes, this process is completed in around 15-20 seconds – although this can vary significantly depending on the complexity of the network and the number of potential aggregates encountered by the pipe-grouping algorithm.

A.10 Representing networks using XML

The eXtensible Markup Language (XML) is a subset of the Standard Generalized Markup Language (SGML - ISO, 1986) and has been designed primarily for data description and dissemination over the Internet. Unlike its close relative, HTML, XML allows the use of custom tags to extend its feature-set.

The structure of XML documents is defined through the use of a schema which may reside either in the document itself or in an external file – commonly referenced over the Internet. This Document Type Definition (DTD) is the key to the extensibility of XML. XML documents are normally manipulated using a suite of parser routines, which combine the information present in the DTD and the XML document itself. Consequently, an application that uses an XML document is abstracted from the actual contents and instead communicates with the parser as a broker, which performs such functions as filling in default values where they are not specified in the document.

XML definitions have three main components:

- **Notations**, which are used to describe application-specific, non-XML data (not used in OpenNet).
- **Elements** which represent individual data elements in the document.
- **Entities** which can be used as shorthand for XML markup in any part of a document. The following sections describe the basic structure of an XML document and its related DTD in relation to a network representation.

A.10.1 Elements

A simple example of an XML element is the basic co-ordinate class. A co-ordinate in OpenNet is defined as having an X and Y value along with an optional Z value representing the elevation of the point. The DTD definition of the co-ordinate element is as follows:

```
<!ELEMENT coordinate EMPTY>
 <!ATTLIST coordinate
  x CDATA #REQUIRED
  y CDATA #REQUIRED
  z CDATA "undefined"
 >
```

The first line defines the co-ordinate element as having no children using the *EMPTY* directive. This is followed by a list of attributes that this element can have. The most basic type of attribute is Character Data, *CDATA*, which is a textual data member.

The XML schema makes no attempt to enforce data type conventions so *CDATA* is used for both numeric and textual data, leaving the application developer to ensure that the correct type of data is present in the attribute. The first two attributes, *x* and *y*, are declared using the *#REQUIRED* directive that indicates that these attributes *must* be specified with any co-ordinate element that is created. The *z* attribute is not mandatory and the parser will return the text "undefined" if an alternative value is not specified in the XML document.

Some examples of valid declarations of co-ordinate elements in an XML document are as follows:

```
<coordinate x="5" y="7" z="22.5"/>
<coordinate x="5" y="7"/>
<coordinate x="5" y="7"></coordinate>
```

Elements without children can be terminated with the "/>" tag closure without necessitating the use of a formal closing tag as seen in the third example.

The flexibility of the Element specification scheme becomes clear when children are considered. Every element can contain child elements for which it is possible to specify whether "zero or one", "one", "one or more" or "zero, one or more" element definitions are allowed. In addition, mutually exclusive children can be defined The following example shows how a line element might be formed in the DTD:

```
<!ELEMENT line (coordinate,coordinate*,coordinate)>
```

This definition requires the presence of a co-ordinate for the start and end-points of the line along with "zero, one or more" interior co-ordinates – identified by the "*" suffix. Valid examples of lines in an XML document may look like:

```
<line>
 <coordinate x="744" y="1384" z="11"/>
 <coordinate x="756" y="1322" z="0"/>
</line>

<line>
 <coordinate x="744" y="1384" z="11"/>
 <coordinate x="747" y="1360" z="5"/>
 <coordinate x="756" y="1322" z="0"/>
</line>
```

The construction of such compound elements allows the representation of complex data structures complete with a measure of data validation. An XML document containing a line element, as outlined above, with only one co-ordinate will be rejected as invalid by the XML parser.

Further data-set validation is achieved through the use of *ID* and *IDREF* attributes. These attributes allow relationships between elements to be defined and again, XML documents containing unresolved relationships will be rejected by the parser. This approach is illustrated by the relationship between node and link elements in which each link must specify a valid "from node" and "to node." The basic structure of a node is thus:

```
<!ELEMENT node (coordinate)>
 <!ATTLIST node
  id ID #REQUIRED
 >
```

Using an *ID* attribute and making its presence mandatory requires that every node specified in an OpenNet representation has a unique identifier. In practice, this attribute is included in every OpenNet element to allow for connectivity relationships to be defined between different types of element. The associated link element is defined as follows:

```
<!ELEMENT link (coordinate*)>
 <!ATTLIST link
  from_node IDREF #REQUIRED
  to_node IDREF #REQUIRED
 >
```

from_node and *to_node* are defined as *IDREF* – references to the ID attributes of other elements and are mandatory. The link element also allows any number of co-ordinate children to be specified. These co-ordinates represent the interior points of the line associated with the link (if any), since the start and end points of the line may be obtained from the co-ordinate attached to the *from* and *to* nodes specified.

A.10.2 Entities

The object-oriented nature of the underlying OpenNet class library means that large numbers of properties are shared, through inheritance, by different object classes. Fortunately, the use of XML entities facilitates the inheritance of attributes and child data-structures between elements.

At their simplest, entities are analogous to expansion macros where the entity keyword is replaced by the entity definition wherever it appears in the Document Type Definition. The most straightforward use of entities is where they are simple constructions, used in many places, for instance to implement a friction regime attribute:

```
<!ENTITY % on_true_false "(0|false|no|1|true|yes)">
```

```
<!ENTITY % on_friction "(colebrook-white|
 hazen-williams|darcy-weisbach)">

<!ELEMENT pipe (link)>
 <!ATTLIST pipe
  friction %on_friction; "hazen-williams"
  open %on_true_false;
 >

<!ELEMENT valve (link)>
 <!ATTLIST valve
  friction %on_friction; "hazen-williams"
 >
```

In this example, both the simplified pipe and valve elements share a common friction attribute, which is expanded inline. This representation is not particularly efficient and does not implement the inherited properties in a meaningful fashion. Instead, entities can be nested. The following example, from node elements, shows the different attributes that are added in each level of the class hierarchy.

```
<!ENTITY % on_common_attributes "
 id ID #REQUIRED
 name CDATA #IMPLIED
">

<!ENTITY % on_node_common_attributes "   %on_common_attributes;
 area_id IDREF &#34;&#34;
">

<!ENTITY % on_waternode_common_attributes "
 %on_node_common_attributes;
 calibration %on_true_false; &#34;false&#34;      calibration_curve
IDREF &#34;&#34;
">
```

Each of the entities illustrated above implicitly includes the attributes from the level above. The """ in the above example is used to represent a double-quote character as these cannot be placed directly into entities. Using this mechanism, it is possible to emulate the inheriting characteristics of the object-oriented model. Each element can subscribe to the appropriate inherited attributes in this fashion and add any attributes specific to its requirements thus:

```
<!ELEMENT node (coordinate)>
 <!ATTLIST node
  %on_waternode_common_attributes;
  highest_elevation CDATA ""
  dummy %on_true_false; "false"
 >
```

Consequently, each node element implicitly includes a mandatory *ID* attribute, along with the other common attributes included above. In practice, element children are also inherited in a similar fashion:

```
<!ELEMENT node (%on_waternode_common_children;
 coordinate)>
```

It can be seen from the examples that the types of representation offered by XML documents map appropriately onto the needs of object-oriented databases and class hierarchies. The same cannot be said for the relationship between XML and conventional relational databases which struggle to deal with the feature-rich and variant content that can be implemented in XML documents. The full schema for representing OpenNet networks in XML may be found in Appendix B along with an example network file.

