

Gems:
A novel method to improve Evolutionary Algorithms.



Author: Rory Murphy BA, MA. Student number 1420 7754.

A thesis submitted to University College Dublin in part fulfilment of the requirements of the degree of M.Sc. in Business Analytics.

Michael Smurfit Graduate School of Business.



September 2016.

Supervisor: Professor Michael O'Neill B.Sc., H.Dip.Comp.Sc., Ph.D.

Head of School: Professor Ciarán Ó hÓgartaigh.

Dedicated to my mother, Anna, and my wife, Orla, for all their love and support.

Table of Contents

ABSTRACT	VI
1. INTRODUCTION	1
2. LITERATURE REVIEW – AN INTRODUCTION TO EAS	7
3. GEM CONCEPTS AND MODELS	22
4. EXPERIMENTAL DESIGN AND IMPLEMENTATION.....	32
5. PHASE 1: INITIAL INVESTIGATIONS.....	50
6. PHASE 2: FURTHER INVESTIGATIONS	64
7. CONCLUSIONS	92
8. SELECTED BIBLIOGRAPHY.....	96
9. APPENDICES	102

List of Figures

Figure 1.1. A tour of 50 USA state capitals.	4
Figure 1.2. Map A: Sections of a tour of USA cities.	4
Figure 1.3. Map B: The tour sections in Map A changed by mutation.....	5
Figure 2.1. Evolutionary Algorithm flowchart	8
Figure 2.2. Comparison of Evolutionary Algorithm with Random Search.	10
Figure 2.3. A solution fitness landscape.	14
Figure 3.1. Gem functionality within EA's "Mutate Child" step	24
Figure 3.2. Four permutations of matching Hop gems.	27
Figure 4.1. Visual and technical ways to represent a tour.	36
Figure 4.2. Representation of a Flip gem.....	39
Figure 4.3. Representation of a Hop gem.	41
Figure 4.4. Geometric progression of the 33 log-stages' end-points.....	45
Figure 4.5. Data Flow Diagram for gathering statistical data.	46

Figure 4.6. Distribution of individuals' fitnesses.	48
Figure 4.7. The logging database schema	49
Figure 5.1. Fitness comparison. Graph-size: 42. Seed population: Random.	51
Figure 5.2. Fitness comparison. Graph-size: 42. Seed population: Greedy.	52
Figure 5.3. Fitness with/ without Gems, various cases.	54
Figure 5.4. Root-cause explanation.....	55
Figure 5.5. Fitness increases for Flip and Hop mutations.....	55
Figure 5.6. Fitness boost for selected cases.	56
Figure 5.7. Fitness boost for Jewellery-box sizes 4 and 8.	58
Figure 5.8. Fitness variance starts low, and reduces.	59
Figure 5.9. Gem activity chart for Graph-size 42, Random-seed population.	59
Figure 5.10. Gem activity chart for Graph-size 76, Random-seed population.	60
Figure 5.11. Gem functionality in Phase 1.....	61
Figure 6.1. Comparison of Phase 1 and Phase 2 gem application rules.....	65
Figure 6.2. Random vs Greedy-seed cases.....	66
Figure 6.3. Four graph-size cases (all random-seed).	67
Figure 6.4. Fitness boost during process.....	68
Figure 6.5. Small vs Large population size.....	69
Figure 6.6. Boost for small and large population sizes.	70
Figure 6.7. Comparison of Gem Usage limits.....	71
Figure 6.8. Gem creation and deletion rates, per graph size.	71
Figure 6.9. Number of gems active per graph size.	72
Figure 6.10. Number of gem applications per100 iterations per graph size.	73
Figure 6.11. IQR and σ for 42-node graph, Random-seed, Population-size: 1000...	74
Figure 6.12. IQR for various graph sizes, Random-seed, Population-size: 1000.	75
Figure 6.13. Rates of creating individuals, Random-seed, Population-size: 1000. ..	76

Figure 6.14. Saturation: gems per chromosome, and per 100 genes.....	77
Figure 6.15. Proportion of pure individuals in small and large populations.....	78
Figure 6.16. Impact of Flip and Hop gems.	78
Figure 6.17. Fitness boost from Flip and Hop gems.	79
Figure 6.18. Numbers of Flip and Hop gems (population-size: 1000).	79
Figure 6.19. Application rates per Flip and Hop gem.....	80
Figure 6.20. Hybrid process results.....	82
Figure 6.21. Hybrid with earlier switch-off	82
Figure 6.22. Clans formed by family trees.....	83
Figure 6.23. Average number of applications per gem.....	88
Figure 6.24. SEA and Gem methods' runtimes (Random seeds).....	89
Figure 7.1. Fitness boost during process.....	92

List of Tables

Table 2.1. Comparison of EA and Random-search results.	10
Table 3.1. Flip and Hop gem-match probabilities, in different problem sizes.....	28
Table 4.1. Data dimensions and ranges for analysis.	33
Table 5.1. Gap between SEA and Gem max fitness, in terms of σ	52
Table 6.1. Ratio of applications per Flip / Hop gem.....	80
Table 6.2: Gem switch-off stages for Hybrid process.....	81
Table 6.3: Percentage of individuals created by gems, at different stages.	86
Table 6.4. Ratio of applications per Flip / Hop gem, by Population-size.....	87
Table 6.5: Probability of matching a gem after 1 mutation, and after 10 mutations.	87
Table 6.6. Gem / SEA run-times per graph size.	90
Table 6.7. Selected comparisons of SEA and Gem run-times.....	90
Table 9.1. Distances (miles) between US cities, as the crow flies.....	103

Table 9.2. Order-of-complexity workings for the SEA.	105
Table 9.3. Order-of-complexity workings for Gems.	106

List of Algorithms

Algorithm 4.1. The Flip match algorithm.	40
Algorithm 4.2. The Hop match algorithm.....	42
Algorithm 4.3. Loops to run all experiments.	46
Algorithm 9.1. Efficient calculation of offspring's fitness.....	102
Algorithm 9.2. Efficient fetching of arc's cost.	103

Abstract

This document presents *gems*, a novel method to augment standard EAs (Evolutionary Algorithms). A gem is a record of a good mutation, stored so that it can be passed to unrelated individuals *horizontally* rather than by vertical descent.

We identify general problem characteristics to indicate when gems may be used, develop and define models and algorithms, implement a proof-of-concept system to compare the new method with standard EAs, explain gem dynamics, and propose a theory to explain their behaviour. Results show that gems more than double the fitness improvements of standard EAs. Furthermore, the harder the problem instance is, the larger the benefit is.

Runtime comparisons show that the method's fitness improvements far outweigh the performance costs.

1. Introduction

1.1 Motivation

This dissertation is motivated by a series of lectures on the MSc in Business Analytics course, entitled "Natural Computing Algorithms" (McGarraghy, 2014).¹ The lectures described an eclectic set of Machine Learning heuristics that are inspired by nature. EAs (Evolutionary Algorithms) are introduced as an important branch of Computational Intelligence "*which use computational models of some of the known mechanisms of evolution*". They have proved useful in many applications such as optimisation problems, just as natural evolution finds well-fitted solutions from a huge space of possibilities.

The lectures prompted the question: "Can EAs improve on natural selection?" This led to the gem idea, which this thesis develops and explores. We identify general characteristics of problems in which gems may be applied to good effect, and implement a proof-of-concept to compare the method to a traditional EA. We report results and try to explain them.

This introduction outlines the standard EA optimisation heuristic, gem concepts, and the TSP (Travelling Salesman Problem), which was selected as the proof-of-concept test problem.

1.2 The Evolutionary Algorithm heuristic

1.2.1 Overview

EAs have been prominent among Machine Learning heuristics since the 1960s², and are classified as one of the "five tribes" of Machine Learning (Domingos, 2015). They are inspired by the Neo-Darwinist theory of evolution, based on natural selection (Darwin, 1859) and Mendel's Laws of Inheritance (Mendel, 1865). EAs

¹ Now published as a book (Brabazon et al., 2015).

² Fogel, *Evolutionary programming*, 1960. Rechenberg's 'evolution strategy' optimization, 1971.

use analogous mechanisms to incrementally improve the fitness of a population of individuals.

While Neo-Darwinism focuses on living individuals, the individuals in EAs are possible solutions to a problem. A fitness function measures the quality of an individual solution. The function is specific to the problem. In TSP, for example, each individual is a tour of N nodes (cities), and fitness may be defined as the reciprocal of the tour's length.

The EA generates a sequence of increasingly fit individuals until it reaches a solution that is fit enough. It may not be the *fittest*: EAs follow the satisficing principle.

EAs are particularly useful in problems where:

- a. There is no known formulaic approach to compute an optimal solution in reasonable time.
- b. The search space is too large to exhaustively evaluate all solutions in reasonable time.
- c. No greedy heuristic is available to find a satisfactory solution (although a greedy heuristic may provide a useful "head start" to a solution).

EAs are important because many problems have these characteristics.³

Combinatorial problems are a classic example, and ubiquitous in practical applications.

1.2.2 The EA heuristic has an Achilles Heel

Natural evolution has an Achilles Heel: it selects based on a complete individual's fitness, rather than the lower level of genes. This raises two contrasting issues:

1. **"Losing the good with the bad"**. An unfit individual may have one particular strong feature. This individual is unlikely to have many descendants, if any. Thus, the strong feature dies with the individual, and is lost to the gene pool.

³ For example, see "Natural Computing" course notes §9.2 (McGarraghy, 2014).

2. **"Keeping the bad with the good"**. A fit individual may have one particular weak feature. This individual is likely to have many descendants. Thus, the weak feature can propagate in the population's gene-pool.

Inspired as they are by natural selection, EAs suffer from the same issues.

Gems are intended to solve the first issue, so making standard EAs more efficient by saving and propagating mutations, even if the original individual does not survive.

1.3 Gems

Standard EAs use *mutations* to explore new solutions. A mutation is a small, random change to an individual's genes. This usually changes the individual's fitness, for better or worse.

A *good* mutation is one that improved fitness. Gems work by identifying good mutations and storing the best mutations, *outside* any individual. These stored mutations (*gems*) are available for use on other, unrelated, individuals. In a standard EA, mutations can only be passed to other individuals by vertical inheritance.⁴ In contrast, gems propagate *horizontally* across a population.⁵

A gem may only be applied to individuals that closely match the original individual that the mutation worked on. If the individuals are similar, it is reasonable to expect that the mutation will have similar effects. Otherwise, there is no such expectation.

1.4 The TSP (Travelling Salesman Problem)

We selected the TSP as our test problem, so a brief description of the TSP is in order.

⁴ *"Descent with modification"*, in Darwin's terms.

⁵ Bacteria can pass their genes horizontally, to unrelated peers. This is how superbugs' immunities spread so rapidly. However, a population of bacteria cannot store genes *outside* any of living individuals – it has no gems.

Figure 1.1 shows a tour of the fifty USA state capitals. A *tour* starts at any city, visits each city only once, and returns to the start. The TSP is the problem of finding the shortest possible tour. This is a combinatorial problem, with $50!$ valid solutions.⁶



Figure 1.1. A tour of 50 USA state capitals.

In EA terms, each tour is an individual. Fitness is measured as the reciprocal of the tour's distance: the shorter the distance, the fitter the individual.

In the TSP, a mutation is a small, random change to the sequence of cities visited.

Figures 1.2 and 1.3 show a section of a tour, before and after mutation.



Figure 1.2. Map A: Sections of a tour of USA cities.⁷

⁶ About 3×10^{64} tours. Or half that if traversing a route "clockwise" and "anticlockwise" is counted as the same tour (on a bi-directional graph).

⁷ Maps thanks to free download from Maps of World at <http://www.mapsofworld.com/usa/usa-outline-map.html>.

Figure 1.3 shows the effects of a mutation on the sections in Map A. It reduces the total tour distance, and thus increases the individual's fitness.



Figure 1.3. Map B: The tour sections in Map A changed by mutation.

The rest of the tour is unchanged, and irrelevant here. The mutation yields the same improvement to *any* tour that contains the original sections. If we find another tour with those sections, and apply this mutation again, it would yield the same improvement again.

This is exactly what a gem does: it records a good mutation so that it can be reapplied to other individuals. Gems may be a double-edged sword: they improve fitness, but they also reduce diversity by inserting the same arcs (node-to-next-node paths) into many tours. This loss of diversity is a recurring theme in this dissertation.

1.5 Document structure

This document sets out to investigate the gem concept and its workings, and compare it to traditional EAs. It is a voyage of discovery: a series of questions and answers leading to more questions. Some answers provide insight, others do not.

Chapter 2 presents an introduction to EAs, with a review of relevant literature.

Chapter 3 describes the generic process, models and design, using the TSP to illustrate the concepts.

Chapter 4 describes the proof-of-concept design and experimental framework.

Chapter 5 is an initial investigation of the experiment's results. Are gems beneficial? How do certain factors effect the results? Phase 1 results are used to revise models, raise further questions, and shape Phase 2 investigations.

Chapter 6 reports on Phase 2 investigations, attempts to understand and explain the method's behaviours, and attempts to address an issue.

Conclusions and further research suggestions are gathered in Chapter 7, followed by a selected bibliography and appendices.

A note on the data and charts presented.

Most results are presented as charts, because a picture paints a thousand words. Space limitations preclude presenting charts for each of the hundreds of experimental cases, therefore only the most relevant cases are presented. However, the trends and observations apply to the other cases too.⁸

⁸ The complete data is available in a set of CSV files, so that readers can check the data, and make their own charts and calculations.

2. Literature review – an introduction to EAs

2.1 Overview and introduction

2.1.1 Methodology

Our research started in Google Scholar and Wikipedia, and extended from there. The search used various combinations of at least three of these terms:

Genetic Algorithms, Evolutionary Computation, Mutation, Cross-over, Recombination, Exploration, Exploitation, Diversity, Schema, Building-blocks, and Complexity.

We conducted a separate search for TSP-specific literature, using:

TSP, Genetic Algorithms, Evolutionary Computation, and Mutation *or* Representation.

2.1.2 Overview of EAs

The works of Darwin (1859), Mendel (1865), Dawkins (1976), and others provide a contextual understanding of biological evolution.

The course notes provided a high-level overview of EAs, including the analogues with nature, the main operators, typical algorithm templates, parameters, examples, and issues (McGarraghy, 2014).

Evolution Strategies (Hansen et al., 2015) provides an excellent overview of EAs, in more depth than was possible in the lectures. It describes many aspects used in the proof-of-concept, including sample algorithms, mutation operators, parameter-control by self-adaptation, and convergence. However, it focuses on problems in *continuous* search spaces, so while the conceptual understanding is beneficial, most of the detail is not applicable.

An Introduction to Genetic Algorithms Theory and Applications (Wainwright, 1993) is more a tutorial than a review. We found it useful to dip into.

Many papers provide a brief history of EAs, and many credit John Holland as the founder. We note that others had developed EA concepts before Holland, notably Fogel (1960, 1966) and Rechenberg (1971). Holland (1975) brought a new emphasis on cross-over (previously, *mutation* was the operator of interest). He also explained

the balance of *exploitation* and *exploration* (especially important in multi-modal fitness landscapes), and developed his *schema* concept to provide the leading theoretical description of how EAs work. Both the exploitation/ exploration balance and the schema concept are of interest for the gem method.

2.1.3 The high-level process

We start with an outline of the EA process as shown in figure 2.1. This provides a framework for the literature review.

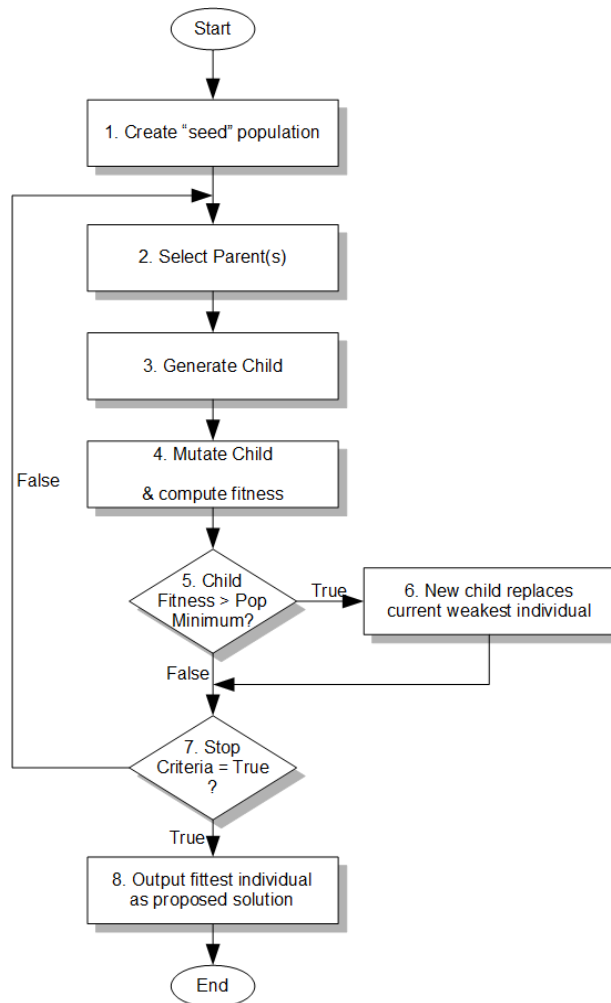


Figure 2.1. Evolutionary Algorithm flowchart

2.1.4 Outline of the steps

Step 1: The seed population acts like a "founding population" in nature. It is a collection of individuals, each of which is a possible solution to the problem.

Steps 2 to 6 are repeated until the stopping criterion is true.

Step 2: Select a parent(s) to generate a child. Several selection rules are available. All reflect natural selection in that the fitter an individual is, the more likely it is to be selected as a parent.

Step 3: Since the parent was selected by Step 2 rules, it is likely to be fitter than average, and since the child inherits its parent's genes, it is likely to be fitter than average too.

Step 4: Apply a mutation operator to the child. Calculate the mutated child's fitness.

Steps 5 & 6: Compare the new individual with the weakest (least fit) individual in the current population. The fitter of the two survives, and the weaker is discarded. The population size remains constant. Again, the natural analogies are striking. Although natural populations wax and wane, the size is limited by scarce resources (Malthus, 1798). On average, the fitter survive and the weaker are replaced.

Steps 7 & 8: If stop criteria are true⁹, then stop and output the fittest individual as the suggested solution. Otherwise, repeat steps 2 to 6.

2.1.5 The value of EAs

The premise for this thesis is that the EA heuristic is good for solving some kinds of problems. As a simple demonstration of the value of EAs, figure 2.2 compares results using the proof-of-concept EA, with those of a Random-search technique. The charts show EA and Random-search maximum fitness as blue and purple lines respectively. Each point is the mean of 400 observations.

⁹ Biological evolution has no stopping rules.

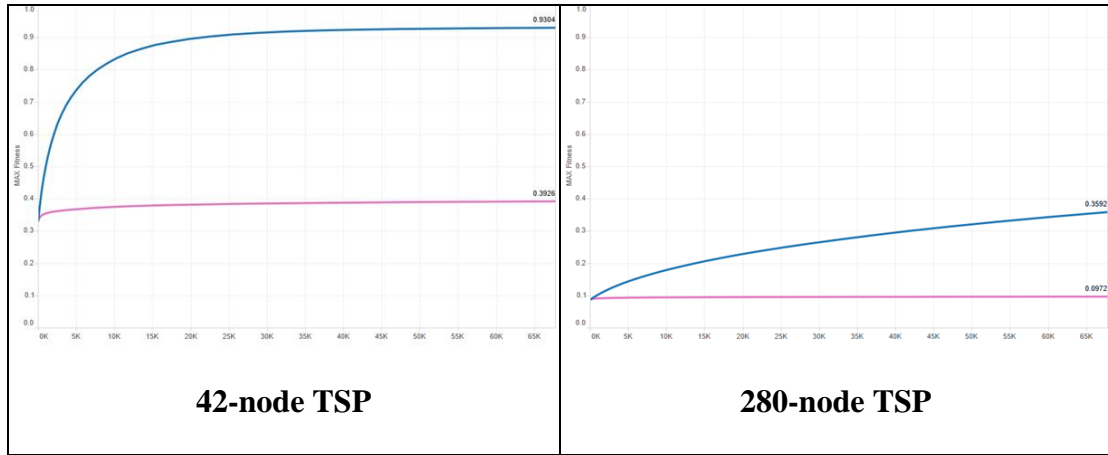


Figure 2.2. Comparison of Evolutionary Algorithm with Random Search.

In each problem, both techniques start at the same initial maximum fitness. Maximum fitness clearly increases as the EA proceeds, whereas it barely increases at all using the Random-search. The figures are:

Graph	EA max. fitness increase	Random-search increase
42-node	+180% (from .33 to .93)	+18% (from .33 to .39)
280-node	+308% (from .09 to .36)	+10% (from .09 to .10)

Table 2.1. Comparison of EA and Random-search results.

2.2 Specific EA topics

2.2.1 Population size

There is abundant literature on population size. The consensus is that large size helps diversity, while small size leads to faster convergence – although, because small populations have less diversity, it may be *premature* convergence, especially in a multi-modal fitness landscape. Small populations are also (a little) less computationally expensive.

Haupt (2000) reports experimental evidence that small population sizes with high mutation rates, give better results than large populations with small mutation rates. He recommends population size of <16 chromosomes (individuals).

Koljonen and Alander (2006) find that "*decreasing population size increases optimization speed to a certain point, after which premature convergence slows the optimization speed down*".

Gotshall and Rylander (1992) state: "*optimal population appears to grow logarithmically with respect to the instance size*" They experimented on the optimum population size for various instance sizes and "*show that increasing the population size increases the accuracy of the GA. Increasing population size also causes the number of generations to converge to increase. The optimal population for a given problem is the point of inflection where the benefit of quick convergence is offset by increasing inaccuracy*" – i.e. the optimum size depends on the problem.

Roewa et al. (2013) find that "... *smaller populations result in lower accuracy of the solution, obtained for a smaller computational time. The further increase of the population size increases the accuracy of solution ... to a population size of 100 chromosomes. The use of larger populations does not improve the solution accuracy and only increase the needed computational resources.*" They settle on population size of 100 as ideal, but this is not generalizable, being for their particular EA and problem domain (cultivation modelling).