Appendix B OpenNet XML Representation

B.1 XML Schema

```
<!--############################################################-->
<!-- Core entities                                              -->
<!--############################################################-->


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Unit declarations                                          -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ENTITY % on_friction "(colebrook-white|hazen-williams|darcy-
weisbach)">
<!ENTITY % on_area_unit "(square-metres|square-feet)">
<!ENTITY % on_length_unit "(millimetres|metres|inches|feet)">
<!ENTITY % on_true_false "(true|false|0|1)">


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Default declarations                                       -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ENTITY % on_default_length_unit "metres">
<!ENTITY % on_default_elevation_unit "metres">
<!ENTITY % on_default_diameter_unit "millimetres">


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Common child declarations                                  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ENTITY % on_common_children
 "description?"
>

<!ENTITY % on_node_common_children
 "%on_common_children;,
 coordinate"
>

<!ENTITY % on_waternode_common_children
 "%on_node_common_children;,
 calibration_curve?"
>

<!ENTITY % on_link_common_children
 "%on_common_children;,
 link"
>

<!ENTITY % on_waterlink_common_children
 "%on_link_common_children;,
 calibration_curve?"
>
```

```
<!ENTITY % on_watervalve_common_children
 "%on_waterlink_common_children;"
>

<!ENTITY % on_curve_common_children
 "%on_common_children;"
>

<!ENTITY % on_datacurve_common_children
 "%on_curve_common_children;,datastep+"
>

<!ENTITY % on_timecurve_common_children
 "%on_curve_common_children;,timestep+"
>

<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Common attribute declarations                                      -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ENTITY % on_common_attributes "
 id ID #REQUIRED
 name CDATA #IMPLIED
">

<!ENTITY % on_node_common_attributes "
 %on_common_attributes;
">

<!ENTITY % on_waternode_common_attributes "
 %on_node_common_attributes;
 area_id IDREF &#34;&#34;
 calibration %on_true_false; &#34;false&#34;
 calibration_curve IDREF &#34;&#34;
">

<!ENTITY % on_link_common_attributes "
 %on_common_attributes;
 length CDATA &#34;0&#34;
 length_unit %on_length_unit; &#34;%on_default_length_unit;&#34;
">

<!ENTITY % on_waterlink_common_attributes "
 %on_link_common_attributes;
 area_id IDREF &#34;&#34;
 calibration %on_true_false; &#34;false&#34;
 calibration_curve IDREF &#34;&#34;
 diameter CDATA #REQUIRED
 diameter_unit %on_length_unit;
&#34;%on_default_diameter_unit;&#34;
 friction %on_friction; &#34;hazen-williams&#34;
 friction_factor CDATA &#34;0&#34;
 minor_loss CDATA &#34;0&#34;
">

<!ENTITY % on_watervalve_common_attributes "
 %on_waterlink_common_attributes;
">
```

```
<!ENTITY % on_curve_common_attributes "
 %on_common_attributes;
 ybase CDATA &#34;0&#34;
 ybias CDATA &#34;1&#34;
">

<!ENTITY % on_timecurve_common_attributes "
 %on_curve_common_attributes;
 timebase CDATA &#34;0&#34;
 timebias CDATA &#34;1&#34;
">

<!ENTITY % on_datacurve_common_attributes "
 %on_curve_common_attributes;
 xbase CDATA &#34;0&#34;
 xbias CDATA &#34;1&#34;
">


<!--#############################################################-->
<!-- Core elements                                               -->
<!--#############################################################-->

<!ELEMENT description (#PCDATA)>

<!ELEMENT coordinate EMPTY>
 <!ATTLIST coordinate
  x CDATA #REQUIRED
  y CDATA #REQUIRED
  z CDATA #REQUIRED
  xyunit %on_length_unit; "%on_default_length_unit;"
  zunit %on_length_unit; "%on_default_elevation_unit;"
 >

<!ELEMENT line (coordinate,coordinate*,coordinate)>

<!ELEMENT link (coordinate*,line?)>
 <!ATTLIST link
  from_node IDREF #REQUIRED
  to_node IDREF #REQUIRED
 >

<!--#############################################################-->
<!-- Node elements                                               -->
<!--#############################################################-->

<!-- Node group element -->
<!ELEMENT nodes (node|reservoir|fixed-head)*>


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Calibration                                                 -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ELEMENT calibration_curve (%on_timecurve_common_children;)>
 <!ATTLIST calibration_curve %on_timecurve_common_attributes;>


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
```

```
<!-- Generic and demand nodes                                        -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ELEMENT node (%on_waternode_common_children;,demand*)>
 <!ATTLIST node
  %on_waternode_common_attributes;
  highest_elevation CDATA ""
  dummy %on_true_false; "false"
 >

<!ELEMENT demand_curve (%on_timecurve_common_children;)>
 <!ATTLIST demand_curve %on_timecurve_common_attributes;>

<!ELEMENT demand (demand_curve?)>
 <!ATTLIST demand
  curve IDREF ""
  demand CDATA #REQUIRED
  type CDATA #REQUIRED
 >


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Reservoirs                                                      -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ELEMENT level_curve (%on_datacurve_common_children;)>
 <!ATTLIST level_curve %on_datacurve_common_attributes;>

<!ELEMENT head_curve (%on_timecurve_common_children;)>
 <!ATTLIST head_curve %on_timecurve_common_attributes;>

<!ELEMENT fixed-head (%on_waternode_common_children;,head_curve?)>
 <!ATTLIST fixed-head
  %on_waternode_common_attributes;
  curve IDREF ""
  head CDATA "0"
  head_type (total-head|available-head) "available-head"
 >

<!ELEMENT reservoir (%on_waternode_common_children;,level_curve?)>
 <!ATTLIST reservoir
  %on_waternode_common_attributes;
  curve IDREF ""
  top_level CDATA #REQUIRED
  bottom_level CDATA #REQUIRED
  level CDATA #REQUIRED
 >

<!--############################################################-->
<!-- Link elements                                              -->
<!--############################################################-->

<!-- Link group element -->
<!ELEMENT links (pipe|valve|meter|pump)*>


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Pipes                                                          -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
```

```
<!ELEMENT pipe (%on_waterlink_common_children;)>
 <!ATTLIST pipe %on_waterlink_common_attributes;>
 <!ATTLIST pipe
  material CDATA ""
  pipe_type CDATA ""
  year_laid CDATA ""
 >


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Valves                                                                -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ELEMENT valve (%on_watervalve_common_children;)>
 <!ATTLIST valve %on_watervalve_common_attributes;>

<!ELEMENT prv (%on_watervalve_common_children;)>
 <!ATTLIST prv %on_watervalve_common_attributes;>

<!ELEMENT sluice (%on_watervalve_common_children;)>
 <!ATTLIST sluice %on_watervalve_common_attributes;>

<!ELEMENT psv (%on_watervalve_common_children;)>
 <!ATTLIST psv %on_watervalve_common_attributes;>

<!ELEMENT mtv (%on_watervalve_common_children;)>
 <!ATTLIST mtv %on_watervalve_common_attributes;>

<!ELEMENT nrv (%on_watervalve_common_children;)>
 <!ATTLIST nrv %on_watervalve_common_attributes;>

<!ELEMENT pbv (%on_watervalve_common_children;)>
 <!ATTLIST pbv %on_watervalve_common_attributes;>

<!ELEMENT thv (%on_watervalve_common_children;)>
 <!ATTLIST thv %on_watervalve_common_attributes;>

<!ELEMENT flv (%on_watervalve_common_children;)>
 <!ATTLIST flv %on_watervalve_common_attributes;>


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Meters                                                                -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ELEMENT meter (%on_waterlink_common_children;)>
 <!ATTLIST meter %on_waterlink_common_attributes;>


<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->
<!-- Pumps                                                                 -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~-->

<!ELEMENT pump (%on_waterlink_common_children;)>
 <!ATTLIST pump %on_waterlink_common_attributes;>


<!--#########################################################################-->
```

```
<!-- Area elements                                                    -->
<!--################################################################-->

<!-- Area group element -->
<!ELEMENT areas EMPTY>


<!--################################################################-->
<!-- Curve Elements                                                 -->
<!--################################################################-->

<!ELEMENT timestep EMPTY>
 <!ATTLIST timestep
  time CDATA #REQUIRED
  y CDATA #REQUIRED
 >

<!ELEMENT datastep EMPTY>
 <!ATTLIST datastep
  x CDATA #REQUIRED
  y CDATA #REQUIRED
 >


<!-- Curve group element -->
<!ELEMENT curves (head_curve|level_curve)*>


<!--################################################################-->
<!-- Network element                                                -->
<!--################################################################-->

<!-- Network element -->
<!ELEMENT network (nodes*,links*,areas*,curves*)>
 <!ATTLIST network
  friction %on_friction; "hazen-williams"
 >
```

B.2 Example XML Network File

```
<?xml version='1.0' standalone="no" ?>

<!DOCTYPE network PUBLIC "OpenNet v1.0 XML" "OpenNet.dtd">

<network friction="colebrook-white">

 <nodes>

  <node id="n27">
   <coordinate x="552" y="1245" z="72"/>
  </node>

  <node id="n13">
   <coordinate x="780" y="1781" z="50"/>
  </node>

  <node id="n19">
   <coordinate x="1419" y="1594" z="22"/>
  </node>

  <node id="n18">
   <coordinate x="1110" y="1574" z="43"/>
  </node>

  <node id="n12">
   <coordinate x="748" y="1570" z="71"/>
  </node>

  <node id="n11">
   <coordinate x="732" y="1334" z="10"/>
  </node>

  <node id="n9">
   <coordinate x="642" y="1074" z="-45"/>
  </node>

  <node id="n20">
   <coordinate x="1061" y="1131" z="50"/>
  </node>

  <node id="n16">
   <coordinate x="963" y="716" z="33"/>
  </node>

  <node id="n17">
   <coordinate x="317" y="505" z="22"/>
  </node>

  <node id="n10">
   <coordinate x="553" y="915" z="35"/>
  </node>

  <node id="n8">
   <coordinate x="553" y="1224" z="72"/>
  </node>

  <node id="n7">
```

```
    <coordinate x="455" y="1395" z="45"/>
   </node>

   <node id="n6">
    <coordinate x="443" y="1793" z="29"/>
   </node>

   <node id="n5">
    <coordinate x="427" y="1893" z="42"/>
   </node>

   <node id="n4">
    <coordinate x="431" y="2000" z="38"/>
   </node>

   <node id="n14">
    <coordinate x="695" y="2122" z="17"/>
   </node>

   <node id="n3">
    <coordinate x="496" y="2240" z="61"/>
   </node>

   <node id="n2">
    <coordinate x="528" y="2468" z="88"/>
   </node>

   <node id="n15">
    <coordinate x="687" y="2403" z="52"/>
   </node>

   <fixed-head id="n1" head_type="total-head">
    <coordinate x="610" y="2716" z="100"/>
    <head_curve id="curve_n1">
     <timestep time="0" y="174.2"/>
     <timestep time="5" y="188.35"/>
     <timestep time="8" y="151.02"/>
     <timestep time="15" y="172.1"/>
    </head_curve>
   </fixed-head>

   <fixed-head id="fh1" head="15.5" head_type="available-head">
    <coordinate x="632" y="2723" z="72.5"/>
   </fixed-head>

   <fixed-head id="fh2" curve="curve_fh2" head_type="total-head">
    <coordinate x="146" y="1547" z="86.2"/>
   </fixed-head>

   <reservoir id="r1" bottom_level="35" level="52" top_level="71">
    <coordinate x="610" y="2716" z="100"/>
    <level_curve id="curve_r1">
     <datastep x="35" y="78.2"/>
     <datastep x="71" y="192.8"/>
    </level_curve>
   </reservoir>

  </nodes>
```

```
<links>

 <pipe id="P14" length="0" diameter="0" length_unit="metres">
  <link from_node="n15" to_node="n14"/>
 </pipe>

 <pipe id="P8" diameter="150">
  <link from_node="n8" to_node="n9"/>
 </pipe>

 <pipe id="P7" diameter="150" diameter_unit="inches">
  <link from_node="n7" to_node="n8"/>
 </pipe>

 <pipe id="P6" diameter="150">
  <link from_node="n6" to_node="n7"/>
 </pipe>

 <pipe id="P4" diameter="150">
  <link from_node="n4" to_node="n5"/>
 </pipe>

 <pipe id="P3" diameter="150">
  <link from_node="n3" to_node="n4"/>
 </pipe>

 <pipe id="P21" diameter="150">
  <link from_node="n9" to_node="n16"/>
 </pipe>

 <pipe id="P16" diameter="150">
  <link from_node="n10" to_node="n17"/>
 </pipe>

 <pipe id="P9" diameter="150">
  <link from_node="n9" to_node="n10"/>
 </pipe>

 <pipe id="P10" diameter="150">
  <link from_node="n11" to_node="n9"/>
 </pipe>

 <pipe id="P20" diameter="150">
  <link from_node="n20" to_node="n16"/>
 </pipe>

 <pipe id="P5" diameter="150">
  <link from_node="n5" to_node="n6"/>
 </pipe>

 <pipe id="P2" diameter="150">
  <link from_node="n2" to_node="n3"/>
 </pipe>

 <pipe id="P1" diameter="150">
  <link from_node="n1" to_node="n2"/>
 </pipe>

 <pipe id="P13" diameter="150">
```

```
    <link from_node="n14" to_node="n13"/>
   </pipe>

   <pipe id="P18" diameter="150">
    <link from_node="n18" to_node="n19"/>
   </pipe>

   <pipe id="P15" diameter="150">
    <link from_node="n1" to_node="n15"/>
   </pipe>

   <pipe id="P17" diameter="150">
    <link from_node="n12" to_node="n18"/>
   </pipe>

   <pipe id="P11" diameter="150">
    <link from_node="n12" to_node="n11"/>
   </pipe>

   <pipe id="P12" diameter="150">
    <link from_node="n13" to_node="n12"/>
   </pipe>

   <pipe id="P19" diameter="150">
    <link from_node="n11" to_node="n20"/>
   </pipe>

   <valve id="V1" diameter="150">
    <link from_node="n11" to_node="n20">
     <coordinate x="744" y="1384" z="11"/>
     <line>
      <coordinate x="744" y="1384" z="11"/>
      <coordinate x="744" y="1384" z="11"/>
     </line>
    </link>
   </valve>

  </links>

  <curves>

   <head_curve id="curve_fh2">
    <timestep time="0" y="90.9"/>
    <timestep time="12" y="114.2"/>
   </head_curve>

  </curves>

 </network>
```