Vrajitoru (1999) searched for "*the optimal balance between two genetic parameters: the population size and the number of generations.*" The focus is on computational complexity, but again it is in a particular domain (document indexing).

2.2.2 Random and Greedy seed populations

We focus on randomly-generated seed individuals because not all problem domains have greedy solutions, and an EA provides more improvement on random solutions than on greedy ones. We include greedy seed trials for some comparisons, so some research in this area was in order. Several authors have studied greedy solutions, and found that they lead to faster convergence. This is not surprising, since greedy solutions are closer to the optimum from the start. However, they are computationally more expensive to create.

Deng et al (2015) propose a new greedy solution for TSP, using k-means clustering. They report that this gives significantly better solutions "*in the same running time.*" This may seem surprising, since clustering is computationally expensive. However, it is only used to create the seed population, not within the "generations" loop.

Oman and Cunningham (2001) use a case-base to generate high-fitness seed-individuals. They experimented on both TSP and JSSP (Job-Shop Scheduling

Problem). They found that generating 100% of the seed individuals greedily "*causes the GA to converge far too quickly on poorer solutions.*" Seeding 25% greedily, resulted in a 25% reduction in number of generations, as did a 50% greedy seed population. When 75% are seeded greedily, there is a 37% reduction – returns seem to be diminishing. They note that the cost of the greedy seeding is "*a negligible fraction of the total run time*" – we presume because it is only done once.

Grefenstette et al. (1987) used a greedy TSP approach to good effect. Their results showed rapid fitness increase in the early stages, which then levelled off.

2.2.3 Parent selection

Parent selection is the principal *exploitation* step, when fitter individuals are more likely to be selected. The greater the advantage that higher fitness confers, the higher the *selection pressure*. If selection pressure is too high, unfit individuals are replaced too quickly, with rapid loss of diversity. This leads to premature convergence.

Much has been written on the various selection techniques. Holland (1975) developed the fitness-proportionate roulette model, in which an individual's probability of selection as a parent, is directly proportional to its fitness. *Ranked-roulette* is similar, except that the probability is determined by the individual's fitness *rank* in the population. Tournament-selection is also based on rank: a (small) number of individuals are randomly selected for the tournament, and the highest-ranked individual in the tournament is selected as a parent.

Most researchers use multiple-parents with cross-over operators, usually with *two* parents.¹⁰ Their genes are recombined to generate a child that inherits from both (similar to sexual reproduction in nature). Since the parents are likely to be fitter than average, and the child inherits its parents' genes, the child is also likely to be fitter than average.

Blickle and Thiele's paper, "A Mathematical Analysis of Tournament Selection." (1995), focuses on real-valued objective functions. They also outline a little-known alternative: "Truncation selection". We note that they assume "*a Gaussian initial*

¹⁰ In principle there may be more than two parents, but we ignore this.

fitness distribution", which is not always the case¹¹. We find their measure of selection pressure dubious: *"the change of the average fitness of the population"*. Nonetheless, they prompted us to reduce tournament size to 5: *"Often tournaments are held only between two individuals ..."* and *"... loss of diversity is independent of the initial fitness distribution ... the number of individuals lost increases with tournament size. About half of the population is lost at tournament size $t = 5$."*

Cantu-Paz (2000), De Jong (1975), Koljonen et al. (2006), and Xie and Zhang (2009) all provide theoretical and experimental results on various selection techniques, or parameters to tune one technique (often tournament), with and without generation gaps¹², to control the impact on diversity.

Deep et al. (2011) describe a parent selection and replacement technique that selects two parents, generates two children, and keeps the best two of the four individuals. This seems counter-intuitive, since if three of the four are highly fit (compared to others in the population), only two survive, and if only one is fit, still two survive. That may be the point: the low selection intensity preserves diversity.

Several authors use techniques to ensure that the mating parents are not too similar. Eshelman and Schaffer's title is self-explanatory: "Preventing premature convergence in genetic algorithms by preventing incest" (1991), and Ting and Buning's well known Tabu selection (2003) also prevents incest.

Whitley and Kauth's *Genitor* algorithm (1988) used Steady-State rather than Generation Gap replacement, and rank-based selection – both now widely accepted, but less so at the time. They assert that ranking *"provides a degree of control over selective pressure that is not possible with fitness proportionate reproduction"*. This seems to be a general consensus, but we are not fully convinced.

We note in passing that tournament selection has lower order of complexity than roulette selection, but none of the literature we found mentioned this.

¹¹ See Figure 4.6 for example.

¹² See §2.2.10.

2.2.4 The fitness function

A fitness function maps the search space to \mathbb{R} . Each individual solution maps to one real number:

$$f: \text{Search space} \rightarrow \mathbb{R}, \quad f(\text{Individual}) = x \in \mathbb{R}.$$

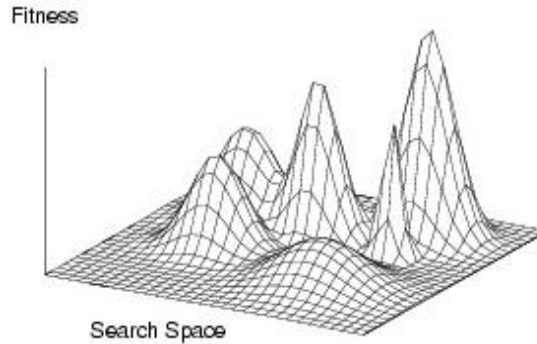


Figure 2.3. A solution fitness landscape.

Some optimisation problems aim to minimise an objective function (e.g. cost), others aim to maximise one (e.g. profit). We aim to *maximise* fitness.

We found surprisingly little research on fitness functions per se, given their importance in parent selection and replacement, and their impact on an EA's time-complexity. Perhaps this is because the function is often obvious given the objective, with few alternatives. For example, in TSP it must be based on the length of the tour, and in the Knapsack-packing problem, it is the value of the packed items.

2.2.5 Generating offspring

Most modern EAs select two parents, and use *cross-over* to combine and mix their genes to generate a child. Since gems are based on mutation rather than cross-over, the proof-of-concept does not implement cross-over.

2.2.6 Mutation

A mutation makes small change(s) to an individual's chromosome, leaving most of it unchanged. Fitness usually changes as a result, but not by much. In nature, mutations are caused by errors in copying a parent's genes, and are usually seen as a

negative phenomenon. In EAs, mutation is beneficial because it enables random exploration of a search space, to check solutions that would otherwise be missed.

At a high level mutation is generic, but the details are necessarily domain specific, both in conceptual models and implemented representations.

Abdoun et al. (2012) provide a survey of TSP mutation operators, with representation suggestions, and report efficacy and time complexity results.

Baudrey et al. (2005) investigate mutation operators in software testing EAs, which made us more conscious of generic aspects beyond the TSP domain.

Haupt (2000) favours high mutation rates. This opens a broader discussion on the comparative merits of cross-over vis-à-vis mutation. Fogel et al (1966) used mutation only, whereas Holland (1975) championed the benefits of cross-over. The debate continues, as Osaba et al.'s paper "Cross-over versus Mutation: A Comparative Analysis ...", published thirty-nine years later, shows. Osaba et al. also observe that cross-overs are *"complex operations in which two individuals combine their characteristics. On the other hand, a mutation is a small modification of a chromosome and requires considerably less time..."*

2.2.7 TSP mutation operators

TSP is a famous problem, and there is extensive literature on its chromosome representations and mutation operators (often more technical than conceptual).

Abdoun et al. (2012) provide a good comparison of TSP mutation operators' performance. Ferreira (2002) presents her findings on the well-known inversion mutation. Deep et al. (2011) propose a new mutation operator, and although we don't use it, their discussion on representations was helpful. Larrañaga et al. (1999) provide a good review of TSP representations and operators. This is more than a mere catalogue – it discusses the pros and cons of various alternatives.

Other authors, while not focused on representations, provide good descriptions that confirmed us in our choices. Grefenstette et al. (1985) describe alternative TSP chromosome representations, Oman and Cunningham (2001) describe their tour representation, and Osaba et al. (2014) present a range of mutation operators.

2.2.8 Other problem domains

A search on Google Scholar shows that the TSP is just one of many NP-hard problems within the graph-network domain¹³, and that the broader combinatorial family includes many more that are not graph-based¹⁴, and more again beyond combinatorial (for example continuous search spaces, and the Super Mario Brothers game!).

Research in such problem domains provides a deeper appreciation of representations and guided our thoughts on generic gem models. These include Bessaou and Siarry (2001), Chiong and Beng (2007), who also describe difficulties in finding efficient representations for operators, Deng et al. (2015), Falkenauer and Delchambre (1997), Herreraa et al. (1998), and Osaba et al. (2014).

Some problems may not be amenable to the gem method. Section 3.7 outlines the characteristics that make a problem amenable to the method (assuming it is amenable to EA in the first place).

2.2.9 Replacement

In contrast to parent selection, replacement strategies may be dubbed "loser selection", in which weaker individuals are more likely to be selected.

Elitism allows fit parents to out-survive younger generations. There is broad consensus that this is beneficial. Oliveto et al. (2007) note that "*Rudolph (1994) proved that canonical GAs ... [without elitism] ... do not converge to the global optimum, while elitist variants do*".

Cantu-Paz (2000) reviews a range of replacement strategies including: (a) insert offspring at random, (b) replace the worst individuals, (c) choose using the selection algorithm used to select the parents (e.g. tournament), (d) delete the oldest (FIFO), and combinations or elitist versions of these. While FIFO strikes us as perverse, at least we were aware of options that we had not previously contemplated.

¹³ Snowplough, Postman, and Capacitated-vehicle, for example.

¹⁴ Bin-packing, Knapsack, and Scheduling, for example.

Smith and Vavak (1999) provide a similar list of strategies, for steady-state EAs. Their experiments "*confirm that without enforced elitism Replace–Random and Replace–Oldest strategies cannot guarantee takeover*". This seems to be a tautology: "Replace-Random" and "Replace–Oldest" seem to directly preclude elitism.

Several authors focus on how replacement strategies impact diversity. Most agree that, while the strategies are a factor in selection pressure, they are not as important as parent selection. The title of Herreraa et al.'s 2008 paper speaks for itself: "Replacement strategies to preserve useful diversity in steady-state genetic algorithms."

Goldberg and Deb (1991) observed that Genitor has high selective pressure *even when parents were selected randomly*. Therefore, in contrast to others, they suggest that Genitor's replacement strategy (replacing the weakest individual) was the main factor in selection intensity.

2.2.10 Steady-state and Generation-gap replacement models.

The *Generation Gap* model replaces the whole population at each generation, much like annual mayfly populations in nature. Parents are selected to generate offspring, and the fittest offspring become the next generation. No parents survive, so elitism is impossible.

In contrast, the *Steady State* model generates one offspring at a time, and replaces one (weakest) individual at a time. This provides elitism: survival depends on fitness, regardless of age. In nature, human populations behave much like this.¹⁵

There is considerable research comparing these models, but little clarity on the factors that might favour one or the other. Most authors simply pick a model, then conduct their experiments within that model. Cantu-Paz's paper (2000), for example, is entitled "Selection intensity in genetic algorithms with generation gaps."

Rogers and Priigel-Bennett (1999) find that "*loss of variance under Steady-state (drift) is twice the rate of loss in Generational*". We note that loss of *fitness* variance

¹⁵ Age *indirectly* effects elitism, but only because it effects fitness.

(as opposed to diversity) is not necessarily bad, as long as fitness is increasing. Syswerda (1999) conducted a direct comparison, as did De Jong and Jayshree (1992). We leave the last word with De Jong: "*the pros/ cons of overlapping generations remains a somewhat cloudy issue.*"

2.2.11 Stopping criteria

EAs typically stop on any of these criteria: (a) an acceptable level of fitness is achieved, (b) a certain number of iterations is reached, or (c) convergence, i.e. fitness has not increased (much) over the previous X iterations.

We found little literature on stopping rules, perhaps because they are quite simple, and obvious. Bhandari et al. (2002) suggest using variance (σ^2) of the sequence of maximum fitnesses as a stopping criterion. The EA stops when $\sigma^2 < \epsilon$ (small). We think this gives too much weight to early fitnesses (which increase rapidly): convergence should be based on fitness changes over *recent* iterations.

2.2.12 Schemas

Gems may throw light on the dynamics of Holland's schema theory (1975). Schema theory explains how EAs work by searching for chromosomes that contain high-performance subsets of genes, called *schema*, that combine in building blocks (which may be hierarchical). Each schema is a group of genes that collectively produce a positive effect.

Schema are likely to be found in high-fitness chromosomes. They propagate exponentially over iterations.

Gems are records of good mutations that act on smallish sets of genes to generate high-fitness chromosomes. Gems propagate through the population. Gems have parallels with schema, and therefore may help to identify them.

Mitchell, Forest, and Holland (1991) provide experimental results that support Holland's theory of schema.

Goldberg (1995) describes low-fitness regions as barriers in the fitness landscape, blocking¹⁶ small-step movement across that region. Cross-over produces larger changes, often large enough to cross low-fitness "valleys" too wide to cross in small steps. However, our proof-of-concept does not use cross-over. We expect that the cross-over operator destroys many schema present in parent chromosomes. The child may contain new schema, but the lack of continuity makes it hard to track the dynamics. Note that researchers often propose new mutation operators that combine smaller mutations, resulting in larger "steps" (for example Deep et al. (2011)).

2.2.13 Preserving diversity

Holland (1975) showed that an EA combines exploration and exploitation in an optimal way: cross-over and mutation promote exploration, while selection of parents, and of replacement survivors, promote exploitation. Holland emphasized cross-over over mutation, perhaps because it was new at the time. Yet the two operators' comparative merits are still debated, as Osaba et al.'s 2014 paper shows.

Since over-exploitation leads to loss of diversity, and hence to premature convergence, there has been much research on preserving diversity. Most of it focuses on parent selection methods, which are usually seen as the primary exploiting phase. Conversely, as already noted, Goldberg and Deb (1991) suggest that *survivor* selection is more important than is usually appreciated.

Gupta et al. (2012) provide a reasonable review of methods to preserve diversity. Friedrich et al.'s review (2009) is better, with deeper comparison and analysis of the mechanics.

Many suggestions are based on restrictive selection of mates. For example, De Jong (1975) introduced the *niche* concept, which clusters similar individuals in mating pools. There is little diversity *within* a niche, but diversity *between* niches is preserved. Jassadapakorn et al. (2011) have a similar theme: "*regard the population as a multi-racial society where a group of similar chromosomes represents a race. When recombination occurs within a race the diversity ... will be low ... when*

¹⁶ Not *Building*-blocks.

recombination occurs between different races the diversity will be high. To control the diversity, the selection criteria for mating include the difference function which measures dissimilarity of two individuals". Eshelman and Schaffer (1991) maintain diversity by preventing "incest", and we have already mentioned Ting and Buning's Tabu parent search, which prevents incest. Shimodaira (1997) presents "DCGA: A Diversity Control Oriented Genetic Algorithm" which uses the Hamming distance between chromosomes (as well as fitness) to select "out-breeding" pairs – the larger the distance between individuals, the more likely they are to be mates.

Grefenstette (1987) applied similar concepts to constructing seed populations (rather than parent selection), giving preference to chromosomes not already represented in the population.

Goldberg and Richardson (1987) propose a fitness sharing technique, in which a chromosome's fitness depends on that of its neighbours – again raising similarity-clustering functions. Collingwood et al (1996) are more radical, proposing multiploidy as a way to preserve diversity.¹⁷ They write: *"unused genes remain in a multiploid genotype, unexpressed, but shielded from extinction until they may later become useful"*. This may seem similar to the gem concept on first reading, but it is not.

We note that the clustering, similarity, and dissimilarity functions¹⁸ proposed by several authors, are computationally expensive. A few acknowledge the overhead (e.g. Collingwood: *"better overall results would have to come at the expense of extra computational time"*), but none seems averse to it.

We also note that some authors seem to wrongly equate diversity with variance of fitness function. Diversity relates to the "horizontal" distances between individuals in a search space. Individuals may be far apart (and thus diverse), yet at approximately the same "fitness contour", resulting in low variance. High variance may be sufficient to show diversity, but it is not necessary.

¹⁷ Storing multiple copies of genes. This is widely observed in nature, for example in plants.

¹⁸ Such as Hamming distance.

2.3 Summary

This chapter presented the results of our literature research, organised around EA concepts and steps. This helped guide our thinking and experimental design: we stand on the shoulders of giants.

We discovered no suggestion of anything like gems. The next chapter describes the new method, how it works with Standard EAs, general design features, and potential issues.

3. Gem concepts and models

3.1 Introduction

This chapter presents an overview of gem concepts, processes, and models. This is followed by more detailed and technical descriptions. It provides the basis to understand the proof-of-concept design, and to interpret the experimental results.

3.2 The concept of a gem

3.2.1 Some mutation terminology.

The term *mutation* is used in different ways, so some definitions are in order.

Definitions:

A mutation *function* maps one individual to another individual in the same search space. We introduce the notation I for the original individual, and I' for the mutated version of I , as follows:

$$m(I) = I' \in \text{search space.}$$

A mutation *instruction* is a mutation function with particular parameters.

A mutation *instance* is the application of a mutation instruction to a particular individual.

A mutation instance's *cost change* is the difference between the post- and pre-mutation individuals' costs.¹⁹

$$\delta C \equiv c(I') - c(I). \quad \delta C < 0 \text{ indicates a cost reduction, which is good.}$$

A mutation instance's *fitness change* is the difference between the fitness of I , and that of I' :

$$\delta F \equiv f(I') - f(I). \quad \delta F > 0 \text{ indicates a fitness increase, which is good.}$$

A *good* mutation instance is one that increased fitness, i.e. $\delta F > 0$.

¹⁹ TSP distance = cost. Cost is a more general term, used in many problem-domains.

3.2.2 *Definition of a Gem.*

A gem is a record of a good mutation, stored in such a way that it can be applied to other individuals. More precisely, it is a record of a mutation *instruction* that produced one of the Top N gains. Most good mutations are not gems because, although they have positive value, it is less than the lowest Top N value.

3.2.3 *How gems work*

Gems work by recording the best mutation instances and re-applying them to other individuals, thus propagating the good effects of the original mutation instances. This overcomes the standard EA's "Losing the good with the bad" issue (§1.2.2).

3.2.4 *Definition of a jewellery-box*

A *jewellery-box* is a container of gems, plus the functionality to manage its gems. Its relationship to its gems parallels that between a population and its individuals.

Jewellery-box functionality includes: creating gems, matching and applying gems, and deleting gems.

3.3 **Individual compatibility.**

A biological mutation may be useful for one creature, but not for another. For example keener eyesight is useful for a magpie, but not for a mole.

Similarly, a mutation instruction has different effects on different individuals. For example, the mutation instruction shown in figure 1.3 moved the city that came after Los Angeles, to come after New York instead, and joins Los Angeles to the node that originally came after Boston. However, applying this instruction to a different tour, in which the city following Los Angeles is San Francisco rather than Boston, would move San Francisco to come after New York – not such a good result!

Therefore, before applying a gem to another individual, we should check that it is a good fit for that individual. This is an important concept, which we call *compatibility*.

Definition: A chromosome (individual) is *compatible* with a gem if it has the same alleles (gene values) as the chromosome in the original mutation instance, for the *relevant* genes.

In the example in figure 1.2 (*before* mutation), we check to see if the new individual has the following three arcs:

Los Angeles \rightarrow Boston, Boston \rightarrow San Diego, and New York \rightarrow Philadelphia.

Any tour that contains these three arcs is *compatible* with the mutation instruction, regardless of the rest of the tour. We can apply the gem's mutation to the new individual in reasonable expectation of a similar gain. In TSP we are guaranteed to have *exactly* the same cost reduction, but this may not be true in other domains.

3.4 The process steps

Mutation lies at the heart of the gem method, so it is not surprising that the method is contained within the mutation step of the Standard EA (Step 4).

Figure 3.1 shows how the process works. The picture in the centre represent the jewellery-box, but is not strictly part of the flowchart.

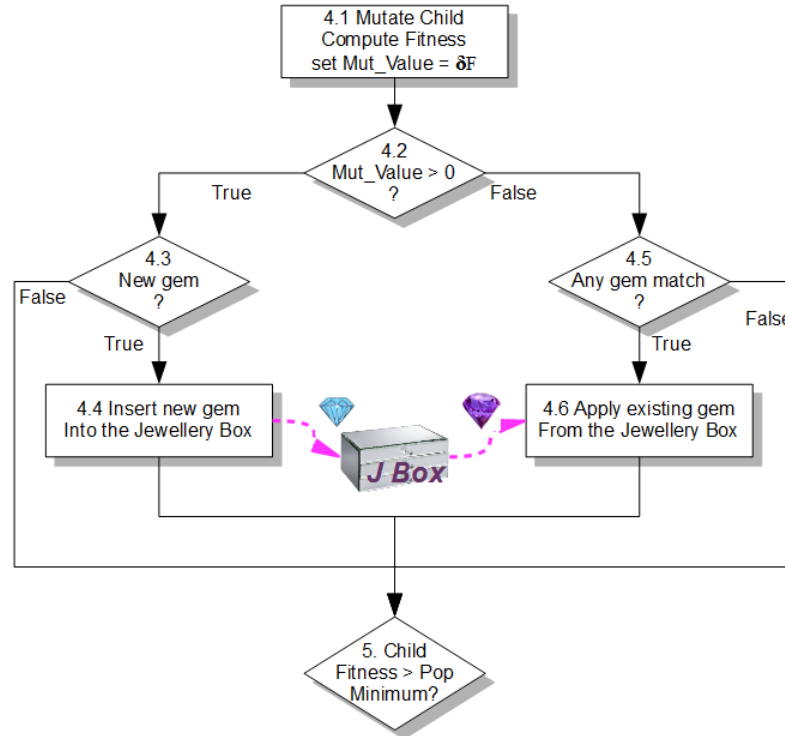


Figure 3.1. Gem functionality within EA's "Mutate Child" step

After calculating a mutated child's fitness, the mutation's value is calculated as:

$$\text{Mutation.Value} = \text{Fitness(I')} - \text{Fitness(I)}.$$

The jewellery-box identifies a high-value mutation instance, creates a gem, and stores it for future use (LHS²⁰ in figure 3.1). Then it applies the stored gem to compatible individuals (RHS).