Bibliography

Papers presented by the candidate

MORLEY, M.S., ATKINSON, R.M., SAVIĆ, D.A. & WALTERS, G.A. 2000. OpenNet: An application-independent framework for hydraulic network representation, manipulation and dissemination. *Proceedings 4th Hydroinformatics Conference of the International Association for Hydraulic Research, Iowa City, U.S.A.* p10.

MORLEY, M.S., ATKINSON, R.M., SAVIĆ, D.A. & WALTERS, G.A. 2001. GAnet: Genetic Algorithm platform for pipe network optimization. *Advances in Engineering Software*. **32**, pp467-475.

MORLEY, M.S., MAKROPOULOS, C.K., SAVIĆ, D.A. & BUTLER, D. 2004. Decision-Support System Workbench for Sustainable Water Management Problems. In: PAHL-WOSTL, C., SCHMIDT, S., RIZZOLI, A.E. AND JAKEMAN, A.J. (eds.) *Complexity and Integrated Resources Management, Transactions of the 2nd Biennial Meeting of the International Environmental Modelling and Software Society, University of Osnabrück, Germany*. iEMSs: Manno, Switzerland.

MORLEY, M.S., TRICARICO, C., KAPELAN, Z., SAVIĆ, D.A. & DE MARINIS, G. 2006. deEPANET: A Distributed Hydraulic Solver Architecture for Accelerating Optimization Applications Working With Conditions of Uncertainty. In: GOURBESVILLE, P., CUNGE, J., GUINOT, V. & LIONG, S-Y. (eds.) *Proceedings 7th International Conference on Hydroinformatics, Nice, France*. pp2465-2472.

MORLEY, M.S. & TRICARICO, C. 2008. *Pressure Driven Demand Extension for EPANET (EPANETpdd) – Technical Report 2008-02*. Centre for Water Systems, University of Exeter, UK. 10pp.

Other Papers arising from this work

Published

ATKINSON, R.M., MORLEY, M.S., SAVIĆ, D.A. & WALTERS, G.A. 1998. The Integration of GIS, Network Analysis and Genetic Algorithm Optimization Software for Water Network Analysis. In: BABOVIC, V. & LARSEN, L.C. (eds.) *Proceedings 3rd Hydroinformatics Conference of the International Association for Hydraulic Research, Balkema, Rotterdam, Netherlands*. pp357-362.

BICIK, J., MORLEY, M.S. & SAVIĆ, D.A. 2008. A Rapid Optimization Prototyping Tool for Spreadsheet-Based Models. In: VAN ZYL, J.E, ILEMOBADE, A.A. & JACOBS, H.E. (eds.) *Proceedings of the 10th Annual Water Distribution Systems Analysis Conference WDSA2008, August 17-20, 2008, Kruger National Park, South Africa*. pp472-482.

[1]DE MARINIS, G., GARGANO, R., KAPELAN, Z., MORLEY, M.S., SAVIĆ, D.A. & TRICARICO, C. 2007a. The Influence of the Hydraulic Simulator in Water Distribution System Rehabilitation Analysis. In: ULANICKI, B., VAIRAVAMOORTHY, K., BUTLER, D., BOUNDS, P.L.M. & MEMON, F.A. (eds.) *Supplementary Proceedings of the Combined International Conference of Computing and Control for the Water Industry (CCWI2007) and Sustainable Urban Water Management (SUWM2007)* de Montford University, Leicester, UK. pp7-14.

[1]DE MARINIS, G., GARGANO, R., KAPELAN, Z., MORLEY, M.S., SAVIĆ, D.A. & TRICARICO, C. 2007b. Extended Period Simulation in the Estimation of the Economic Level of Reliability for the Rehabilitation of Water Distribution Systems. *Proceedings 2nd Leading Edge Conference on Strategic Asset Management – LESAM 2007, Lisbon, Portugal.* IWA and LNEC, Lisbon, Portugal on CD-ROM.

[1]DE MARINIS, G., GARGANO, R., KAPELAN, Z., MORLEY, M.S., SAVIĆ, D.A. & TRICARICO, C. 2008. Risk-Cost Based Decision Support System for the Rehabilitation of Water Distribution Networks. In: VAN ZYL, J.E, ILEMOBADE, A.A. & JACOBS, H.E. (eds.) *Proceedings of the 10th Annual Water Distribution Systems Analysis Conference WDSA2008, August 17-20, 2008, Kruger National Park, South Africa.* pp652-664.

MAKROPOULOS, C., MORLEY, M., MEMON, F., BUTLER, D., SAVIĆ, D. & ASHLEY, R. 2006. A Decision Support Framework for Sustainable Urban Water Planning and Management in New Urban Areas. *Water Science & Technology*, **54,** nos. 6-7, pp 451-458.

SAVIĆ, D.A., WALTERS, G.A. & MORLEY, M.S. 1997. Applications of Genetic Algorithms in the Water Industry. *2nd meeting of the EPSRC Advanced Computing Techniques Community Club, Rutherford-Appleton Laboratory, Didcot,U.K.* Poster presentation.

WALTERS, G.A., SAVIĆ, D.A., MORLEY, M.S., DE SCHAETZEN, W.F.B. & ATKINSON, R.M. 1998. Calibration of Water Distribution Network Models Using Genetic Algorithms. *Proceedings 7th International Conference on Hydraulic Engineering Software, Como, Italy.* Computational Mechanics Publications, Southampton, U.K. pp131-140.

In preparation

KEEDWELL, E.C., MORLEY, M.S. & SAVIĆ, D.A. 2008. An Investigation into Caching for Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computing (submitted).*

[1] The authors on these papers are presented in alphabetical order and are not intended to be a reflection of the relative contribution of the individual authors to the research presented therein.

List of References

7-TECHNOLOGIES A/S. 2002. *Aquis User Manual.* 7-Technologies A/S, Birkerød, Denmark.

ADVANTICA, INC. 2003. *SynerGEE Water User Manual.* Advantica Inc./Stoner Software, Houston, Texas, U.S.A.

ALPEROVITS, E. & SHAMIR, U. 1977. Design of Optimal Water Distribution System. *Water Resources Research.* **13**, 6. pp885-900.

ANG, W.K. & JOWITT, P.W. 2006. Solution for Water Distribution Systems under Pressure-Deficient Conditions. *Journal of Water Resources Planning and Management - ASCE*, **132**(3) pp175-182.

BABAYAN, A.V., SAVIĆ, D.A. & WALTERS, G.A. 2003. Least-cost design of water distribution networks under uncertain demand. In: MAKSIMOVIC, C., BUTLER, D. & MEMON, F. (eds.). *Advances in Water Supply Management.* A.A. Balkema Publishers, pp. 139-146.

BAKER, J.E. 1985. Adaptive Selection Methods for Genetic Algorithms. In: GREFENSTETTE, J.J. (ed.) *Proceedings of the First International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, Hillsdale, U.S.A. pp101-111.

BALLA, M.C. & LINGIREDDY, S. 2000. Distributed genetic algorithm model on network of personal computers. *Journal of Computing in Civil Engineering., ASCE*, **14**(3), pp199-205.

BHAVE, P.R. 1978. Noncomputer Optimization of Single-Source Networks. *ASCE, Proc. J. Environmental Engineering Division.* **104**(4). pp799-813.

BLIND, M. & GREGERSEN, J.B. 2004. Towards an Open Modelling Interface (OpenMI) – The HarmonIT Project. In PAHL-WOSTL, C., SCHMIDT, S., RIZZOLI, A.E. AND JAKEMAN, A.J. (eds), *Complexity and Integrated Resources Management, Transactions of the 2nd Biennial Meeting of the International Environmental Modelling and Software Society*, iEMSs: Manno, Switzerland. ISBN 88-900787-1-5

BORLAND INTERNATIONAL. 1997. *Delphi version 3.0 User's Guide.* Borland International, Scotts Valley, U.S.A.

BULLNHEIMER, B., HARTL, R.F. & STRAUSS, C. 1999. "A new rank based version of the Ant System: A computational study. *Central European Journal for Operations Research and Economics*, **7(**1), pp25-38.

CHU, P.C. & BEASLEY, J.E. 1997. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, **24**, pp17-23.

CORCORAN, A.L. 1993. *libGA User's Manual.* University of Tulsa, U.S.A.

CORMEN, T.H., LEISERSON, C.E., RIVEST, R.L. & STEIN, C. 2001. *Introduction to Algorithms, Second Edition.* MIT Press and McGraw-Hill.

CUNHA, M. DA C. & SOUSA, J. 1999. Water Distribution Network Design Optimization: Simulated Annealing Approach. *Journal of Water Resources Planning and Management - ASCE*, **125**(4), pp215-221.

DANDY, G.C., SIMPSON, A.R. & MURPHY, L.J. 1996. An improved genetic algorithm for pipe network optimization. *Water Resources Research*, **32**(2), pp449–458.

DANDY, G.C. & ENGELHART, M. 2001. Optimal scheduling of water pipe replacement using genetic algorithms. *Journal of Water Resources Planning and Management - ASCE*,, **127**(4), pp214-223.

DARWIN, C.R. 1859. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*. John Murray, London. 502pp.

DEB, K. 2001. *Multi-objective optimization using evolutionary algorithms*, Wiley Interscience, Hoboken, U.S.A.

DIJKSTRA, E.W. 1959. A note on two problems in connection with graphs. *Numerische Mathematik*, **1**, pp269-271.

1993. Application of GIS network routines for water-flow and transport. *Journal of Water Resources Planning and Management - ASCE*, **119**(2), pp229-245.

DORIGO, M., MANIEZZO, V. & COLORNI, A. 1996. The ant system: optimization by a colony of cooperating ants. *IEEE Transactions on Systems, Man & Cybernetics – Part B,* **26**(1), pp29–42.

EBEHART, R. C. & KENNEDY, J. 1985. A new optimizer using particles swarm theory. *Proceedings 6th International Symposium on Micro Machine and Human Science*, IEEE Service Centre, Piscataway, U.S.A., pp39-43.

ENGELHARDT, M.O., SKIPWORTH, P.J., CASHMAN, A., SAVIĆ, D., SAUL, A.J. & WALTERS, G.A. 2002. *A Whole Life Costing for Water Distribution Network Management.* Thomas Telford Ltd, London UK (ISBN 0-7277-3166-1), 216pp.

ENGELHARDT, M. & SKIPWORTH, P. 2005. WiLCO – State of the art decision support . *Water Management for the 21st Century – Prooceedings Computing and Control in the Water Industry,* Exeter, UK.