LHS: If the mutation's value > 0 , then it is a *candidate* new gem.

4.3: The jewellery-box checks that (a) it has spare space for a new gem, *or* (b) any of its gems has lower value than that of the candidate gem.

If so, a new gem is created and inserted into the jewellery-box. This will either (a) increase the number of gems in the jewellery-box, *or* (b) replace the lowest-value gem in the full jewellery-box.

RHS If the mutation's value < 0 , then an existing gem may be applied.

The jewellery-box searches for a gem compatible with the individual (4.5).

If it finds a matching gem, it applies that gem to the individual (4.6).

At this point the individual is either (i) fitter than its parent due to application of a gem, (ii) fitter due to a random mutation, or (iii) less fit than its parent.

The jewellery-box has finished its work, and the SEA (Standard EA²¹) process resumes at step 5 (replacement).

3.5 Gem representation and functionality

Mutation functions are specific to a problem domain (e.g. TSP rather than the Knapsack problem). Therefore the internal algorithm and representation details are domain-specific.

Each gem contains three groups of variables:

²⁰ Left Hand Side. RHS means Right Hand Side.

²¹ From this point forward, we abbreviate Standard EA to SEA.

- a. Values used to test if an individual is compatible with the gem. These are specific to the problem domain, e.g. TSP.
- b. The original mutation instance's parameter values, so that it can be re-applied as an instruction. These are also domain-specific. For TSP, they are a sub-set of the compatibility-test values, double-jobbing for a different purpose (gem application).²²
- c. Control and book-keeping values such as the number of times that the gem has been applied. These are generic to all domains.

Gems also include two functions:

- a. **match(*individual*)** tests whether the individual is compatible with the gem or not. It returns True if the individual is compatible, otherwise it returns False.
- b. **apply(*individual*)** applies this gem to the individual. This is only allowed if match() returns True.

3.6 Probability of matching depends on the mutation-type.

The probability of matching an individual to a gem is critical. If it is too low, gems may be created, but will not be applied often enough to have a significant impact.

We start with matching a Hop mutation²³, using the USA cities example to explain the workings. A Hop mutation involves 3 arcs. In the example, these are:

A. [Los Angeles → Boston → San Diego]	B. [New York → Philadelphia]
OR [Los Angeles ← Boston ← San Diego]	OR [New York ← Philadelphia]

"OR" is required because the mutation will yield the same gain if the arcs' directions are reversed (the graph is *bi-directional*). The arcs in A. and/ or B. can even be reversed independently, giving $2 \times 2 = 4$ permutations for which the mutation instructions will work.

²² We believe this may be true for many problem domains, although not for all.

²³ Also called "Insertion mutation" (Fogel, 1988).

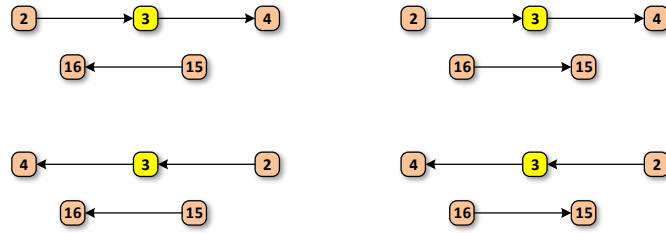


Figure 3.2. Four permutations of matching Hop gems.

The probability that the arc starting at Los Angeles, ends at Boston (of all the $N-1$ possible end-points) is: $1/(N-1)$.

The probability that the arc starting at Boston, ends at San Diego (of the $N-2$ remaining end-points) is: $1/(N-2)$.

The probability that the arc starting at New York, ends at Philadelphia is: $1/(N-4)$.

Therefore the probability of matching a "Hop" gem = $4 / (N-1)(N-2)(N-4)$.

The Flip²⁴ mutation involves only two arcs (the two that are "snipped" so that the path between them can be reversed and spliced back into the tour). The probability of matching the first arc = $1/(N-1)$, and of matching the second arc = $1/(N-3)$.

The arrow-directions in a Flip must either *both* go "clockwise", or *both* go "anti-clockwise". If one arc is reversed but the other is not, it is not compatible.²⁵ A Flip gem has only two valid permutations, so the probability of matching a Flip gem = $2/(N-1)(N-3)$.

Table 3.1 shows the match probabilities for the five graphs used in the experiments.

Nodes	Flip	Hop	P(F)/P(H)
-------	------	-----	-----------

²⁴ Also called "Simple Inversion mutation" (Holland, 1975).

²⁵ We learned this the hard way. Our first "Flip match" algorithm returned true even if only one arc was reversed. Gem application split the tour into two smaller circuits, which is not a valid tour. We noticed that results included some "tour" costs that were below the known minimum.

	$1/(N-1)(N-3)$	$\times 2 = \text{match}$ probability	$1/(N-1)$ $(N-2)(N-4)$	$\times 4 = \text{match}$ probability	
21	0.00263158	0.00526316	0.00015480	0.00061920	8.5
42	0.00060976	0.00121951	0.00001605	0.00006418	19.0
76	0.00018265	0.00036530	0.00000250	0.00001001	36.5
150	0.00004535	0.00009069	0.00000031	0.00000124	73.1
280	0.00001289	0.00002579	0.00000005	0.00000019	135.7

Table 3.1. Flip and Hop gem-match probabilities, in different problem sizes.

Observations:

- The match probability may be so small that, while gems work *technically*, they have negligible practical effect. We were concerned about this, and were both pleased and puzzled to be proved wrong.
- Match probabilities decrease rapidly as graph size increases.
- Match probabilities are much smaller for a Hop gem than for a Flip gem.
- We expect gems to work best for mutation functions that only effect a few genes, because the more genes that are involved, the lower the probability of a match. Therefore "compound" mutations should be avoided. The literature contains many examples of complicated mutations, and they can usually (or always?) be broken into a short sequence of simpler mutations. For example, the Inversion²⁶ mutation flips a sub-path exactly as our Flip mutation does, but splices it back into the tour at a *different* location, somewhat like our Hop mutation. Similarly, the Exchange mutation (Banzhaf, 1990) is composed of two Hops.

Point a. was such a concern that we considered allowing gem application following a *partial* match. For example, we could allow a Hop gem application even if only two of its three arcs matched. This complicates the match rules considerably, and in general they could be much more complicated (for example, for a mutation involving

²⁶ Fogel, 1990. Not to be confused with "Simple Inversion".

five arcs, in which only any three of the five need match). Fortunately, partial matching transpired to be unnecessary in our proof-of-concept.

3.7 Generic gem considerations

3.7.1 General problem domain characteristics

Gems are intended for use in many problem domains, not just the TSP. They require certain general characteristics to work in a domain. The assumption is that the EA heuristic has been selected as a suitable method to solve the problem at hand – only then can gems provide value (the practitioner should, of course, consider other methods such as Simulated Annealing). Given that EAs *can* be applied, we believe that these characteristics are not very restrictive. They include:

- a. $\delta F = f(I') - f(I)$ can be calculated for any mutation. This is simple for single-parent children, because both $f(I')$ and $f(I)$ are known. However, for multi-parent children with cross-over, we must calculate $f(\text{child})$ before applying the mutation, then calculate $f(\text{child}')$ – an additional fitness calculation.
- b. A mutation's details can be recorded so that it can be re-applied later. This may be difficult to represent in some domains, but we believe it is usually possible with good design. However, it could be computationally expensive.
- c. Later individuals can be matched for gem compatibility (and see §3.6 d. in favour of simple mutations). Again, we believe this is usually possible, especially given b. above, although it could be computationally expensive.
- d. Suppose that mutation instance X, when applied to individual A, produced a valid individual A'. We need assurance that applying the same mutation instruction to individual B, will produce a *valid* individual B'.
- e. We may expect that a mutation instruction will yield a similar result when applied to another compatible individual. This is certainly true for TSP: if a mutation instance yields a cost-change of δC , then all other instances of the

same mutation instruction will yield the same change in cost δC .²⁷ But this is not necessarily true in other problem domains.

3.7.2 *Generic implementation*

Gems do not effect most of the EA. This facilitates good software design by allowing a neat "plug in" implementation.

The proof-of-concept implementation was designed for the TSP. The aspiration is to abstract the design of generic aspects.²⁸ This should be quite straightforward for the jewellery-box, which has no problem-domain specifics. However, designing a generic interface for gems could be more challenging, because gems are tightly coupled to domain-specific mutation operators and representations.

Our aspiration for a generic implementation is encouraged by JCLEC, a publicly available Java library for EAs. JCLEC is "*a software system for Evolutionary Computation (EC) research ... It provides a high-level software framework to do any kind of Evolutionary Algorithm...*". It also provides functionality and code examples for TSP, Knapsack, and Real-number problems.

3.7.3 *Two-parent cross-over.*

The new method only effects mutation, *after* parent selection and cross-over. Therefore it is easily incorporated into an EA with multiple-parent cross-over. Also, by mixing the parents' genes, cross-over promotes diversity.

3.8 **Potential issues**

The rate of gem creation (balanced against the cull rate) determines the number of gems available for application. The more gems there are in the jewellery-box, the more likely that an individual will match one of them, and the gem will be applied.

²⁷ Similar consistencies may apply to other graph problems (e.g. Snowplough).

²⁸ As Java interfaces, for example.

Gems promote exploitation over exploration. If gem matches are too frequent, there will be too many applications of the same mutations to many individuals, resulting in loss of diversity. Thus premature convergence to a local optimum, or even to a sub-optimum on a local fitness peak, is more likely. A probability-of-application setting can help control over-application *if* gem matches are too frequent (this depends on the problem domain and algorithm settings).

The reverse is also true: the fewer gems in the jewellery-box, the less likely that one will match, so the less likely that one is applied.

Thus we want a "Goldilocks" probability of matching: not too high (to avoid saturation), and not too low (to make it worthwhile). Phase 1 investigations provide initial insights into these issues, and they are pursued in Phase 2.

A final concern is that gems may be computationally more expensive than the gains are worth. Phase 2 includes time cost comparisons and cost-benefit analysis, and Appendix 9.1 presents order-of-complexity workings.

4. Experimental design and implementation

4.1 TSP as the proof-of-concept problem.

We selected the TSP as the proof-of-concept test problem because:

1. It is extensively researched, and widely used as an exemplar of combinatorial problems.
2. It is intuitively clear, so it should not impede understanding of gems.
3. Its mutation operators in particular are easy to understand. They are also well documented, with many to choose from.
4. It satisfies the handful of characteristics that make a problem amenable to the gem method.
5. We decided to avoid artificial "trap" problems. TSP has practical real-world value, and has extensions to more complex practical problems such as Postman and Capacitated-Vehicle.
6. A public library of TSP problem instances, called TSPLIB, is available. It is widely used and trusted in research. Furthermore, TSPLIB provides the optimum solutions, which we use in our fitness function.

4.2 Experiment design

4.2.1 The experimental scenarios

The experiments compare SEA results with those of the gem method, across many cases (scenario variations). These cases are the cross-product of a small set of dimensions, each with a small set of values, as listed below:

Dimension	Values	Relevance
1. Graph size	21, 42, 75, 150, 280.	Size and shape of the solution landscape.
2. Random/ Greedy seed individuals	All Random, Semi-greedy, or Greedy.	Location of individuals in search space, and fitness "height".

3. Population size	100, 1000 individuals	Gem saturation and loss of diversity.
4. Selection-pressure	e.g. Tournament sizes.	Loss of diversity.
5. Mutation type	Flip, Hop.	Gem impact may differ for different mutation types.
6. Maximum gem usage	30, 150 uses.	Under-use would have little value, and over-use may cause saturation, loss of diversity, and premature convergence.
7. Jewellery-box size	4, 8.	Larger size allows more gems and hence more usage (at higher computational cost).

Table 4.1. Data dimensions and ranges for analysis.

Dimensions 1 and 2 determine the solution landscape's shape, and where seed individuals are located in that landscape.

Dimensions 3, 4, and 5 effect EA process dynamics, regardless of gems.

Dimensions 6 and 7 effect gem behaviour.

Phase 1 investigations uses 360 cases ($3 \times 3 \times 2 \times 2 \times 2 \times 5$). This helped identify cases of interest for Phase 2.²⁹ Several cases were dropped (e.g. Semi-greedy and Selection-pressure), less were added (e.g. Graph-size 280), so that Phase 2 uses 120 cases ($5 \times 2 \times 2 \times 2 \times 3$).

4.2.2 Problem instances

The five problem instances (21, 42, 76, 150, and 280-node graphs) were downloaded from TSPLIB. They have different search space sizes and landscape shapes.

²⁹ Following the Sparsity-of-effects principle for factorial experiments

https://en.wikipedia.org/wiki/Sparsity-of-effects_principle

Suppose that figure 2.3 shows the search space and fitness landscape for the 42-node TSP instance. The 280-node instance's space would be vastly larger (and much lower-lying, as §5.4 shows).

4.2.3 Random vs. Greedy seed individuals

"Random vs Greedy" effects where individual solutions are likely to be located in the landscape, but does not alter the landscape. Random solutions are likely to be scattered everywhere around the search space, including low-fitness regions. Greedy solutions are likely to produce fitter individuals, concentrated around fitness peaks. They are not necessarily close to each other – they may be at similar heights on different peaks.

We hope that the method is beneficial for random seed cases in particular, because some hard problems have no adequate greedy solutions. Therefore the experiments compare gem effects on random and greedy seed populations. Half of the trials are seeded with purely *random* individuals, and the other half with *greedy* individuals.

To generate a random individual: (1) randomly select a start node (city), then (2) randomly select one node at a time until all have been selected (guarding against visiting the same node twice), then (3) return to the start node. The sequence of nodes forms the individual's chromosome, or tour. The cost is likely to be much higher than the minimum, giving the individual quite a low fitness.

There are several *greedy* ways to create a tour. We start by (1) selecting a random node, then (2) select the *nearest unvisited* node as the next node in the sequence. Continue this until the last node, then (3) return to the first node. The sequence of nodes forms the chromosome, or tour.³⁰ The cost is likely to be small, giving a highly fit individual (although probably not the optimum). Experiments gave mean greedy fitness = 72%, and mean random fitness = 27%, for the same 42-node graph.

Note that if the "greedy" algorithm selected *all* nodes greedily, there would be too little diversity among the chromosomes for the EA to work on. To avoid this, the first three nodes in each tour are selected *randomly*, and the rest selected greedily.

³⁰ Grefenstette (1987) uses the same approach to generate greedy tours.

4.2.4 Population size

Gems may impact diversity, and population size definitely does. Therefore the experiments use two population sizes. 50% of the trials use populations of 100 individuals, and the other 50% use populations of 1,000 individuals.

4.2.5 Choice of mutation operators

There are many TSP mutations to choose from.³¹ We selected *flip* and *hop* for our experiments because their algorithms differ enough to raise generic implementation questions, and flip is known to be more effective than hop (Abdoun et al., 2012; Ferreira, 2002). Their match probabilities are also different (table 3.1).

To simplify comparisons, each trial uses only one mutation type: half use only Flip, the other half use only Hop.

4.3 Process design choices

4.3.1 Individual representation

The chromosome structure is specific to the problem-domain. In TSP, each chromosome is a possible tour visiting all the nodes in the graph.

	Moving-node graphic	Fixed-node graphic	Adjacency array
Original individual I			[9, 3, 6, 7, 8, 5, 2, 10, 4, 1]

³¹ For example: Abdoun et al., 2012; Deep and Mebrahtu, 2011; Osaba et al., 2014.

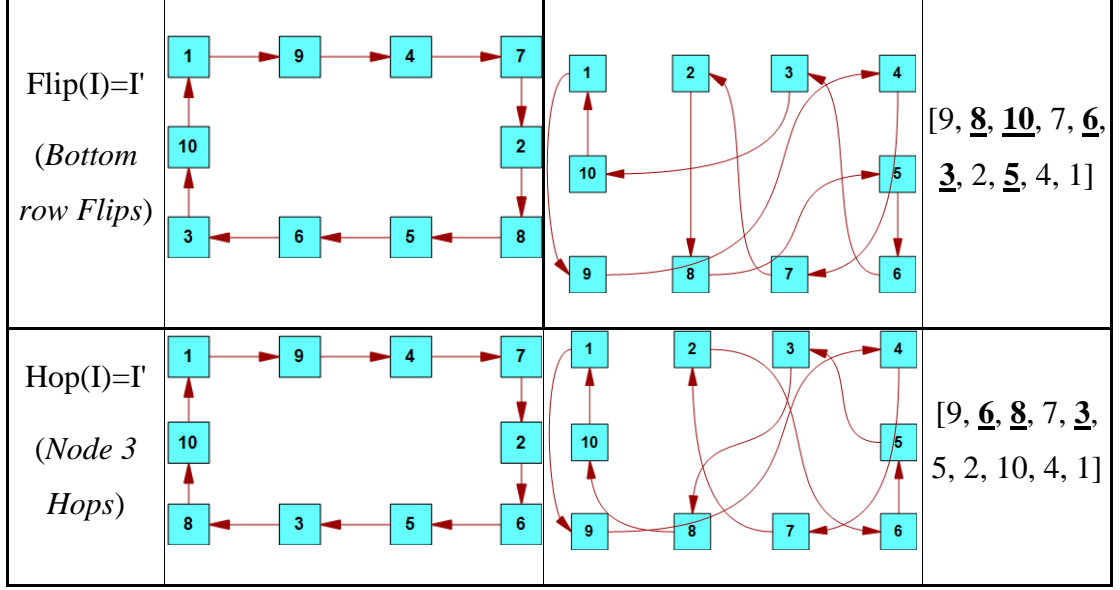


Figure 4.1. Visual and technical ways to represent a tour.

The proof-of-concept implements the Adjacency-array representation shown in the rightmost column. The array element's *index* represents the "from" node (the fixed node), and the element's *value* represents the "to" node. Other representations are available³² but we prefer the Adjacency-array because:

- It facilitates both Random and Greedy tour creation. For each *from* node, we generate a *to* node (randomly or greedily). The *to* node value is stored in the array at the *from* index, and then becomes the index for the next *from* node. All array access is direct (by index) rather than searching through the array's values. This is $O(1)$ rather than $O(n)$.
- It facilitates our mutation function. The function's parameters are the *from* nodes, which the algorithm can access directly by index: it doesn't have to search the values. This is $O(1)$ rather than $O(n)$.
- It facilitates gem representation. This is not as simple as the mutation mapping, because a gem only stores the nodes relevant to the mutation, not the full tour. Therefore we cannot use the index as the (implicit) *from* node. Nevertheless, with a little "reverse mapping" we could design the algorithms with no searches of the full array, so this too is $O(1)$ rather than $O(n)$.

³² E.g. Ordinal-array, Path-array, Binary-matrix. See Larrañaga et al. (1999), and Abdoun et al. (2012).

- d. It is minimal space: no shorter representation is possible.

4.3.2 *No invalid individuals*

Some EA implementations allow the creation of invalid individuals (for example a "tour" consisting of two separate half-tours), and penalise such individuals with a low fitness value. We do not allow this because:

- a. We find it perverse in principle,
- b. It raises tricky technical issues for gem application and matching logic, and
- c. It conflicts with a key concept: applying a gem to an individual should have a similar result to that of the original mutation instance. Invalid individuals do not make sense in this context.

4.3.3 *Parent selection and generating a child.*

Parent selection is a major exploitation factor. Since gems are also exploitative, the choice of parent selection technique is important, and effects selection pressure.

The proof-of-concept selects a *single* parent, and copies a clone to be the child (which will be mutated). Note that this was the prevalent approach before Holland (1975) proposed two-parent cross-over. Most EAs use two parents, but our focus is on mutation, not cross-over.

We used fitness-proportionate Roulette selection for the Phase 1 trials, and changed to Tournament selection in Phase 2.³³ Five individuals are selected randomly, and the fittest of these becomes the parent in this iteration.

4.3.4 *Fitness function*

Let $c(I)$ = cost of an individual (i.e. total tour distance). We define the *un-normalised fitness* of I as:

$$f(I) \equiv 1 / c(I) \in \mathbb{R}.$$

³³ Thanks to Professor Michael O'Neill's advice, and Blickle (1995).

Thus a 10 km tour has fitness = 1/10, and a 20 km tour has fitness = 1/20. The first tour is half as long, and therefore twice as fit, as the second tour. This definition works well if all tours are on the *same* graph, but we want to compare results across *different* graphs. Therefore we define *normalised fitness* as:

$$f(I) \equiv (\text{optimum cost}) / c(I) \in \mathbb{R}$$

Thus the fitness range is (0, 1]. This facilitates easy visual comparison and interpretation of chart results, even across different TSP instances.

For example, the shortest (optimum) tour on the 42-node graph is 1273 km. Let X and Y be two individuals with $c(X) = 1600$ and $c(Y) = 2000$ (km). Then:

$$f(X) = 1273 \text{ km} / 1600 \text{ km} = 0.796.$$

$$f(Y) = 1273 \text{ km} / 2000 \text{ km} = 0.637.$$

The shortest tour on the 150-node graph is 6528 km. Let A and B be two individuals with $c(A) = 8000$ and $c(B) = 16000$. Then:

$$f(A) = 6528 \text{ km} / 8000 \text{ km} = 0.816.$$

$$f(B) = 6528 \text{ km} / 16000 \text{ km} = 0.408.$$

4.3.5 Mutation function

The mutation functions use parameters to specify the nodes to be changed:

```
Individual.mutateFlip(nodeA, nodeB)
```

```
Individual.mutateHop(nodeA, nodeB)
```

This provides re-usability: since the functions are independent of how the parameter values are generated, they can be called with random or non-random parameter values. SEA mutations generate random values for nodeA and nodeB, whereas gems apply mutations with pre-recorded parameter values.

Coincidentally, both Flip and Hop have two parameters. Other mutation types may have more parameters. For example, "Slide" has three parameters: two to identify a section (as in Flip), plus one to specify the location that the section will move to.

4.3.6 Steady-state model and Replacement

The proof-of-concept uses the steady-state model rather than a generation-gap. This allows elitism, facilitates our experimental data collection model and the charts we want, and is easy to implement.

If a new child is fitter than the weakest individual in the population, then the child replaces the weakest individual.³⁴ Otherwise the new child is discarded.

4.3.7 Stopping rule

The experiments are designed to compare the SEA's progress with that of the gem method. This could be awkward if some trials stopped earlier than others. For simplicity, the algorithm stops all trials after 76,000 iterations.³⁵

Furthermore, since the proof-of-concept is only intended to investigate the method, it does not output the fittest individual.

4.3.8 Flip Gem representation and match algorithm.

A **Flip** gem is represented by an array of four integers, as shown below:

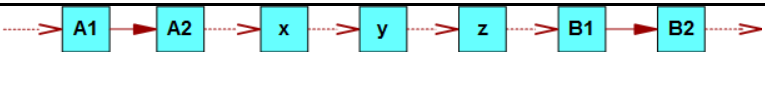
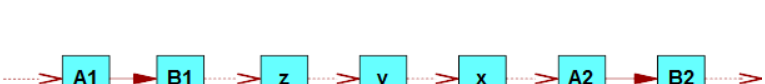

I node sequence	
m(I) = I' node sequence	
The path from A2 to B1 is reversed.	
int[4] representation	[A1, A2, B1, B2]
Omits nodes x, y, z.	

Figure 4.2. Representation of a Flip gem.

³⁴ For other replacement strategies, see Herreraa and Lozanoa, 1998, and Smith and Vavak, 1999.

³⁵ See §2.2.11 for other stopping criteria.

Let `gem[]` = the gem's `int[4]` array, and `ind[]` = the individual's array, with zero-based indices.

The Flip compatibility test is:

`gem[0]=ind[A1] And gem[1]=ind[A2] And gem[2]=ind[B1] And
gem[3]=ind[B2].`

Or vice versa (2 permutations).

The algorithm is shown below. *this* = the gem instance that is testing an individual's compatibility, and *aInd* = the individual to be tested.

```
Algorithm: boolean matchFlip(aInd)  
  
theReturn = False ;  
  
If ((aInd.nodeA1==this.nodeA1) And (aInd.nodeA2==this.nodeA2))  
{  
  
    // First arc matches  
  
    If ((aInd.nodeB1==this.nodeB1) And  
        (aInd.nodeB2==this.nodeB2)) {  
  
        theReturn = True ;           // Second arc matches  
  
    }  
  
} Else If ((aInd.nodeA2==this.nodeA1) And  
(aInd.nodeA1==this.nodeA2)) {  
  
    // REVERSED first arc matches  
  
    If ((aInd.nodeB2==this.nodeB1) And  
        (aInd.nodeB1==this.nodeB2)) {  
  
        theReturn = True ;           // REVERSED second arc matches  
  
    }  
  
}  
  
return theReturn ;
```

Algorithm 4.1. The Flip match algorithm.

4.3.9 Hop Gem representation and match algorithm

A **Hop** gem is represented by an array of five integers, as shown below:

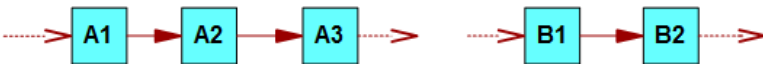
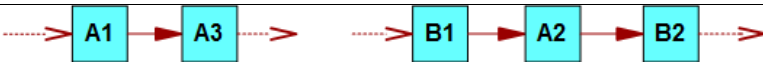
I node sequence	
Nodes between A3 and B1 are irrelevant.	
m(I) = I' node sequence	
Hop gem's int[5] representation	[A1, A2, A3, B1, B2]

Figure 4.3. Representation of a Hop gem.

The Hop compatibility test is:

gem[0]=ind[A1] And gem[1]=ind[A2] And gem[2]=ind[A3] And
gem[3]=ind[B1] And gem[4]=ind[B2].

Or vice versa, or "A" part as is but "B" part reversed, or "A" part reversed but "B" part as is (4 permutations).

The algorithm is shown below.

Algorithm: boolean matchHop(aInd)
<pre> theReturn = False ; If ((aInd.nodeA1==this.nodeA1) And (aInd.nodeA2==this.nodeA2) And (aInd.nodeA3==this.nodeA3)) { // Arcs 1 and 2 match If ((aInd.nodeB1==this.nodeB1) And (aInd.nodeB2==this.nodeB2)) { theReturn = True ; // Arc 3 matches } Else If ((aInd.nodeB2==this.nodeB1) And (aInd.nodeB1==this.nodeB2)) theReturn = True ; // REVERSED Arc 3 matches } } Else If ((aInd.nodeA3==this.nodeA1) And </pre>

```
(aInd.nodeA2==this.nodeA2) And (aInd.nodeA1==this.nodeA3)) {  
    // REVERSED Arcs 1 and 2 match  
    If ((aInd.nodeB1==this.nodeB1) And  
        (aInd.nodeB2==this.nodeB2)) {  
        theReturn = True ;           // Arc 3 matches  
    } Else If ((aInd.nodeB2==this.nodeB1) And  
        (aInd.nodeB1==this.nodeB2))  
        theReturn = True ;           // REVERSED Arc 3 matches  
    }  
}  
return theReturn ;
```

Algorithm 4.2. The Hop match algorithm.

These two match functions seem to beg for a generic design.³⁶ In Java, an interface could expose a single "match" function, with an extra parameter to specify mutation type: `match(mutationType, individual)`. The mutation-specific rules are hidden within the function, but it must branch into bespoke code *somewhere*.³⁷

4.3.10 Gem Application

If a match is successful, the gem is immediately applied to the individual. It calls the mutation function using the parameters recorded from the original mutation instance.

Using the notation introduced above, a Flip gem applies its mutation as follows:

```
mutateFlip(gem[0], gem[2]) ;
```

A Hop gem applies its mutation as follows:

```
mutateHop(gem[0], gem[3]) ;
```

³⁶ See §3.7.3.

³⁷ Probably in a "switch-case" structure.

Note: Our first implementation used a parameter to enforce a minimum iteration-gap between two applications of the same gem, and another to control the probability of gem application after a match. Each was intended to preserve diversity, but each was dropped because they were over-engineering for a proof-of-concept. Nonetheless, they may be useful in other problem instances.

4.3.11 The jewellery-box

The jewellery-box contains up to 8 gems in a fixed-length array. It also includes control variables, for example to track the minimum-value gem, and to limit the number of times that a gem may be applied.

It includes methods to: (a) insert a new gem, (b) search for a matching gem, (c) apply a gem, and (d) delete a gem.

The proof-of-concept's jewellery-box required a few more variables and functions to log gem-usage statistics, but this is "non-functional": operationally, the jewellery-box doesn't need to log statistics.

4.3.12 Gem Creation algorithm

Algorithm: boolean insertGemIfOkay(aCandidateGem)

```
theReturn = False ;
If (spareSpace > 0) {      // spareSpace = count of Nulls in
gem array.
    for (index = 0 to gemArray length - 1) {
        if (gemArray[index] == null) {
            gemArray[index] = aCandidateGem ;
            spareSpace -- ;
            theReturn = True ;
            break ; // Exit the for loop.
        }
    }
}
} Else If (aCandidateGem.value > minGemValue) {
```

```
gemArray[minGemIndex] = aCandidateGem ;  
theReturn = True ;  
refresh minGemValue and minGemIndex ; // a separate  
little function.  
}  
return theReturn ;
```

Algorithm 4.1. Gem creation.

4.3.13 Gem Deletion rules

A gem is deleted if:

- a. It has been applied the maximum number of times (30 or 150), or
- b. It has been active for the maximum number of iterations (2500), or
- c. It is replaced by a higher-value gem, as shown in Algorithm 4.1.

4.3.14 Gem Value

Gem.Value is used to test whether to replace a current gem with a candidate new gem (c. in previous section).

The proof-of-concept sets $\text{Gem.Value} = \delta F = f(I') - f(I)$.

Note that, although applying Gem X to different individuals always gives the same *cost* reduction δC , the *fitness* improvement δF varies, depending on an individual's fitness. The example below illustrates this (see Appendix 9.2 for the general proof).

- [1] Let $c(I)$ = the cost of tour I , $m(I) = I'$, $\delta C = c(I') - c(I)$, and $\delta F = f(I') - f(I)$.
- [2] Let I_1 and I_2 be two individuals, I_2 created after I_1 .
- [3] Suppose that $c(I_1) = 90$ and $c(I_2) = 70$ (cost typically reduces over time).
- [4] $m(I_1) = I_1'$ and $m(I_2) = I_2'$. Each mutation yields the same result: $\delta C = -10$.
- [5] $\Rightarrow c(I_1') = 80$, and $c(I_2') = 60$.
- [6] $\Rightarrow \delta F_1 = 1/80 - 1/90 = \mathbf{1/720}$, and $\delta F_2 = 1/60 - 1/70 = \mathbf{1/420}$.
- [7] So $\delta F_2 > \delta F_1$, although $\delta C_2 = \delta C_1$.

This is why we set $\text{Gem.Value} = \delta F$ rather than δC . δF favours later gems over earlier ones, so promoting faster turnaround to refresh gems, and reducing saturation by any one gem.

4.4 Data collection and analysis

4.4.1 Logging stages

Each EA trial loops through thousands of iterations. Observations are data snapshots taken at a series of iteration milestones, which we call *stages*. For example, Stage 0 = iteration 0, Stage 1 = iteration 200, etc.

Fitness increases steeply at first, then starts to flatten and eventually becomes almost horizontal. Gems are also more active in the early stages. Therefore we are more interested in the early stages, and want more data snapshots early on. Stage 1 finishes at iteration 200: its interval width is $(200 - 0) = 200$. After that, each stage width is $(9/8) \times$ previous stage's width, so the intervals increase geometrically, as shown below.

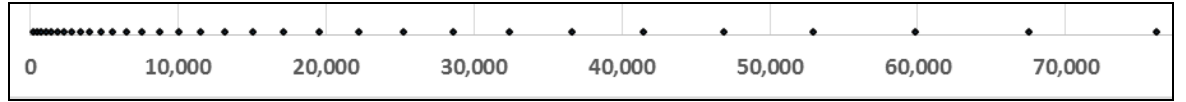


Figure 4.4. Geometric progression of the 33 log-stages' end-points.

Stage 2's width is 225 iterations, stage 10's is 577, and stage 33 has 8,667 iterations.

4.4.2 Data collection framework

A set of nested loops, one for each case dimension, runs all the experimental cases, with and without gems. It runs 100 trials for each case. Each trial logs statistics (e.g. maximum fitness) for each stage. The algorithm is shown below:

```
Algorithm: Run_all_trials()
For each jBox_usage // 0=SEA, 1=Hybrid, 2=Full Jewellery-box
  For each graph_instance // 42, 76, 150, or 280.
    For each random_greedy_level // Random or Greedy.
      For each population_size // 100 or 1000.
```

```

For each mutation_type // Flip or Hop.
    If (jBox_usage == 0) // no jBox.
        Run 100 EA trials
        // Each trial logs statistics × 33 stages.
    Else // jBox_usage = 1 or 2, Hybrid or Full
        For each gem_max_usage // 30 or 150
            Run 100 EA trials
            // Each trial logs statistics × 33 stages.

```

Algorithm 4.3. Loops to run all experiments.

Trial observations are stored in a database. After trials are finished, the data are extracted from the database for analysis and visualisation, as shown below.

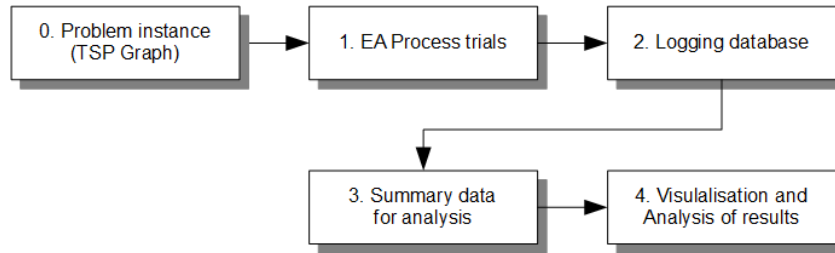


Figure 4.5. Data Flow Diagram for gathering statistical data.

4.4.3 Statistics gathered

We did not gather all statistics from the start. Phase 1 helped us decide which data Phase 2 should collect. The changes are described at the end of Phase 1.

Data	Type
JBusage + Case-id + maxGemUsage + Trial-id + Stage-id.	Compound key
Fitness Max, Min, Median, and 25% and 75% quartiles.	End of stage X
Fitness Mean and Standard-deviation.	End of stage X
Count new individuals, and "natural" ³⁸ individuals, added.	During stage X

³⁸ A natural individual is one created by a random mutation rather than by a gem.

Count of active gems.	End of stage X
Count of gems added.	During stage X
Sum of gems added gains (δF).	During stage X
Count of gem applications.	During stage X
Sum of gem application gains.	During stage X
Count of gems in population.	End of stage X
Count of deleted gems, for each of three deletion reasons, plus sum of applications, fitness gains, and lifetimes ³⁹ .	During stage X

The "End stage X" statistics are presented "as is". However, the "During stage X" statistics are normalised for fair comparison across stages of varying widths. The normalisation formula is:

$$\text{Standardised rate statistic} = (\text{Raw statistic}) \times 1000 / (\text{Stage width}).$$

For example, a count of 120 gem applications in Stage 3 (width = 285 iterations) is standardised as:

$$120 \times 1000 / 285 = 421 \text{ (applications per 1000 iterations).}$$

And 120 applications in Stage 10 (width = 649) is standardised as:

$$120 \times 1000 / 649 = 185.$$

4.4.4 Reliability of statistics.

All fitness statistics are means of lower-level sample data.

Figure 4.6 shows histograms of fitness distributions from samples of 10,000 individuals. Each histogram has ten equal-width intervals between the minimum and maximum fitness values.

³⁹ It transpired that fitness gains and lifetimes of deleted gems, were not relevant.

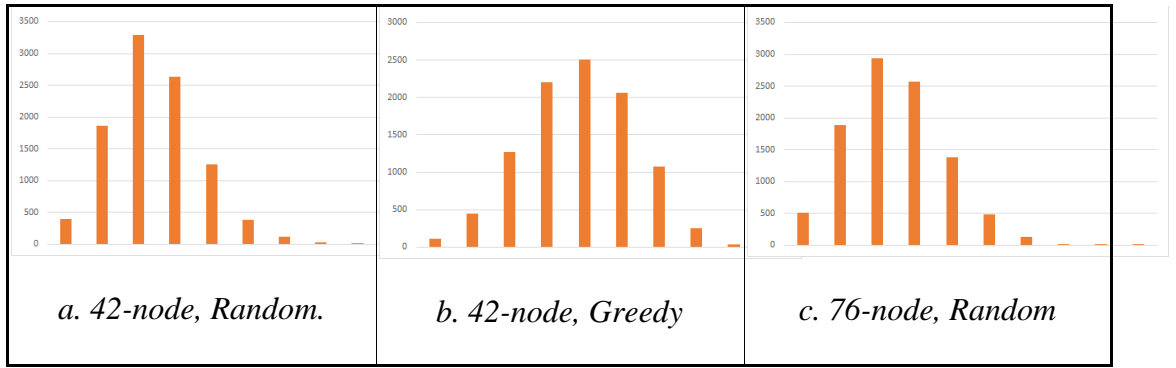


Figure 4.6. Distribution of individuals' fitnesses.

The three fitness distributions have different ranges⁴⁰, but all are bell-shaped, and look conducive for CLT⁴¹ application even with small sample sizes. With trial populations of 100 or 1000 individuals, and roll-up statistics as means of 100 trials (the usual minimum for CLT is 25 or 30), we believe the statistics are very reliable.

Furthermore, with the benefit of hindsight:

- Results show that variance in the seed population is low, and drops sharply in the early stages of the process. Such low variance lends confidence to statistical reliability.
- The smoothness of the charts gives further comfort that the "Law of large numbers" is working well.
- The Hybrid investigations effectively run the same experiments twice. The two plot-lines are so close that they cannot be distinguished visually. This lends confidence to the statistics.

4.4.5 Statistics database model

Bottom-level data are logged in a database. Afterwards, these data are rolled up to Stage Statistics (all are *means* of 100 trials). The bottom level data are no longer used – we only analyse the Stage Statistics. The Stage Statistics are like the Fact Table in a Kimball Star arrangement.

⁴⁰ a. = [0.213, 0.369], b. = [0.527, 0.888], and c. = [0.234, 0.262].

⁴¹ Central Limit Theorem.

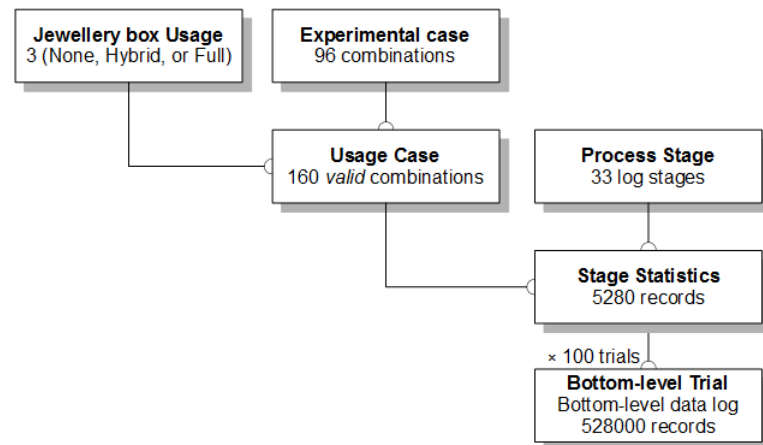


Figure 4.7. The logging database schema

5. Phase 1: Initial investigations

5.1 Introduction

Phase 1 attempts to answer initial questions, draw conclusions, prompt next questions, and review scope and decisions for Phase 2 investigations. Specifically we ask the following questions:

What is the impact of gems?

How does this impact change across the following case variations?

Random vs. Greedy-seed; Problem instance size; Mutation-type;
Jewellery-box size.

Are there indications of premature convergence?

What picture of gem activity emerges?

5.2 What is the impact of gems?

Figure 5.1 plots the increase in the population's maximum fitness as the iterations proceed, for the SEA, and for Gems. The points are data observations logged at the stages described in §4.4.1, and the lines interpolate between the points. Each line starts at the same fitness (24%) before the different processes begin.

Each SEA point is based on averages of 100 or 1000 individuals, with at least 4 cases, and 100 runs per case. Each Gem point is based on twice as many cases, for the two jewellery-box sizes.

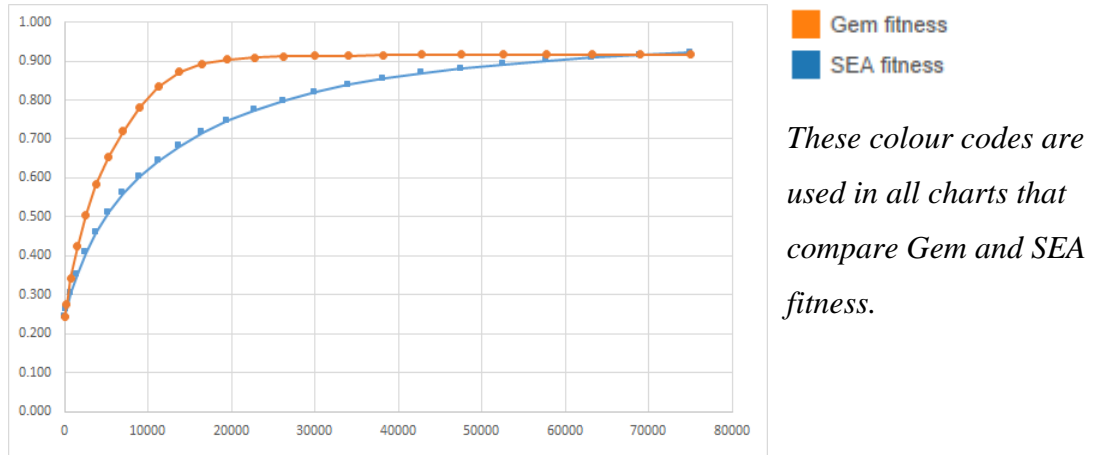


Figure 5.1. Fitness comparison. Graph-size: 42. Seed population: Random.

Initial observations:

- Both lines rise steeply in the early stages, but the slope decreases as the iterations proceed. This is expected, since the fitter an individual is, the harder it is to improve fitness by random mutation.
- Gems confer a significant advantage over the SEA in the early stages. The same is true in all other graph-sizes, to a greater or lesser extent.
- Gem fitness flattens out while the SEA's fitness continues to rise (slowly), and eventually overtakes gem fitness. We call it the "Tortoise and Hare" effect, after Aesop's fable. It is the most significant unwanted phenomenon to be investigated, and addressed if possible.

Table 5.1 supports the claim that the gap between the two methods is significant. Given the number of observations per data point (1600 for SEA, 3200 for Gems), we used z- rather than t-score to find the p-value of the *smallest* gap (4.44). It was less than 0.00001. Note that Gap/Max(σ) is much larger at other stages, and for larger graph sizes.

Data point	Fitness gap	σ^{42} : with/ without gems	Gap / Max(σ) ⁴³

⁴² The mean of σ observations, *not* the σ of observed means, which is much smaller.

⁴³ We divide by the *maximum* of each pair of σ , to err on the side of caution.

Graph-size 42, Random-seed, Stage 5	0.0680	0.01225, 0.01121	5.55
Graph-size 42, Random-seed, Stage 10	0.0409	0.00253, 0.00920	4.44
Graph-size 76, Random-seed, Stage 5	0.0503	0.00540, 0.00471	9.31
Graph-size 76, Random-seed, Stage 10	0.1529	0.00422, 0.00517	29.60
Graph-size 76, Greedy-seed, Stage 5	0.0384	0.00388, 0.00632	6.08
Graph-size 76, Greedy-seed, Stage 10	0.0264	0.00179, 0.00500	5.27

Table 5.1. Gap between SEA and Gem max fitness, in terms of σ .

5.3 Does random vs. greedy change the impact?

The previous chart was for a 42-node instance with a *random* seed population. The next chart is for the same instance, but with a *greedy* seed population.