ESRI. 1999. *ArcInfo User's Manual.* Environmental Systems Research Institute, Redlands, California, U.S.A.

EUSUFF, M.M. & LANSEY, K.E. 2003a. Water Distribution Network Design Using The Shuffled Frog Leaping Algorithm. *Proceedings World Water and Environmental Resources Congress (ASCE)*, Orlando, U.S.A.

EUSUFF, M.M. & LANSEY, K.E. 2003b. Optimization of water distribution network design using the shuffled frong leaping algorithm. *Journal of Water Resources Planning and Management - ASCE*, **129**(3), pp210-225.

FIELDSEND, J.E., EVERSON, R.M. & SINGH, S. 2003. Using Unconstrained Elite Archives for Multi-Objective Optimization, *IEEE Transactions on Evolutionary Computation* **7**(3), pp 305-323.

FONSECA, C.M. & FLEMING, P.J. 1993. Genetic algorithms for multi-objective optimization; optimization, formulation discussion and generalization. In: FORREST, S. *Proceedings 5th International Conference on Genetic Algorithms*, University of Illinois at Urbana-Champaign, U.S.A. pp416-423.

FUJIWARA, O. & KHANG, D.B. 1990. A two-phase decomposition method for optimal design of looped water distribution networks. *Water Resources Research,* **26**(4), pp539-549.

FUJIWARA, O. & LI, J. 1998. Reliability Analysis of Water Distribution Networks in Consideration of Equity, Redistribution and Pressure-Dependent Demand. *Water Resources Research*, **34**(7) pp1843-1850.

FULLERTON, J.N., WALTERS, G.A. & SAVIĆ, D.A. 2002 Simplified Modelling of Storm Water Flows for Optimization. In: BREBBIA, C.A. & BLAIN, W.R. (eds.), *Hydraulic Information Management*, WIT press, Southampton, U.K. pp133-142.

GARDNER, M. 1970. Mathematical Games: The Fantastic Combinations of John Conway's new Solitaire Game, "Life". *Scientific American*. (223). pp120-123.

GEEM, Z.W., KIM, J.H. & LOGANATHAN, G.V. 2002. Harmony Search Optimization: Application to Pipe Network Design. *International Journal of Model Simulation*, **22**(2), pp125-133.

GERMANOPOULOS, G. 1985. A technical note on the inclusion of pressure dependent and leakage terms in water supply network models, *Civil Engineering Systems*, **2**(3), pp171-179.

GERMANOPOULOS, G. 1985. Pipe network optimization by enumeration. *Proc. Spec. Conf. on Computer Applications in Water Resources*. ASCE, New York, U.S.A. pp572-581.

GOLDBERG, D.E. 1989. *Genetic algorithms: in search, optimization and machine learning*. Addison-Wesley, Reading, Massachusetts, U.S.A. 412pp.

GOLDBERG, D.E. 1987. Genetic algorithms in pipeline optimization. *Journal of Computing in Civil Engineering - ASCE*, **1**(2), pp128-141.

GOLDBERG, D.E. & RICHARDSON, J. 1987. Genetic Algorithms with Sharing for Multimodal Function Optimization. In:GREFENSTETTE, J.J. (ed.) *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Hillsdale, U.S.A. pp41-49.

GOLDBERG, D.E., KORB, B. & DEB, K. 1989. Messy genetic algorithms: motivation, analysis and first results. *Complex Systems,* **3**, pp493-530.

GOLDBERG, D.E., DEB, K. & KORB, B. 1991. Do not Worry, Be Messy. In: BELEW, R. & BOOKER, L. (eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, U.S.A. pp24-30.

GRAY, F. 1953. *Pulse Code Communication*, U. S. Patent 2 632 058.

GUPTA, I., GUPTA, A. & KHANNA, P. 1999. Genetic algorithm for optimization of water distribution systems. *Environmental Modelling and Software,* **14**, pp437-446.

HALHAL, D., WALTERS, G.A., SAVIĆ, D.A. & OUAZAR, D. 1999. Scheduling of Water Distribution System Rehabilitation using Structured Messy Genetic Algorithms, *Evolutionary Computation*, **7**(3), pp311-329.

HALHAL, D., WALTERS, G.A., OUAZAR, D. AND SAVIĆ, D.A. 1997. Water network rehabilitation with structured messy genetic algorithm. *Journal of Water Resources, Planning and Management, ASCE,* **123**(3), pp137-146.

HEWLETT PACKARD COMPANY. 2001. *Programming with Judy: C Language, Judy version 4.0*. Hewlett Packard, Fort Collins, Colorado, USA. p/n B6841-90001.

HOLLAND, J.H. 1975. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, Massachusetts, U.S.A.

ISO (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION). 1986. *ISO 8879:1986(E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML). First edition*. International Organization for Standardization, Geneva, Switzerland.

JANIKOW, C. & MICHALEWICZ, Z. 1991. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. In: BELEW, R. & BOOKER, L. (eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, San Mateo, U.S.A. pp31-36.

JOSUTTIS, N.M. 1996. *The C++ Standard Library: A Tutorial and Reference.* Addison-Wesley. ISBN 0-201-37926-0.

KADU, M.S., GUPTA, R. & BHAVE, P.R. 2008. Optimal Design of Water Networks Using a Modified Genetic Algorithm with Reduction in Search Space. *Journal of Water Resources Planning and Management - ASCE*, **134**(2), pp147-160.

KAPELAN, Z., SAVIĆ, D.A. & WALTERS, G.A. 2003a. Robust least cost design of water distribution systems using GAs. In: MAKSIMOVIĆ, C., BUTLER, D. & MEMON, F. (eds.). *Advances in Water Supply Management.* A.A. Balkema Publishers, pp. 147-155.

KAPELAN, Z., SAVIĆ, D.A. & WALTERS, G.A. 2003b Multiobjective sampling design for water distribution model calibration. *Journal of Water Resources Planning and Management - ASCE,* **129**(6), pp466-479.

KAPELAN, Z., SAVIĆ, D.A. & WALTERS, G.A. 2004. A multiobjective approach to rehabilitation of water distribution networks under uncertainty, *Proceedings of the 6th International Symposium on Systems Analysis and Integration Assessment, WATERMATEX*, IWA Beijing, China.

KAPELAN, Z., SAVIĆ, D.A. & WALTERS, G.A. 2005. Multiobjective Design of Water Distribution Systems under Uncertainty. *Water Resources Research.* **41**(11), W11407.

KEEDWELL, E.C. & KHU, S-T. 2006. Novel Cellular Automata Approach to Optimal Water Distribution Network Design. *Journal of Computing in Civil Engineering- ASCE,* **20**(1), pp49-56.