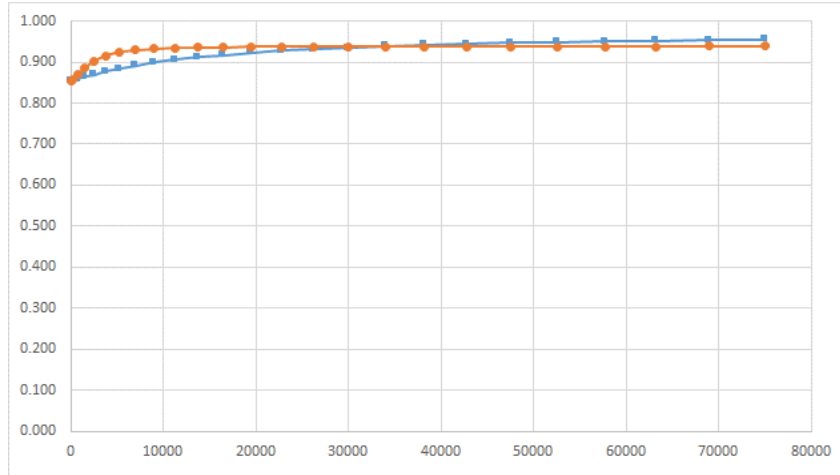


Figure 5.2. Fitness comparison. Graph-size: 42. Seed population: Greedy.

Comparing the Random and Greedy charts prompts three observations:

- Both charts show the steepest slope at the start, flattening out in later stages.

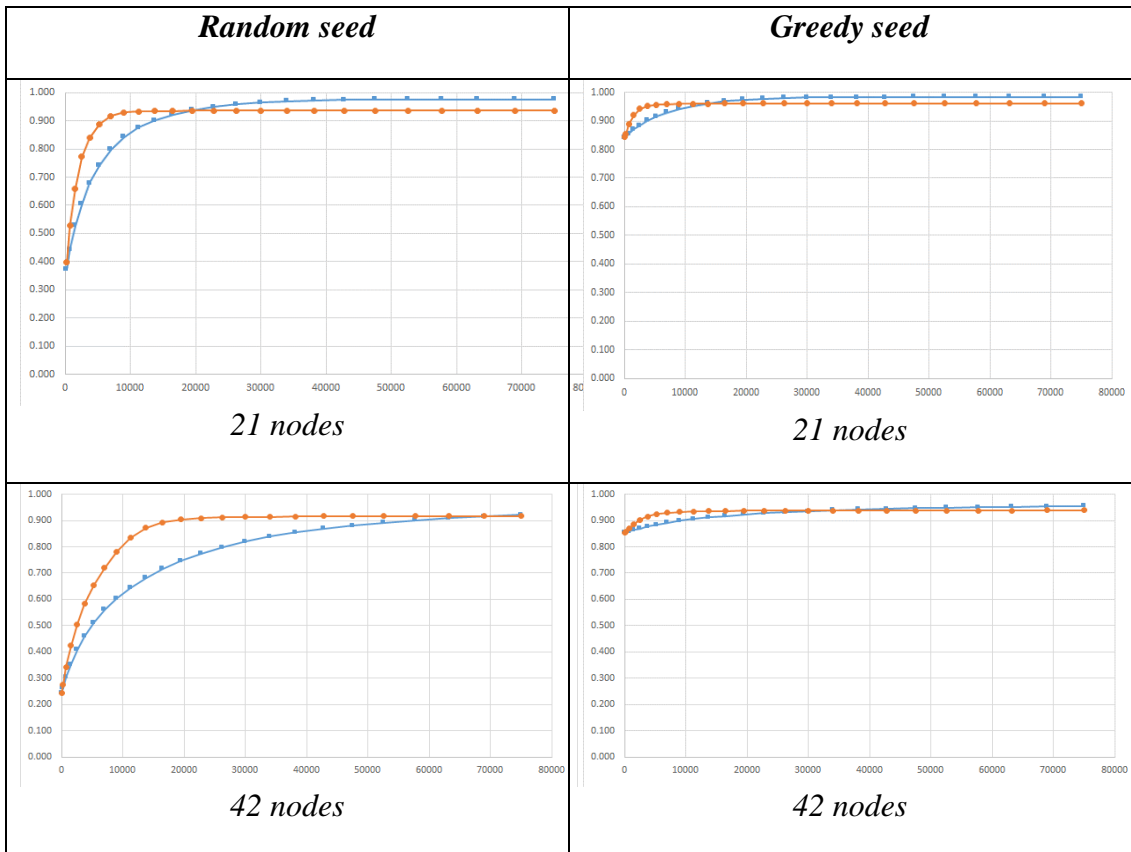
- b. The Random chart starts at a lower fitness level (24%) than the Greedy chart (85%). This is expected, since greedy tours are likely to be much fitter (shorter) than random tours.
- c. The "Tortoise and Hare" effect occurs much earlier in the Greedy chart.

Point b. partly explains why gems have a larger impact on Random-generated seed populations. Mutations are more likely to improve low-fitness than high-fitness individuals. This in turn increases the probability of new gems. Phase 2 will use gem-activity logs to investigate this further.

5.4 Does Graph-size change the impact?

5.4.1 Small, medium, and large graphs

The charts below compare fitness improvement with and without gems, for different graph sizes. For each graph size, the Random seed chart is shown on the left, and the Greedy seed chart on the right.



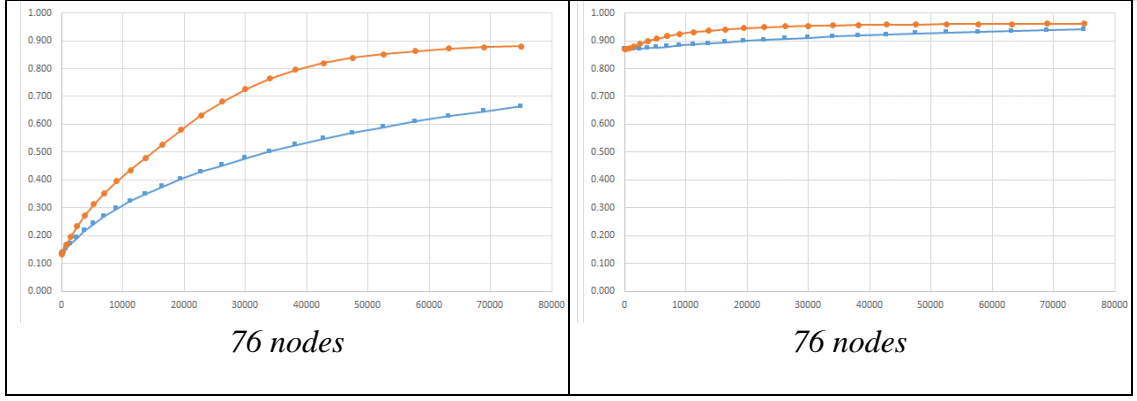


Figure 5.3. Fitness with/ without Gems, various cases.

5.4.2 Interpretation

As expected, the difference between Random and Greedy seed populations is similar across all graph sizes. The charts also show that:

- As graph-size increases, the "Random seed" fitness decreases. This suggests that, for larger graph sizes, more of the search-space is in "low-fitness" regions than for smaller graphs.
- Gems are more beneficial in larger problem instances, certainly in the early stages. We had hoped for this (because these are more difficult problems), but had not expected it because the match probabilities decrease so steeply as graph size increases (table 3.1).
- While the SEA does not overtake Gems in the 75-node problem, it is catching up. We may reasonably expect it to overtake Gems, given more iterations.

Point 1 may help explain point 2. The lower an individual's fitness, the more likely that a mutation will produce a gain, and the higher this gain is likely to be. Thus low population fitness leads to more candidate gems. This is the same *proximate* cause as for the Random/ Greedy difference. However, the *root* cause is different: lower fitness is due to large graph-size rather than random seed individuals.

Figure 5.4 shows how the different root causes have similar effects. It indicates that the largest impacts can be expected when *both* root causes are true, i.e. Random-seed population for large problem instances.

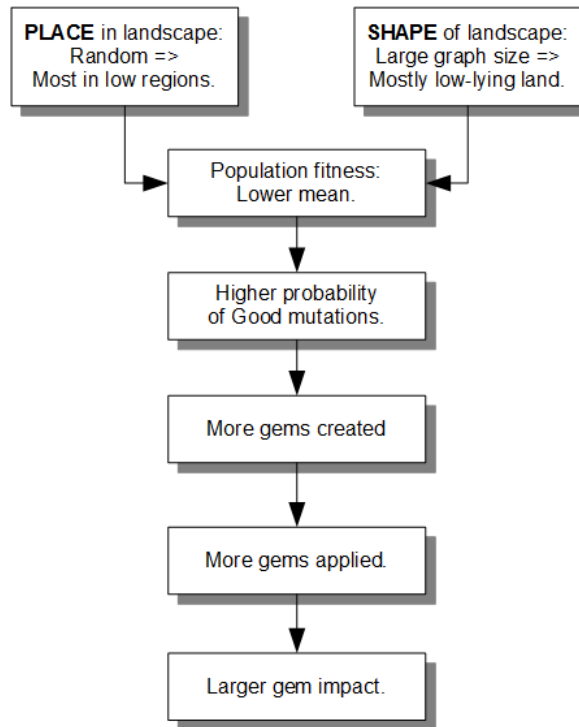


Figure 5.4. Root-cause explanation.

5.5 Does mutation-type change the impact?

We anticipated that Flip gems would have far higher impact than Hop gems, because they have far higher probability of matching individuals, and thus being applied. We were wrong.

Figure 5.5 shows fitness increasing in the SEA (blue lines), and the Gem process (orange lines), each filtered to 76-node graphs, Random-seed cases.

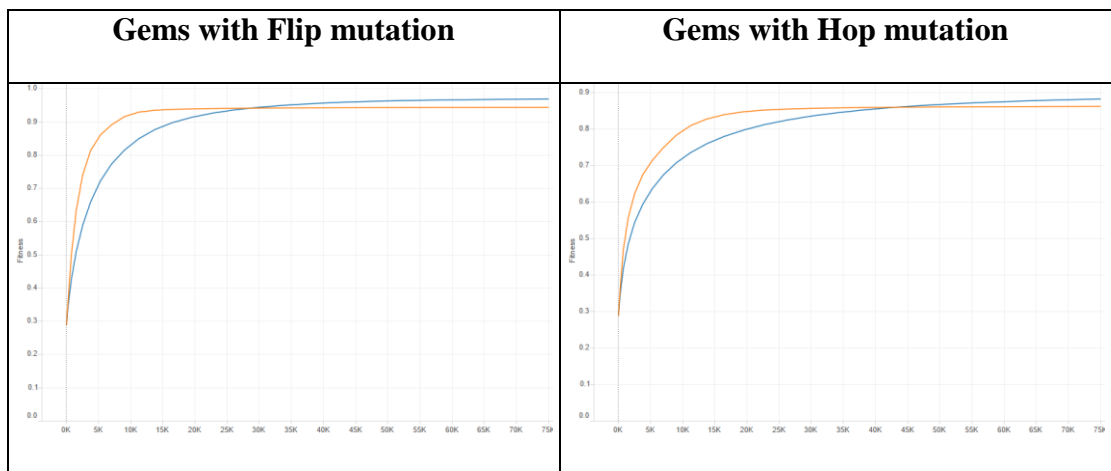


Figure 5.5. Fitness increases for Flip and Hop mutations.

The difference between the Flip and Hop gems' gains is *much* smaller than expected, and charts for the other cases also show little difference between Flip and Hop gems.

Note that SEA fitness improves faster with the Flip mutation than with the Hop mutation. Therefore we normalise the comparison by defining *boost* as:

Boost $\equiv (g/g_0) / (f/f_0)$, where g and g_0 = current and original fitness with Gems, and f and f_0 = current and original fitness with SEA.

Since $g_0 = f_0$, this simplifies to:

Boost $\equiv g/f$.

The next charts give side-by-side comparison of Flip and Hop gems' boost for two graph sizes. The line drops below 1 at the point where the SEA surpasses the Gem method.

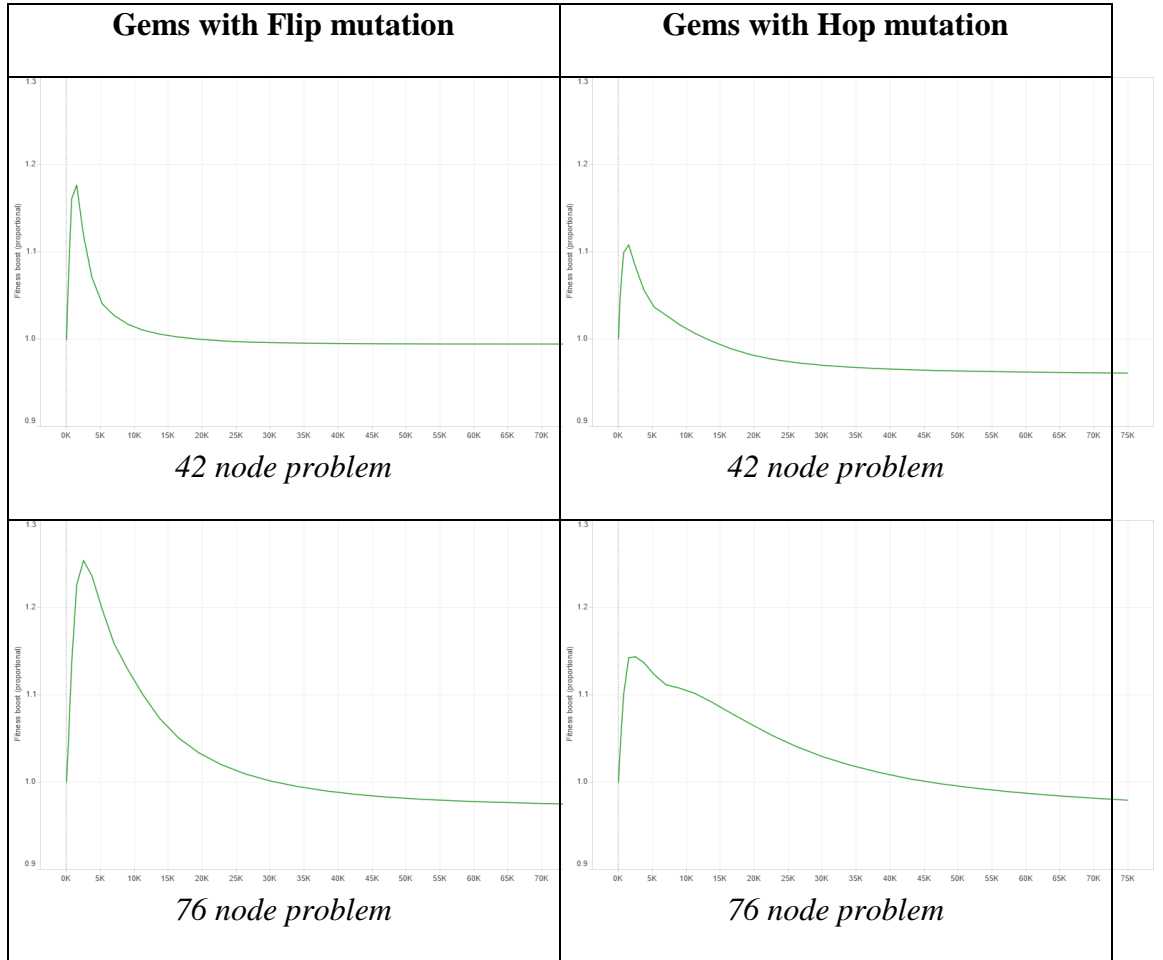


Figure 5.6. Fitness boost for selected cases.

While Flip gems provide consistently larger boosts than Hop gems, the difference is much smaller than the difference in match-probabilities: Flip is 8.5 times more likely on the 42-node graph, and 36.5 times more likely on the 76-node graph (table 3.1).

Phase 2 will probe this puzzle further.

5.6 Does Jewellery-box size impact the process?

Jewellery-box size is the size of the fixed-length array of gems (4 or 8 in Phase 1 trials), which limits the maximum number of gems that can be stored at a time.

We anticipated that the larger the jewellery-box's size, the larger its impact, but the difference would be sub-linear (the impact of size 8 would be less than double the impact of size 4). There are several reasons that the difference is sub-linear, including:

- a. The jewellery-box is only full in the early stages. After that, it usually has spare space. When the size-8 Jewellery-box has 4 or less gems, its contribution is identical to the size-4 Jewellery-box.
- b. Even when the size-8 Jewellery-box has 5, 6, or 7 gems, it is not double the number in the size-4 Jewellery-box.
- c. When the size-8 Jewellery-box is full, its best four gems (ranked by gem-value) would also be in the size-4 Jewellery-box. The additional four gems are the lowest value.
- d. Only one gem is matched and applied at a time. Thus, if one of the best 4 gems is matched, it doesn't matter if a lower-ranked gem would also match, since it wouldn't be applied anyway.

The charts below indicate such a sub-linear difference. These charts are for the 42-node and 150-node problems with Random seeds. Charts for the other cases showed similar results.

Jewellery-box size: 4	Jewellery-box size: 8
-----------------------	-----------------------

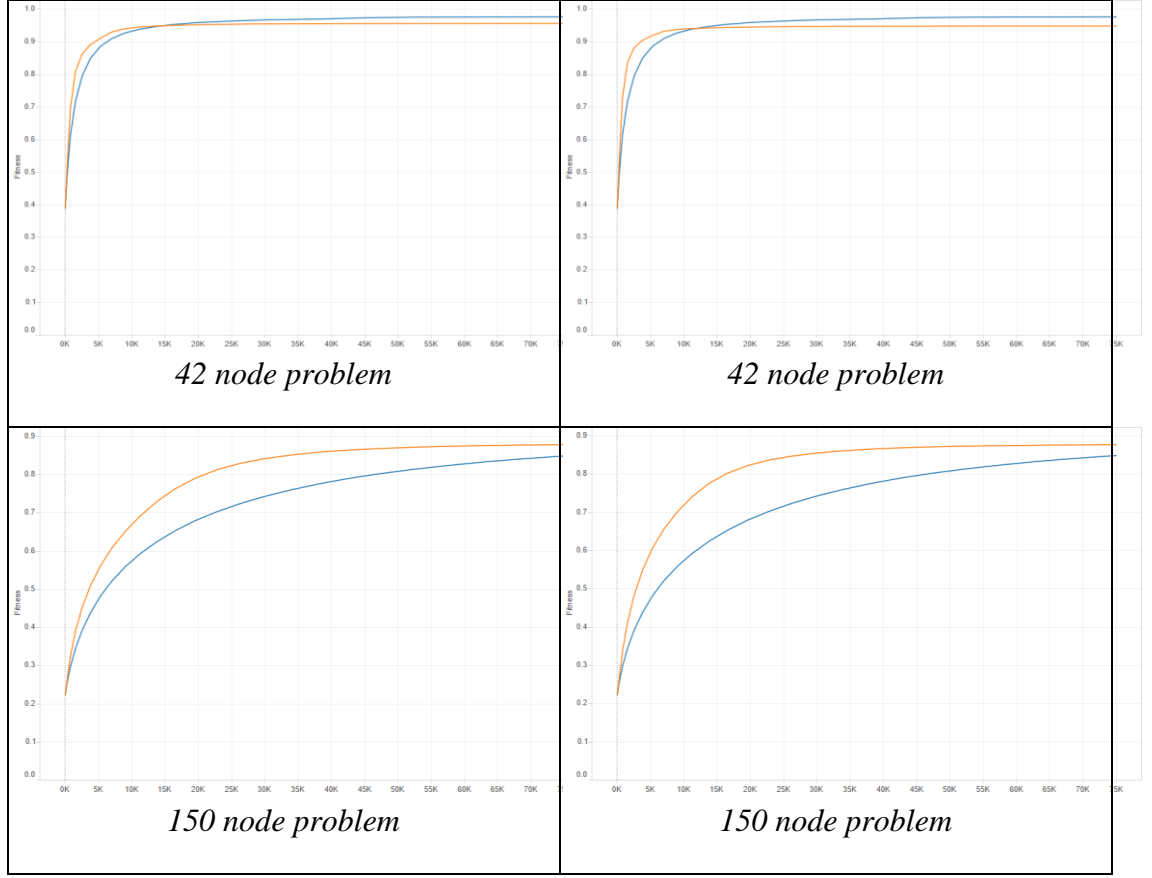


Figure 5.7. Fitness boost for Jewellery-box sizes 4 and 8.

In light of the above, Phase 2 does not use jewellery-box size as a case dimension.

5.7 Are there indications of premature convergence?

The charts show the fitness slope decreasing for both the SEA and the Gem method. However, with gems it tends to flatten out when still well below optimum. This supports concerns that gems cause over-exploitation, leading to loss of diversity and premature convergence.

One conjecture is that gems are "saturating" the population: many individuals have identical sections in tours, like the sections in figure 1.3.

High fitness variance indicates, but is not necessary for, high diversity (a highly fit population has low fitness variance, regardless of its individuals' diversity).

Figure 5.8 shows fitness variance for two cases. In each case, variance starts low, then decreases steeply before flattening out. This is consistent with, but not sufficient for, reducing diversity. We were surprised to see that fitness variance is lower with SEA than with Gems. This is also true in all other cases.

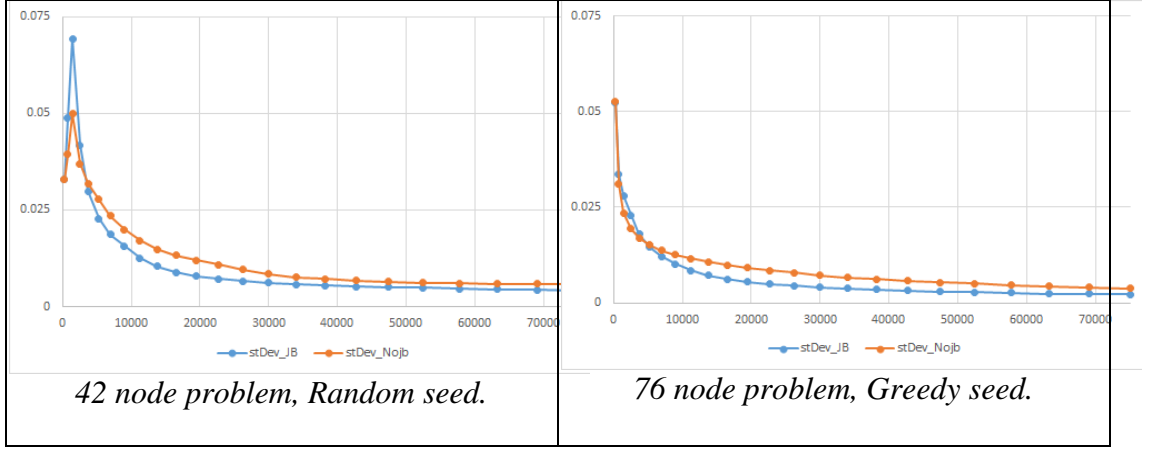


Figure 5.8. Fitness variance starts low, and reduces.