KERNIGHAN, B.W. & RITCHIE, D.M. 1988. *The C Programming Language - 2nd edition*, Prentice Hall, Englewood Cliffs, New Jersey, U.S.A.

KNUTH, D.E. 1997a. *The Art of Computer Programming. Voume 1: Fundamental Algorithms (Third Edition).* Addison-Wesley, Reading, U.S.A.

KNUTH, D.E. 1997b. *The Art of Computer Programming. Voume 3: Sorting and Searching (Third Edition).* Addison-Wesley, Reading, U.S.A.

KRATICA, J., TOSIC, D., FILIPOVIC, V. & LJUBIC, I. 2001. Solving the Simple Plant Location Problems by Genetic Algorithm, *RAIRO Operations Research*, **35**, pp127–142.

LIPPAI, I., HEANEY, J.P. & LAGUNA, L. 1999. Robust water system design with commercial intelligent search optimizers. *Journal Computing in Civil Engineering – ASCE.* **13**(3). pp135-143.

LUPIEN, A.E., MORELAND, W.H. & DANGERMOND, J. 1987. Network analysis in Geographic Information Systems. *Journal of Photogrammetric Engineering and Remote Sensing*, **53**, 10, pp1417-1421.

MAPINFO CORPORATION. 1998. *MapInfo Professional User's Reference.* MapInfo Corporation, Troy, New York, U.S.A.

MAIER, H.R., SIMPSON, A.R., ZECCHIN, A.C., FOONG, W.K., PHANG, K.Y., SEAH, H.S.& CHAN LIM TAN, C.L. 2003. Ant Colony Optimization for Design of Water

Distribution Systems. *Journal of Water Resources Planning and Management - ASCE*, **129**(3), pp200-209.

MAKROPOULOS, C.K., BUTLER, D. & MAKSIMOVIĆ, C. 2003. Fuzzy Logic Spatial Decision Support System for Urban Water Management. *Journal of Water Resources Planning and Management - ASCE,* **129**(1), pp69-77.

MAKROPOULOS, C.K. & BUTLER, D. 2005. A multi-objective evolutionary programming approach to the 'object location' spatial analysis and optimisation problem within the urban water management domain. *Civil Engineering and Environmental Systems*, **22**(2), pp85-101.

MEIER, R. W. & BARKDOLL, B. D. 2000. Sampling design for network model calibration using genetic algorithms. *Journal of Water Resources Planning and Management - ASCE,* **126**(4), pp245-250.

MICHALEWICZ, Z. 1992. *Genetic algorithms + data structures = evolution programs.* Springer-Verlag, Berlin, Germany..

MORGAN, D.R. & GOULTER, I.C. 1985. Optimal urban water distribution design. *Water Resources Research*, **21**, 5, pp642-652.

MUNAVALLI, G.R. & KUMAR, M.S. 2003. Optimal scheduling of multiple chlorine sources in water distribution systems. *Journal of Water Resources Planning and Management - ASCE,* **129**(6), pp493-504.

MURPHY, L.J., DANDY, G.C. & SIMPSON, A.R. 1993. Design of a Pipe Network Using Genetic Algorithms. *Water*, **20**(4), pp40-42.

PFAFF, B. 2004. Performance Analysis of BSTs in System Software. In: COFFMAN, E.G. (Jr.), LIU, Z. & MERCHANT, A. (eds.) *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, ACM, New York, U.S.A. pp410-411.

POVINELLI, R. & FENG, X. 1999. Improving Genetic Algorithms Performance by Hashing Fitness Values. *Proceedings Artificial Neural Networks in Engineering*, St. Louis, Missouri, U.S.A. pp399-404.

REED, P., KOLLAT, J.B. & DEVIREDDY, V. 2007. Using Interactive Archives in Evolutionary Multiobjective Optimization: A Case Study for Long-Term Groundwater Monitoring Design, *Environmental Modeling & Software*, **22**(5), pp683-692.

RIGGS, R.L. 1994. Application of AM/FM/GIS technology to the pipeline industry. *Proceedings AM/FM GIS 1994*, pp437-447.

ROSSMAN, L.A. 1993. *EPANET User's Manual.* United States Environmental Protection Agency, Cincinnati, U.S.A.

ROSSMAN, L.A. 2000. *EPANET 2 User's Manual.* United States Environmental Protection Agency, Cincinnati, U.S.A.

ROSSMAN, L.A. 2005. *Storm Water Management Model User's Manual version 5.0.* United States Environmental Protection Agency, Cincinnati, U.S.A.

SAVIĆ, D.A. & WALTERS, G.A. 1994. Evolution Programs in Optimal Design of Hydraulic Networks. In: PARMEE, I.C. (ed.) *Adaptive Computing in Engineering Design and Control - '94*. pp146-150. University of Plymouth, U.K..

SAVIĆ, D.A. & WALTERS, G.A. 1995. An evolution program for optimal pressure regulation in water distribution networks. *Engineering Optimization,* **24**(3), pp197-219.

SAVIĆ, D.A. & WALTERS, G.A. 1997. Genetic algorithms for least-cost design of water distribution networks. *Journal of Water Resources Planning and Management - ASCE,* **123**(2), pp67–77.

SAVIĆ, D.A. & WALTERS, G.A, RANDALL-SMITH, M. & ATKINSON, R.M. 2000. Large Water Distribution Systems Design through Genetic Algorithm Optimization. In: HOTCHKISS, R.H. & GLADE, M. (eds.) *ASCE 2000 Joint Conference on Water Resources Engineering and Water Resources Planning and Management*, July 30-August 2, Minneapolis, USA. (proceedings published on CD), p10.

SCHAAKE, J. & LAI, D. 1969. *Linear programming and dynamic programming application of water distribution network design (Report 116).* MIT Press, Cambridge, U.S.A.

DE SCHAETZEN, W.B.F., WALTERS, G.A. & SAVIĆ, D.A. 2000. Optimal sampling design for model calibration using shortest path, genetic and entropy algorithms. *Urban Water,* **2**, pp141-152.

SILVERSTEIN, A. 2002. *Judy IV Shop Manual.* Hewlett Packard, Fort Collins, Colorado, U.S.A.

SIMPSON, A.R., DANDY, G.C. & MURPHY, L.J. 1994. Genetic algorithms compared to other techniques for pipe optimization. *Journal of Water Resources Planning and Management - ASCE,* **120**(4), pp423-443.

SOLOMATINE, D.P. 1996. Object orientation in hydraulic modelling architectures. *Journal of Computing in Civil Engineering - ASCE*, **10**, 2, pp125-135.