Phase 2 will revisit this.

5.8 What picture of gem activity emerges?

The next two charts show gem creation and usage in the 42-node problem, and in the 75-node problem. Both are based on Random-seed trials, because these have more gem activity and so have more data to see patterns in.

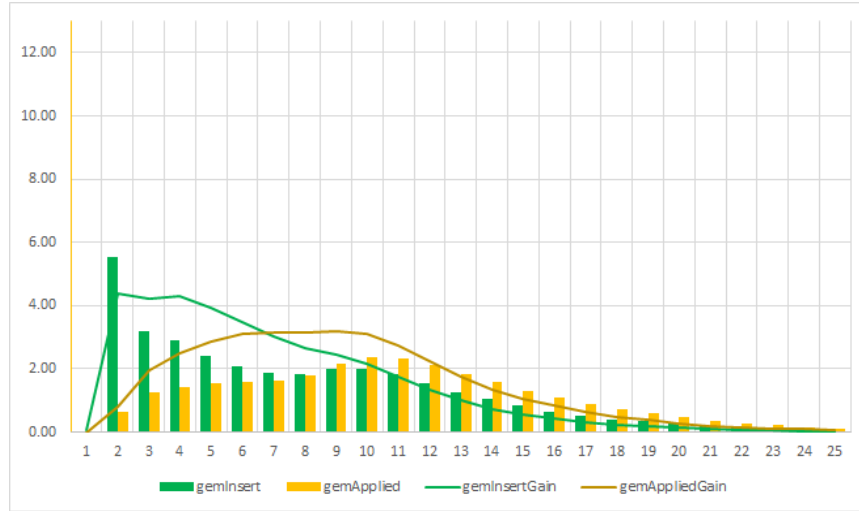


Figure 5.9. Gem activity chart for Graph-size 42, Random-seed population.

More gems are inserted in the early stages than later. This is natural, since good mutations become rarer as fitness increases, and so new gems become less likely.

The gem insert *gains* (green line) are relatively high in the first few stages, but decrease more rapidly than the corresponding *counts*. This shows that the average gem's value is reducing. This is expected: as fitness increases, the probability of mutations yielding large gains decreases, so even good mutations yield smaller gains.

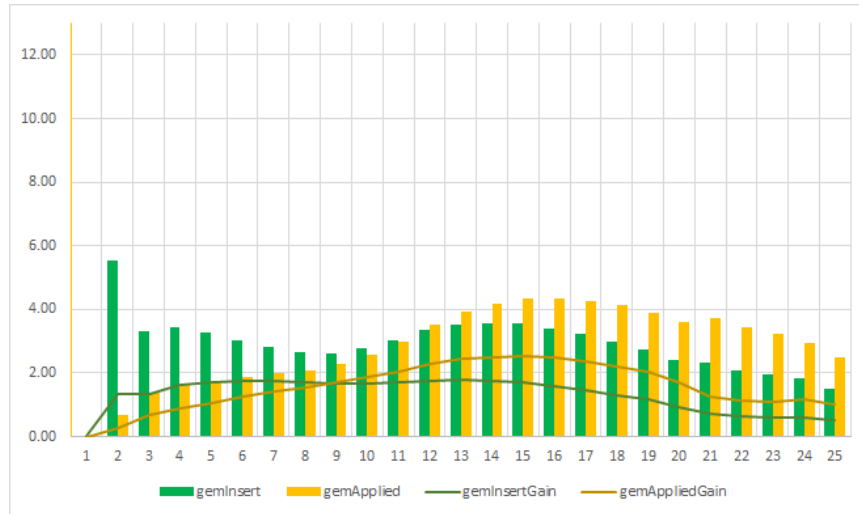


Figure 5.10. Gem activity chart for Graph-size 76, Random-seed population.

Similar observations apply to the 76 node problem. The main difference is that there is more gem activity, and it continues for longer. The explanation seems clear: since this problem starts at lower fitness than the 42 node problem, it has a higher probability of good mutations, and thus new gems. Furthermore, the fitness remains lower than the 42 node fitness, so the higher rate of activity continues for longer.

These observations accord with earlier charts showing higher gem impact on problems that start with lower fitness.

The number of gem *applications* (amber bars) increases noticeably, even while gem insertions decrease. This was not anticipated, but it is consistent with a simple "time-lag" effect: a gem inserted at iteration x , is applied *after* iteration x . However, time-lag may not be the only cause, so Phase 2 will investigate this further.

5.9 Conclusions and decisions for Phase 2 investigations

5.9.1 Change logic: Run Random-mutation **before** attempting Gem-match.

In Phase 1, the algorithm searched for a gem match *first*, and only applied a random mutation if no match is found. To see the difference, compare the flowchart in figure 5.11 below to that in figure 3.1.⁴⁴

⁴⁴ Figure 5.11 uses numbers from figure 3.1 for easier comparison.

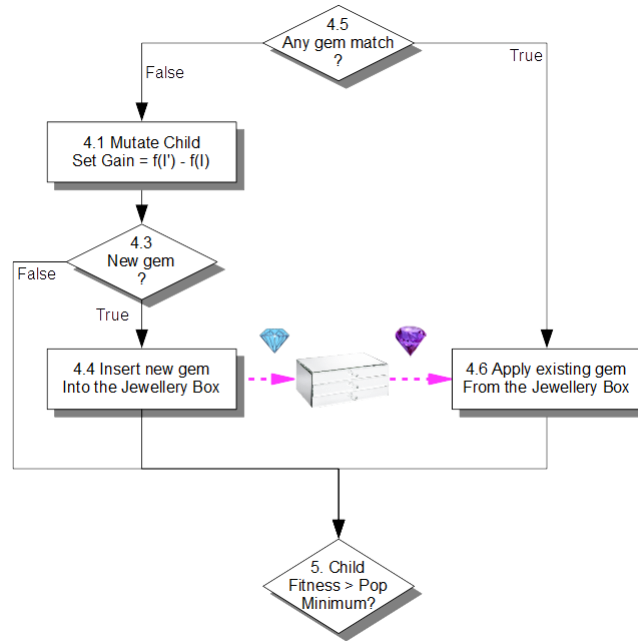


Figure 5.11. Gem functionality in Phase 1.

This increases the probability of gem application (exploitation) at the expense of random mutation (exploration). While it may not be the *main* cause of diversity loss, it is certainly *a* factor. Therefore Phase 2 will change the rules: a random mutation is applied first, and only if $\delta F < 0$ will it search for a gem match.

5.9.2 Change the parent selection technique.

Phase 1 trials used fitness-proportionate roulette selection, which tends to result in premature convergence (McGarraghy, 2014, §6.6). This exacerbates the same tendency in the Gem method. Therefore we changed to tournament selection in Phase 2, to further reduce premature convergence.

5.9.3 Question re Hop matches

The probability workings indicate that there should be far fewer Hop gem matches than Flip gem matches, yet both occur at similar rates. This is flagged for Phase 2 investigations.

5.9.4 Introduce a hybrid experiment.

Phase 2 will run trials to test the effect of switching Gems off before they saturate the population. Gems are active at the start, then deactivated at stage X so that the SEA process runs for the rest of the stages. We call this the *hybrid* method.

5.9.5 Case variations and simplifications for Phase 2

Decision for Phase 2	Reason
Use larger graph sizes (42, 76, 150, and 280 nodes). Keep the 21-node graph for comparison with Phase 1.	Larger problem instances are harder problems. Since gems have a larger impact on larger graphs, it may be easier to spot interesting behaviours.
Focus on Random-seed problems Greedy-seed trials will run and the data is logged, but often ignored in results.	Phase 2 will focus on Random cases, because we give higher practical value to solving problems that have no Greedy solution.
Drop Selection Pressure as a case variation. All trials use tournament size: 5.	Phase 1 showed no notable change due to Selection Pressure. Dropping it will simplify the trials, and the data analysis.
Drop Jewellery-box Size as a case variation. All trials use the same size: 8.	The impact of changing the size was sub-linear and easily explained. Dropping it will simplify the trials, and the data analysis.
Drop Minimum Gem Usage Gap.	This feature was intended to reduce saturation. It interacts with other features, making the results difficult to interpret. In hindsight, it is over-engineering, with too many moving parts for a proof-of-concept. Dropping it will simplify the trials, and the data observations.

Simplify the Gem Value attribute. Set value = δF .	Gem value is used to decide if an existing gem will be replaced with a new gem. Phase 1 used a function of ($\delta F/f(I)$ and the iteration), which was over engineered.
---	---

5.9.6 Log new data for statistics

Data-logging changes for Phase 2	Reason
Record the count of new individuals created.	This may give insight into the exploration/exploitation balance, and saturation. (New Individuals) – ("gem" individuals) = ("natural" individuals).
Drop the top-20 th percentile observations. Log the 25%, median, and 75% quartiles.	Quartiles are widely used, and easy to interpret as a measure of spread.
Use the geometric log-stage model (§4.4.1).	Phase 1's stage widths increased <i>arithmetically</i> , to focus data logging in the early stages. But the Phase 1 charts enticed us to zoom further into the early stages.

6. Phase 2: Further investigations

6.1 Introduction

6.1.1 Phase 2 overview

The main purpose of the proof-of-concept is to test the gem method, and see how large a fitness boost gems yield (if any). Phase 1 showed that gems provide a large fitness boost, but raised concerns about premature convergence.

Therefore Phase 2 begins with a review and confirmation that the method still provides fitness boosts, after amending the algorithm to promote exploration at the expense of gems.

Section 6.3 presents empirical insights into gem activity, to inform later investigations and conjectures.

A major concern has been that gems promote exploitation over exploration, with consequent loss of diversity. Section 6.4 analyses variance, diversity, and gem saturation, to inform later investigations.

Section 6.5 investigates one of the main questions from Phase 1: why are Flip and Hop gem applied at a similar rate, given their very different match probabilities?

Section 6.6 investigates a hybrid process as a potential solution to premature convergence.

Section 6.7 proposes the clan theory as an explanation of many gem behaviours, including some that surprised us.

Finally, section 6.8 presents a comparison of SEA and Gem runtimes. This enables a cost-benefit analysis: gems certainly boost the SEA fitness, but at what cost?

6.1.2 Notes on results presented

Phase 2 focuses on Random-seed cases, because (a) these are of most interest given that many problems do not have satisfactory greedy solutions, and (b) Phase 1 showed most gem activity in these cases, so trends may be easier to spot.

Most research presents either the maximum or mean fitness. We use the upper 75th fitness percentile as a compromise.⁴⁵ Charts of the three measures tell much the same story.

6.2 Phase 2's revised Gem rules.

6.2.1 Phase 2's gem rules allow more random mutations than Phase 1's rules.

In Phase 1, if an individual was compatible with a gem, the gem was always applied, pre-empting random mutation. This favoured exploitation over exploration.

In Phase 2, if a random mutation improves fitness, it is applied, pre-empting gem application. This allows more random mutations, reduces gem applications, and so helps to restore the exploration/ exploitation balance. This should reduce premature convergence, at the cost of a smaller gem boost in the early stages.

Figure 6.1 shows Phase 2's additional random mutations (the small square at the bottom left, labelled $\delta F > 0$). We hope it is enough to offset premature convergence.

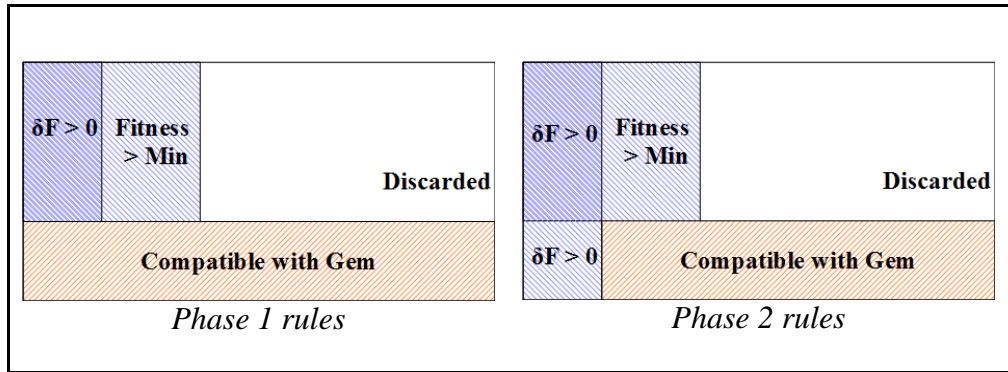


Figure 6.1. Comparison of Phase 1 and Phase 2 gem application rules.

Let $P(\text{compatible with gem}) = C$, and $P(\delta F > 0) = V$ (C and V are independent).

Let Phase 1 probability of creating a new *natural* individual = X .

Then the probability of creating a new *natural* individual in Phase 2 = $X + CV$.

⁴⁵ The 75th percentile's distribution is also more bell-shaped than the Maximum's distribution.

6.2.2 Random vs. Greedy cases

Figure 6.2 compares Random and Greedy-seed results, for the 42-node and 150-node graphs. Similar Random vs Greedy-seed patterns are apparent for all the other graph sizes and cases.

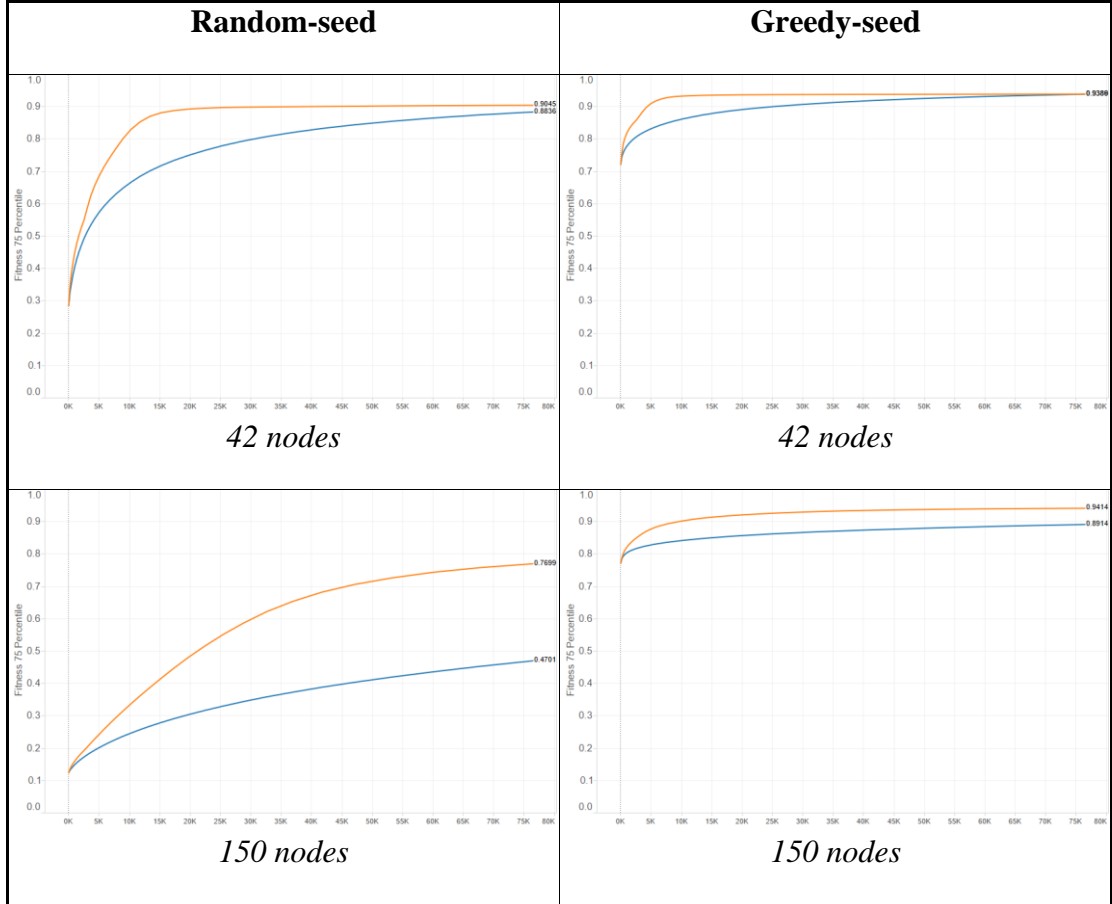


Figure 6.2. Random vs Greedy-seed cases.

These fitness boosts are similar to those of Phase 1. Furthermore, the change of logic to reduce gem application seems to have worked: while premature convergence is still visible in the 42-node problem, it is less severe than in Phase 1 (see figure 5.3).

We see no reason to change our root-cause analysis of gems' larger impact on Random-seed cases.⁴⁶

⁴⁶ See figure 5.4.

6.2.3 Graph-size cases

The charts in figure 6.3 compare SEA and Gems for various graph sizes. They accord with Phase 1's results. Charts (a) and (b) show the Gem line levelling out. While less severe than in Phase 1, premature convergence is still an issue. One may expect a similar levelling out in charts (c) and (d), given sufficient iterations.

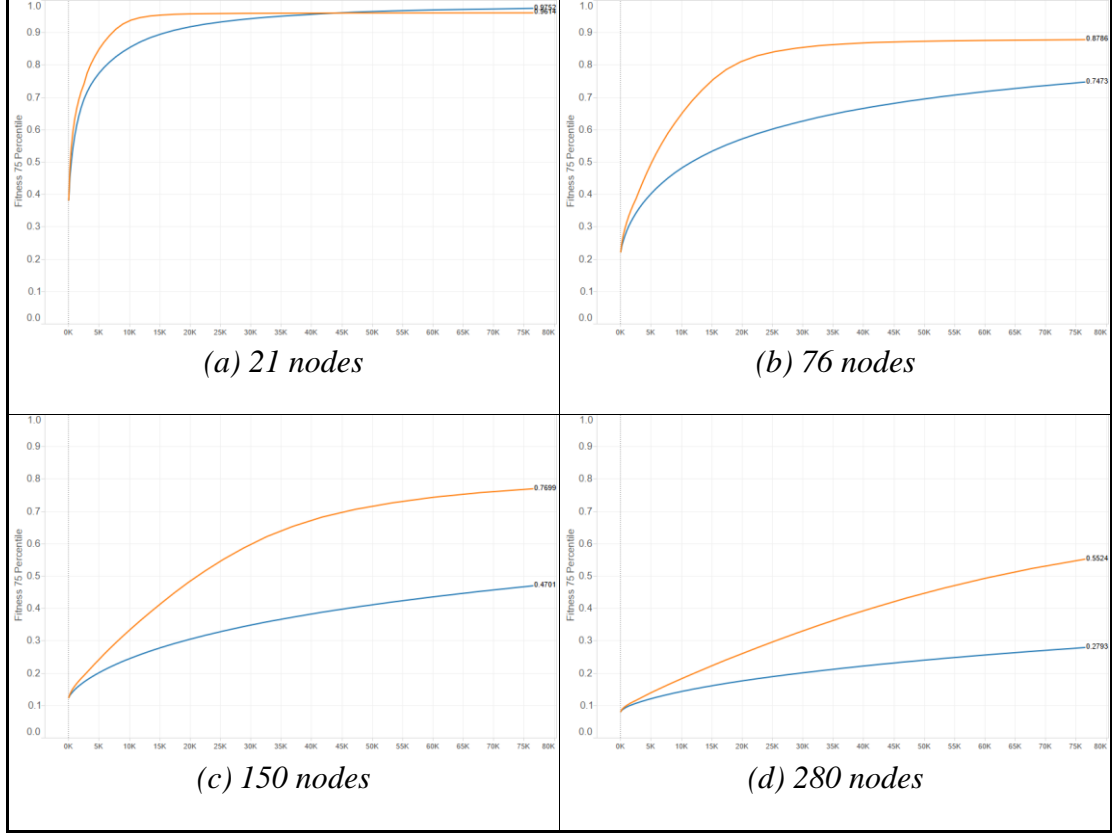


Figure 6.3. Four graph-size cases (all random-seed).

Again, we see no reason to change the root-cause analysis of gems' larger impact on larger problem instances.

SEA fitness differs for each graph size. Therefore, as in Phase 1, we use normalised fitness *boost* to compare the cases.⁴⁷ For example, the maximum fitness boost for the 150-graph problem is 1.911, meaning that at this point (end of stage 29), the fitness achieved with gems is almost double that achieved by the SEA.

⁴⁷ See §5.5.

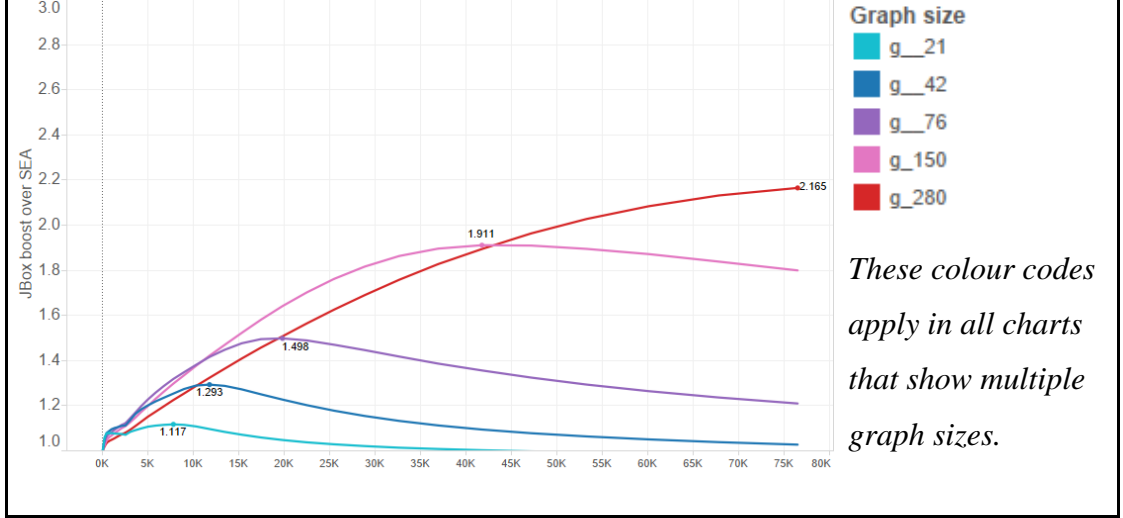


Figure 6.4. Fitness boost during process.⁴⁸

Figure 6.4 also suggests that process dynamics are similar for different problem instances, but both horizontal and vertical scales are larger for larger problems – the process "life-cycle" is slower. Thus we may expect the downturns visible in the smaller problems, to occur in larger problems also, if the process ran for more iterations.