SRINIVAS, N., DEB, K. 1994 Multiobjective function optimization using non-dominated sorting genetic algorithm*, EvolutionaryComputation*, **2**(3): pp221–248.

STEPANOV, A.A. & LEE, M. 1994. *The Standard Template Library.* Technical Report HPL-94-34. Hewlett Packard, Fort Collins, Colorado, USA.

STROUSTRUP, B. 1997. The C++ Programming Language (Third Edition). Addison-Wesley, Reading, U.S.A.

STRUCTURAL TECHNOLOGIES LTD. 1996. *StruMap Geographic Information System User's Manual.* Structural Technologies Ltd., Studley, U.K.

STÜTZLE, T. & HOOS, H.H. 2000. MAX-MIN Ant System. *Future Generation Computer Systems,* **16**, pp889-914.

SYSWERDA, G. 1989. Uniform Crossover in Genetic Algorithms. In: SCHAFFER, J. (ed.) *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, San Mateo, U.S.A. pp2-9.

TAHER, S.A. & LABADIE, J.E. 1996. Optimal design of water-distribution networks with GIS. *Journal of Water Resources Planning and Management - ASCE*, **122**, 4, pp301-311.

TEMPLEMAN, A.B., 1982. Discussion of "Optimization of Looped Water Distribution Systems", by QUINDRY, G.E., BRILL, E.D. AND LIEBMAN, J.C. *Journal Environmental Engineering. Division - ASCE*, **108**(EE3), pp599–602.

THURLEY, R.W.F., SAVIĆ, D.A. & WALTERS, G.A. 1999. The application of parallel processing to GA-based optimization of water supply systems. In: POWELL, R. & HINDI, K.S. (eds.) *Computing and Control for the Water Industry*. Research Studies Press, Baldock, Hertfordshire, U.K. pp329-336.

TODINI, E. & PILATI, S. 1987. A Gradient Method for the Analysis of Pipe Networks. *Proceedings International Conference on Computer Applications for Water Supply and Distribution.* Leicester Polytechnic, U.K.

TRICARICO, C. 2005. *A Rehabilitation Model for Water Distribution Systems: Multiobjective Optimization Based on the Cost of Reliability (PhD Thesis).* Università degli Studi di Cassino, Cassino FR, Italy. 238pp.

TRICARICO, C., DE MARINIS, G., GARGANO, R. & LEOPARDI, A. 2005. Peak Demand for Small Towns. In: SAVIĆ, D.A., WALTERS, G.A., KING, R. & KHU, S.-T. (eds.) *Proceedings of the Eighth International Conference on Computing and Control for the Water Industry (CCWI2005).* University of Exeter, UK. **2**, pp113-118.

TRICARICO, C., GARGANO, R., KAPELAN, Z., SAVIĆ, D.A. & DE MARINIS, G. 2006. Economic Level of Reliability for the Rehabilitation of Hydraulic Networks, *Journal of Civil Engineering and Environmental Systems.* **23**(3) pp191-207.

TSAKIRIS, G. & SALAHORIS, M. 1993. GIS technology for management of water distribution networks. In: CABRERA, E. & MARTÍNEZ, F. (eds.) *Water Supply Systems: State of the art and future trends.* Computational Mechanics Publications, Southampton, U.K. pp359-378.

VAN ZYL, J.E., BORTHWICK, J. & HARDY, A. 2003. OOTEN: An Object-Oriented Programmer's Toolkit for EPANET. In: MAKSIMOVIĆ, C., BUTLER, D. & MEMON, F.A. (eds.) *Proceedings of the Seventh International Conference on Computing and Control for the Water Industry (CCWI2003).* Imperial College London, UK. *Supplementary Paper.*

VITKOVSKY, J.P. & SIMPSON, A.R. 1997. *Calibration and Leak Detection in Pipe Networks Using Inverse Transient Analysis and Genetic Algorithms.* Department of Civil and Environmental Engineering, University of Adelaide: Adelaide, Australia. 97pp.

VON NEUMANN, J. 1966. *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana, U.S.A. 388pp.

W3C (WORLD WIDE WEB CONSORTIUM). 2000. *Extensible Markup Language (XML) 1.0 (Second Edition).*

WALLINGFORD SOFTWARE. 2005. *InfoWorks WS User Manual.* Wallingford Software, Wallingford, U.K.

WALSKI, T.M. 1984. *Analysis of Water Distribution Systems.* van Nostrand Reinhold Co., New York, U.S.A.

WALSKI, T.M, BRILL, E.D., GESSLER, J., GOULTER, I.C., JEPPSON, R.M., LANSEY, K., HAN-LIN LEE, LIEBMAN, J.C., MAYS, L., MORGAN, D.R. & ORMSBEE, L. 1987. Battle of the Network Models: epilogue. *Journal of Water Resources Planning and Management - ASCE,* **113**(2), pp191-203.

WALSKI, T.M. 1990. *Water distribution systems: simulation and sizing.* Lewis Publishers, Boca Raton, U.S.A.

WALTERS, G.A., SAVIĆ, D.A., MORLEY, M.S., DE SCHAETZEN, W.F.B. & ATKINSON, R.M. 1998. Calibration of Water Distribution Network Models Using Genetic Algorithms. *Proceedings 7th International Conference on Hydraulic Engineering Software, Como, Italy.* Computational Mechanics Publications, Southampton, U.K. pp131-140.

WALTERS, G.A., HALHAL, D., SAVIĆ, D.A. & OUAZAR, D. 1999. Improved design of "Anytown" distribution network using structured messy genetic algorithms. *Urban Water,* **1**(1), pp23-38.

WU, Z.Y. & SIMPSON, A.R. 2001. Competent genetic-evolutionary optimization of water distribution systems. *Journal of Computing in Civil Engineering - ASCE,* **15**(2), pp89-101.

ZECCHIN, A.C., MAIER, H.R., SIMPSON, A.R., LEONARD, M. & NIXON, J.B. 2007. Ant Colony Optimization Applied to Water Distribution System Design: Comparative Study of Five Algorithms. *Journal of Water Resources Planning and Management - ASCE*, **133**(1), pp87-92.

ZECCHIN, A.C., SIMPSON, A.R., MAIER, H.R., LEONARD, M., ROBERTS, A.J. & BERRISFORD, M.J. 2006. Application of Two Ant Colony Optimization Algorithms to Water Distribution System Optimization. *Mathematical and Computing Modelling*, **44**(5-6), pp451-468.

ZITZLER, E. & THIELE, L. 1999. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions of Evolutionary Computation.* **3**(4), pp257-271.