6.2.4 Population-size cases

Population-size is of interest because smaller populations lose diversity faster than large populations, and loss of diversity leads to premature convergence. The charts in figure 6.5 are filtered to Flip mutations and max-gem-use = 150, to amplify the gem effect.

Population-size = 100	Population-size = 1000
-----------------------	------------------------

⁴⁸ The 21-node graph is shown for comparison with Phase 1. The rest of Phase 2 will focus on the larger (harder) problem instances, ignoring the 21-node graph.

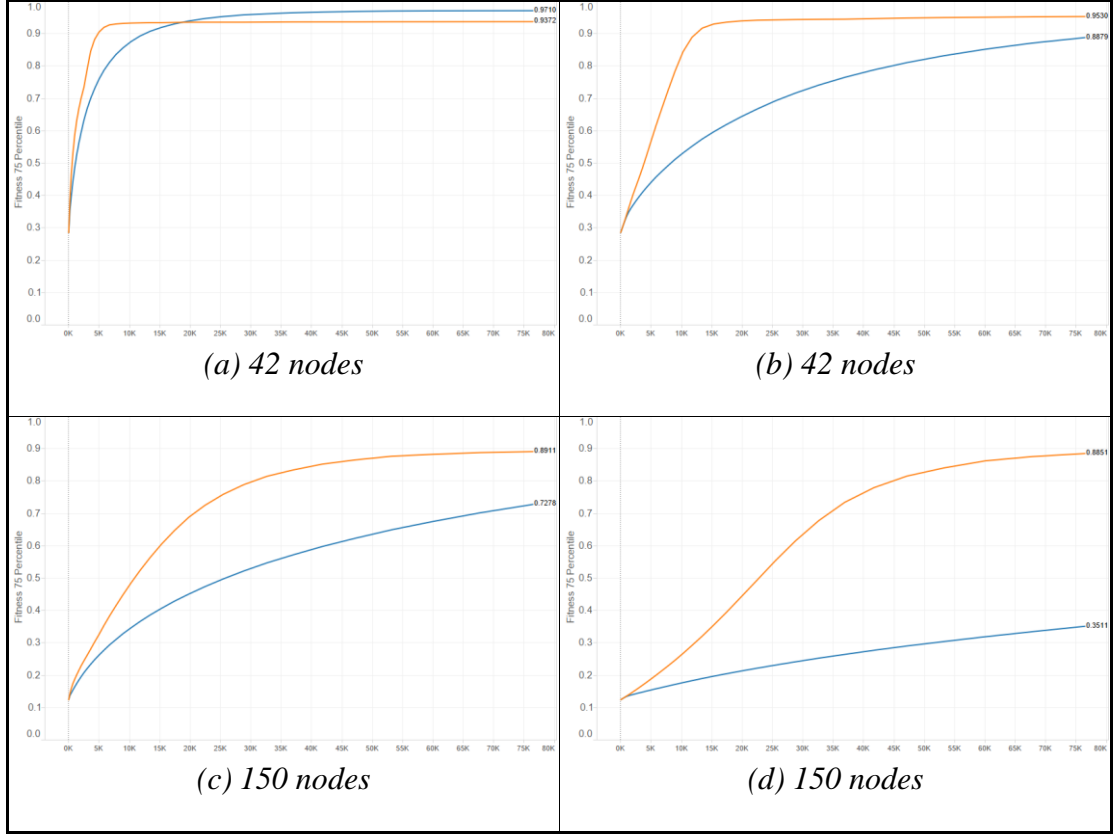


Figure 6.5. Small vs Large population size.

We make the following observations:

- Premature convergence is more apparent in small populations, especially for the 42-node graph. This is due to faster loss of diversity in small populations, in agreement with the research (Koljonen and Alander, 2006).
- The steeper fitness improvements in small populations, agrees with the research (see Gotshall and Rylander, 1992; Koljonen and Alander, 2006; Roeva et al., 2013).

The SEA's fitness increase differs with population size, so again fitness-boost is used to normalise across population sizes.

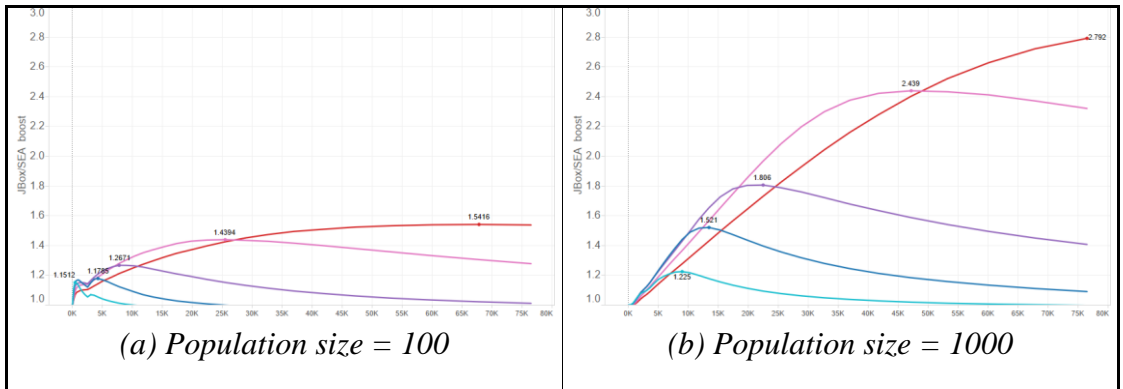
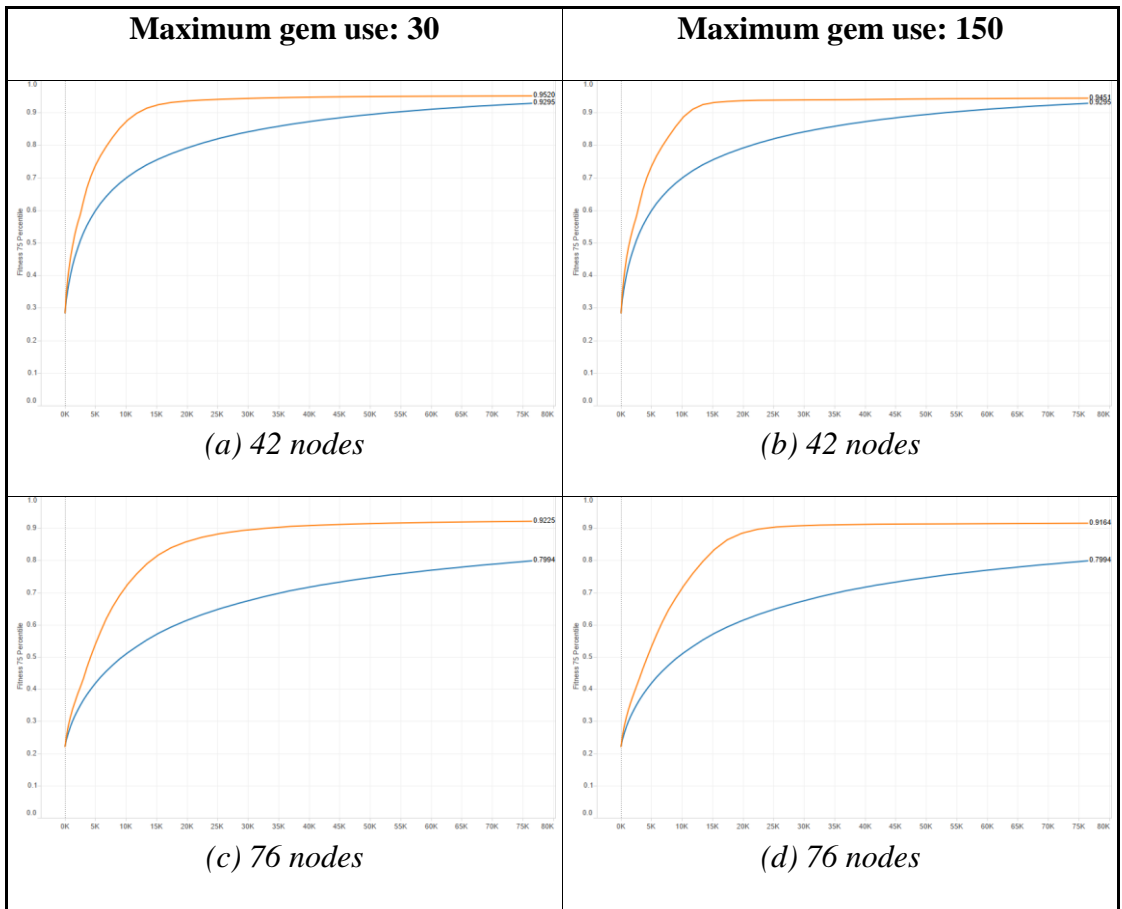


Figure 6.6. Boost for small and large population sizes.

Gem boost is far smaller when population size = 100. However, one may still decide to set a small population size. Diversity is only a means to an end: the EA's purpose is to find the fittest solution, and research indicates that fitness improves faster in smaller populations, at the risk of premature convergence. It is a judgement call.

6.2.5 Maximum gem usage cases

Maximum usage limits the number of times any single gem may be applied. We expected that the larger limit (150) would have a larger effect, both positive (faster fitness improvement) and negative (faster loss of diversity).



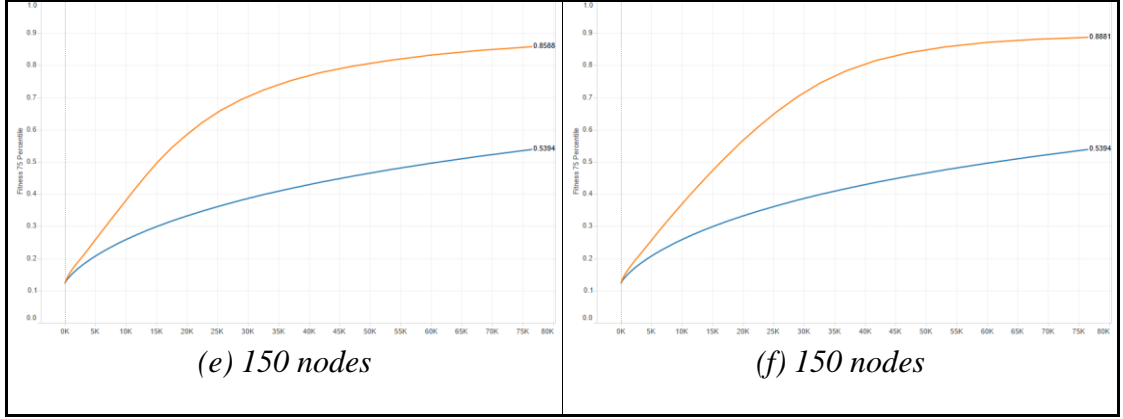


Figure 6.7. Comparison of Gem Usage limits.

Note that the gem fitness line has a slightly gentler "elbow" for max-gem-use = 30, than for 150. This is most obvious in the 42-node case. The tentative conclusion is that it results from the anticipated effects (positive and negative). This supports the conjecture that over-use of gems leads to premature convergence.

The difference between the two limits (30 and 150) is less than anticipated. This may be due to a balancing feedback: a lower limit causes more gem deletion, which gives more space to create new gems. Hence the jewellery-box stores similar numbers of gems in each case.

6.3 Gem dynamics and activity

6.3.1 Gem creation and deletion

Figure 6.8 shows gem creation on the left, and gem deletion on the right, for the five problem instances.

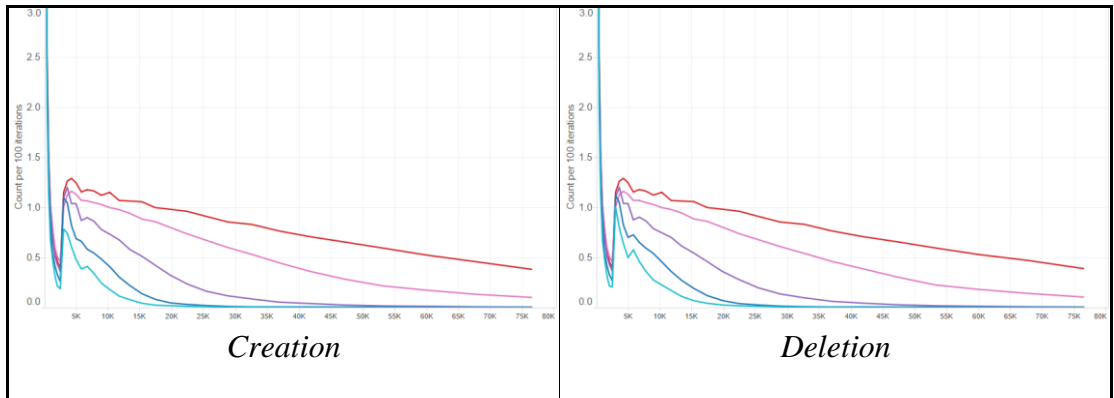


Figure 6.8. Gem creation and deletion rates, per graph size.

Observations:

- a. The creation and deletion rates are very similar. This is natural, as there is a "balancing" feedback: (1) the more gems that are created, the more that can be deleted, and (2) the more gems that are deleted, the more room there is to create new gems.
- b. Gem creation (and deletion) rates are highest for the largest problem instance (the 280-node graph, shown on the red line). Again, this is natural. The root-cause diagram (figure 5.4) shows that larger graph sizes start and continue with lower fitness, and hence more gems are created.
- c. All plot-lines have a spike in the early stages. This is because gems are deleted 2,500 iterations after creation, which also allowing more gem creation at about the same stage.⁴⁹

The balance of gem creation and deletion determines the number of active gems (maximum = 8). Figure 6.9 shows the number of active gems for the various graph sizes.

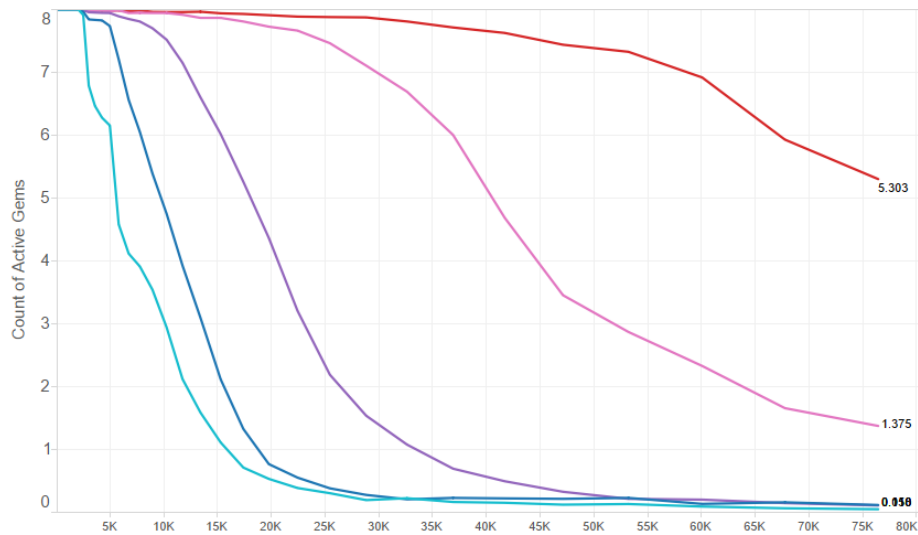


Figure 6.9. Number of gems active per graph size.

The larger the problem instance, the more gems are active. This is easily explained: larger problem instances start with lower fitness, and remain at lower fitness for longer. Therefore good mutations, and new gems, are more likely for larger problem instances, throughout the process.

⁴⁹ Logging stages 8 and 9 end at iterations 2511 and 3025 respectively.

6.3.2 Gem application

Figure 6.10 shows the number of gem applications per 100 iterations, for each graph size. As expected, there are more applications in the larger problem instances, since there are more gems in such problems (figure 6.9). This also accords with the results in §6.2.2 (larger boosts for larger problem instances). A coherent overall picture of gem dynamics is emerging.

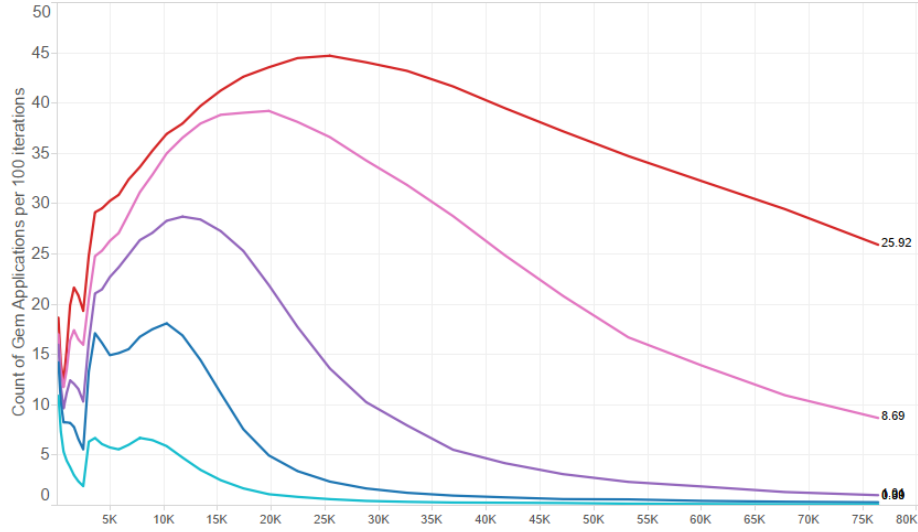


Figure 6.10. Number of gem applications per 100 iterations per graph size.

6.4 Gem impact on diversity.

6.4.1 Fitness IQR and standard deviation

High variance indicates high diversity. Figure 6.11 compares IQR⁵⁰ and standard deviation for SEA and with Gems. The upper pair of lines show fitness IQR for SEA (blue) and Gems (orange). The lower pair of lines show standard deviation for SEA and Gems.

⁵⁰ Inter Quartile Range: the range that includes the distribution's middle two quartiles, from 25% to 75%.

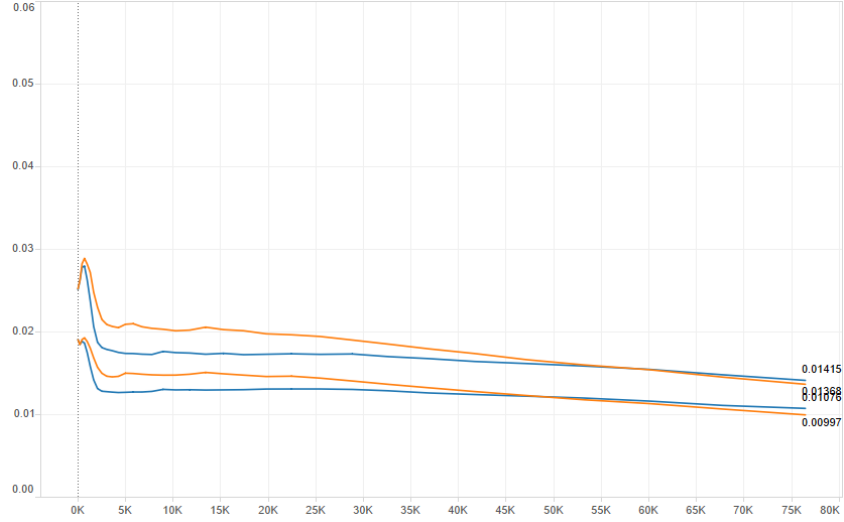
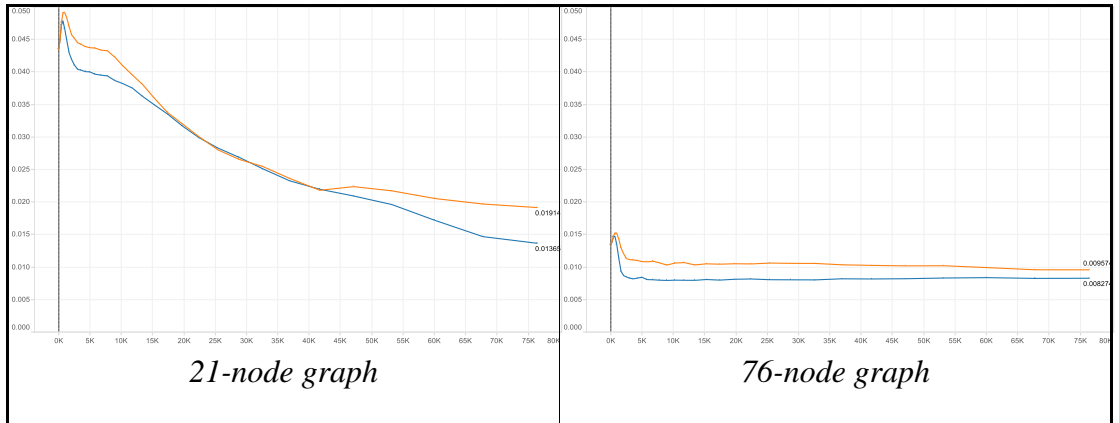


Figure 6.11. IQR and σ for 42-node graph, Random-seed, Population-size: 1000.

Observations:

- Both IQR and σ start low, reduce rapidly, and then flatten out. This occurs in all other cases. For example, IQR and σ start much larger with Greedy seeds, but the trend is similar: a rapid reduction followed by levelling out.
- In all cases, both IQR and standard deviation are higher for larger populations, and the difference between Gems and SEA is larger when maximum gem usage is 150 rather than 30 (not surprisingly). Therefore the IQR and σ charts are filtered to these cases, to make visual comparison easier.

Figure 6.12 compares the SEA and Gem IQRs for various graph sizes. Standard deviation follows a similar trend, but is not shown to reduce clutter.



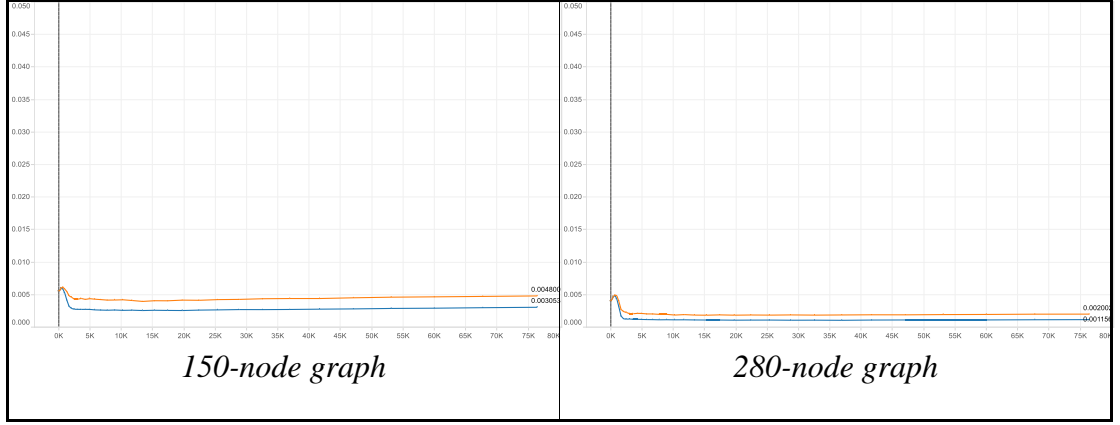


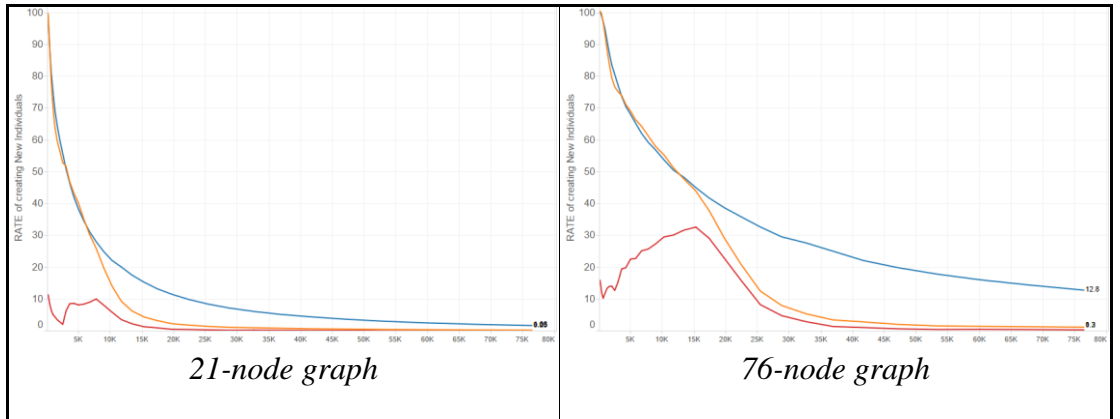
Figure 6.12. IQR for various graph sizes, Random-seed, Population-size: 1000.

Given the concern that gems cause diversity loss, we did not expect the IQR to be larger for Gems than for the SEA, nor did we expect the larger graph sizes to have smaller IQR.

Note that the mean IQR/σ over all cases = 1.353, and the IQR/σ figures per case, and per stage, are very close to this. This is close to IQR/σ for the Normal distribution (1.349). It is consistent with bell-shaped distributions, as expected by the CLT.

6.4.2 Rates of new individual creation

Figure 6.13 compares the rate of new individual creation in the SEA and Gem processes (blue and orange lines). As before, we focus on cases where the comparisons are most striking: Random cases with Flip mutations and high maximum gem usage. Similar trends are observed in all other cases.



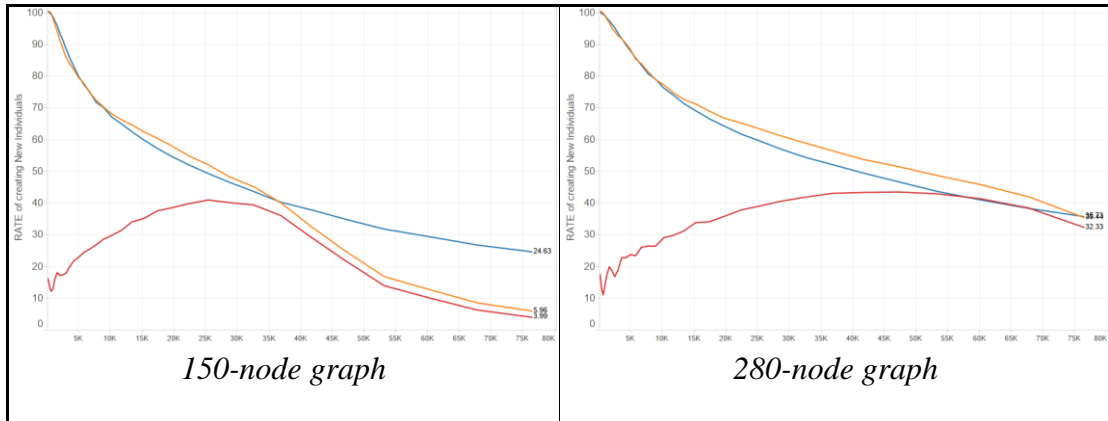


Figure 6.13. Rates of creating individuals, Random-seed, Population-size: 1000.

Observations:

- In all cases, both SEA's and Gem's new individual creation rate start high and drop rapidly. This is expected: the probability of a mutation yielding a fitness gain decreases as fitness increases, and the probability that a new individual will replace an existing one, also decreases.
- In all cases, while the gem method may create more new individuals in early stages, it generates less in later stages. This is probably mostly due to the higher fitness of the gem population in later stages.
- From about the stage that gem application (lower red line) starts to decline, Gem new individuals (orange) are closely aligned to gem applications. The gap between the orange and red lines represents the number of individuals created "naturally", and it is far less than the number created by gems.

6.4.3 Gem saturation

Figure 6.14 shows the number of gems per individual (red line). This includes gems inherited from ancestors, so it is cumulative. The blue line shows the number of gems per 100 genes, to normalise across different chromosome sizes.

Population-size = 100	Population-size = 1000
-----------------------	------------------------

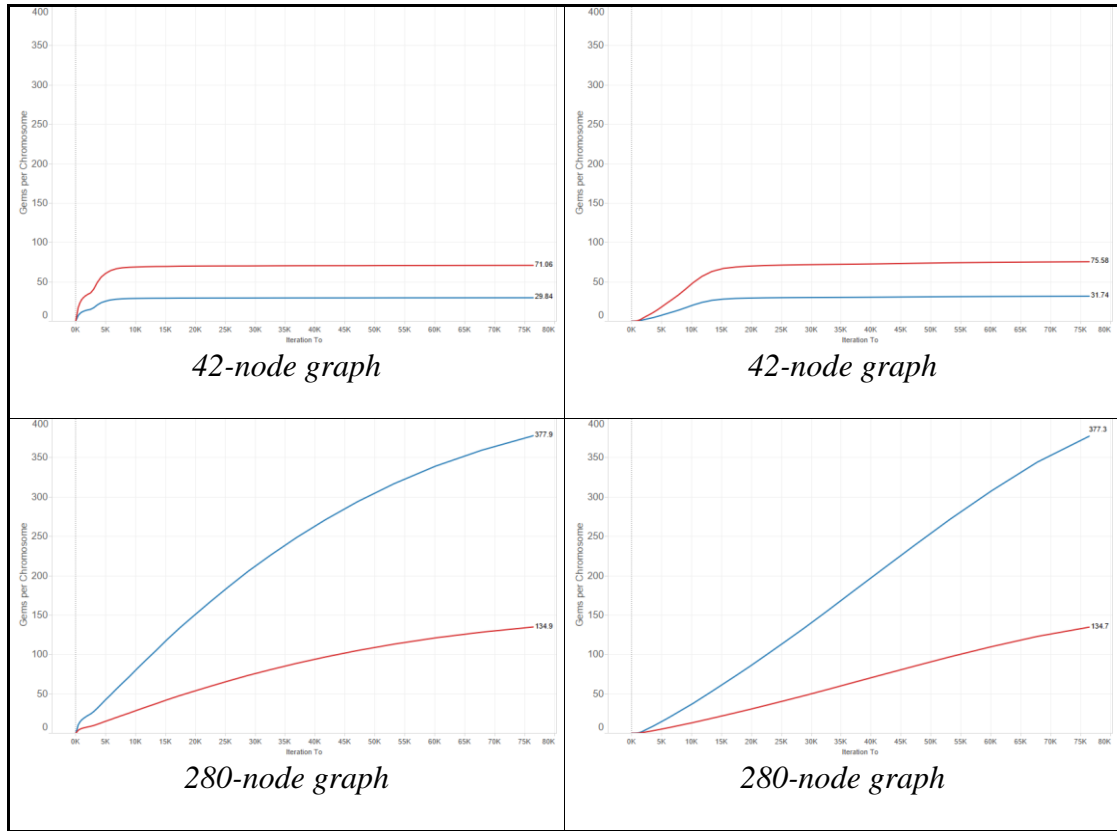


Figure 6.14. Saturation: gems per chromosome, and per 100 genes.

Note that the saturation per 100 genes is far higher for large chromosomes. This is due to two factors:

- There are more gem applications for larger graph sizes, and this accumulates in descendant individuals.
- The clan theory (§6.7) predicts higher gem-match frequency for larger chromosomes.

In the context of saturation, the number of *pure* individuals is also interesting. A pure individual is one that has no gems in its ancestry. Figure 6.15 shows the rapid loss of pure individuals. Gem application is so frequent that no pure individual survives the first 7,500 iterations (the horizontal axis is truncated to 7500 iterations), even in larger populations where such survivors are more common.

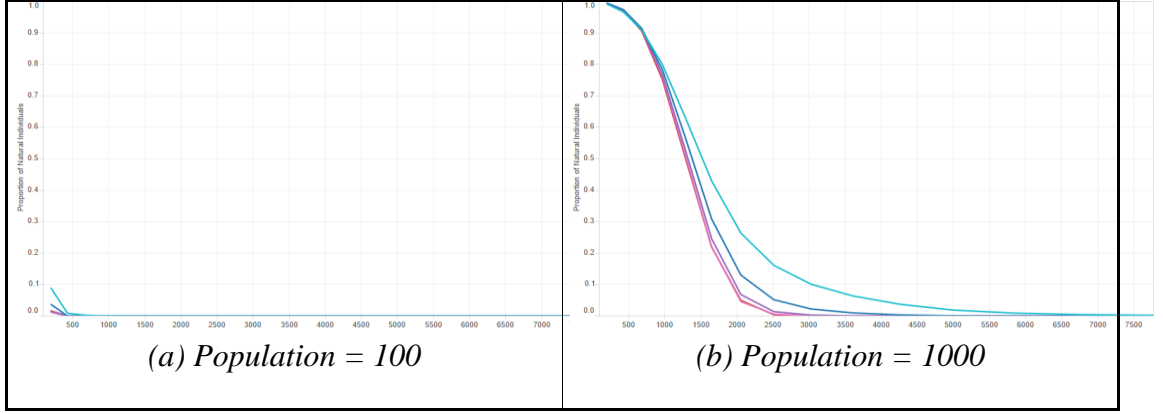


Figure 6.15. Proportion of pure individuals in small and large populations.

In summary, gems propagate so rapidly that they soon saturate the population.

6.5 Comparison of Flip and Hop impact and gem activity.

6.5.1 Comparison of impact on fitness

Phase 1 left an open question: Given the large difference in Flip and Hop gems' match probabilities, why are their impacts so similar? Figure 6.16 shows that the puzzle persists.

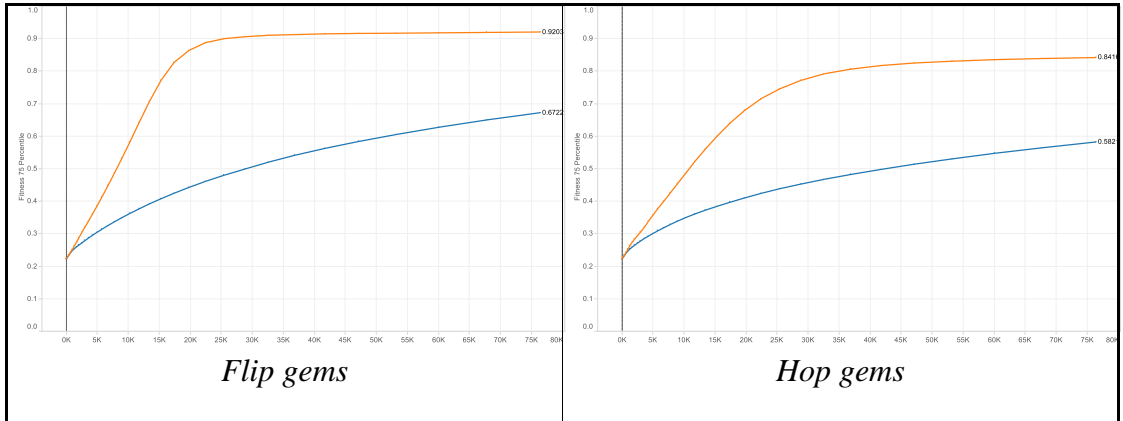


Figure 6.16. Impact of Flip and Hop gems.

Note that the SEA also differs for Flip or Hop cases. Therefore figure 6.17 compares the normalised fitness boost. As expected, in both cases the boost increases with graph size, from the lowest (blue for the 21-node graph) to the highest (red for the 280-node graph).

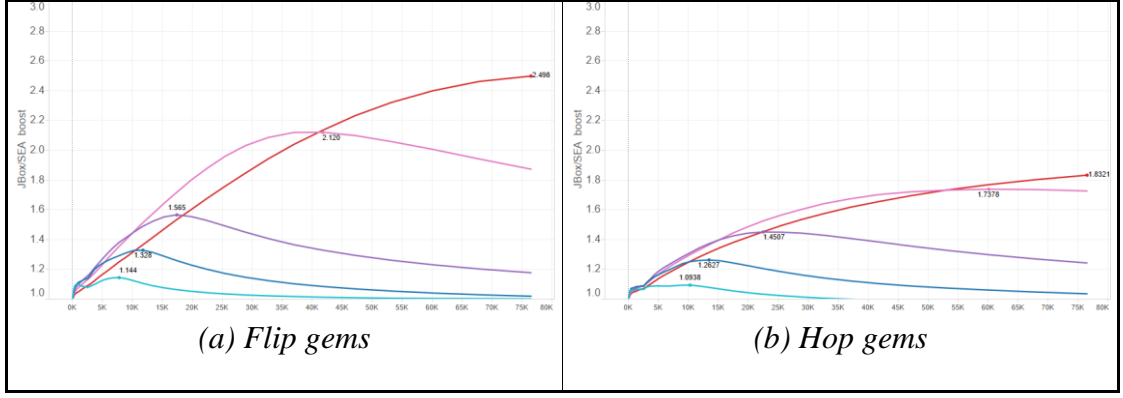


Figure 6.17. Fitness boost from Flip and Hop gems.

The key point is that, while the charts show different impacts for Flip and Hop gems, the difference is far less than we expected from their match probabilities.

6.5.2 Flip and Hop gem application rates

Figure 6.18 plots the number of gems in the jewellery-box for different graph sizes.

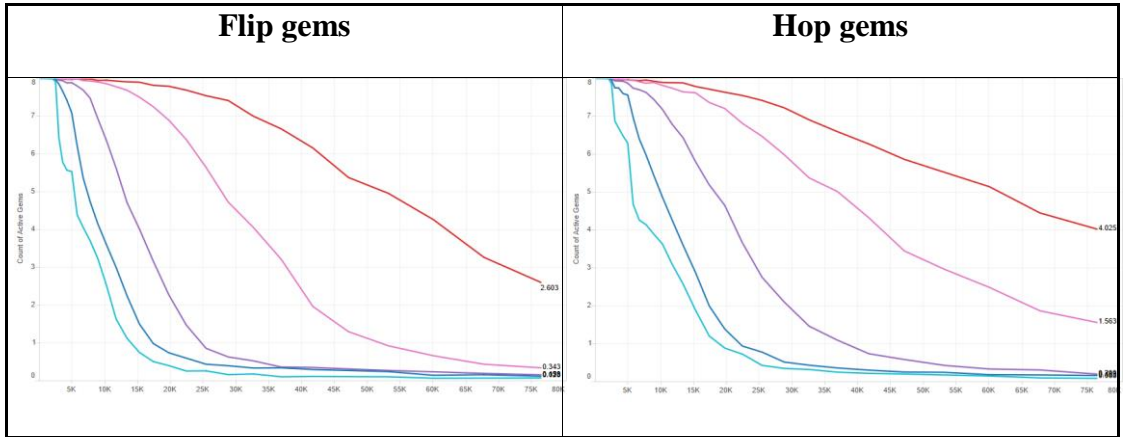


Figure 6.18. Numbers of Flip and Hop gems (population-size: 1000).

The key point is that the number of Flip and Hop gems are very similar in scale and shape. Thus we can rule out one explanation for the high frequency of Hop-gem applications: it is *not* because there are more Hop gems in the jewellery-box.

Figure 6.19 shows the average number of applications *per gem*.

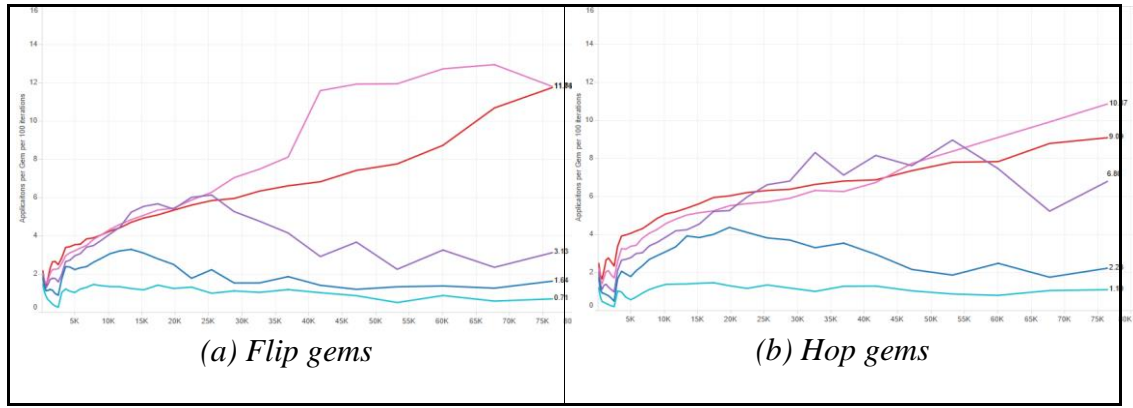


Figure 6.19. Application rates per Flip and Hop gem.

Application rates are quite similar for both gem types. Table 6.1 summarises the large disparity between match probabilities and observed results.⁵¹

Graph size →	21	42	76	150	280
Flip/Hop match probability ratios ⁵²	8.5	19.0	36.5	73.1	135.7
Flip/Hop observed match/gem ratios	1.02	0.78	0.89	1.04	0.92

Table 6.1. Ratio of applications per Flip / Hop gem.

This is the key finding that begs an explanation. Why are Flip and Hop gems' match frequencies so similar?

6.5.3 Conclusions

Results show that the number of applications per gem is indeed far higher than expected, and the difference between Flip and Hop gem application frequency is far lower than expected. Section 6.7 proposes the Clan Theory as an explanation.

None of the above should distract from the main result: gems confer a significant fitness boost to the EA process.

⁵¹ Each observed value is based on 26,400 data points.

⁵² See table 3.1.

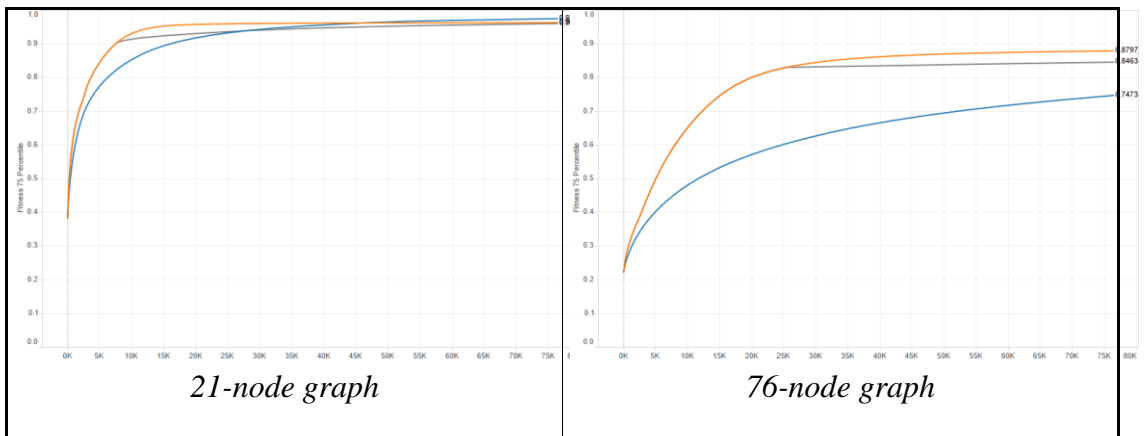
6.6 Will a Hybrid process yield the gems' gains, yet avoid premature convergence?

The hybrid process applies gems during the early stages, then switches them off in hope of avoiding premature convergence.⁵³ Gem activity and premature convergence occur earlier for small graphs, so gems are switched off earlier for smaller graph.

Graph size	Switch-off stage	Iteration
21	15	7,761
42	20	15,289
76	24	11,746
150	28	19,773
280	30	53,217

Table 6.2: Gem switch-off stages for Hybrid process.

Figure 6.20 shows the Gem and SEA processes as orange and blue lines, as usual. The Hybrid process' fitness is shown as a grey line branching from the gem line at the switch-off stage.



⁵³ Premature convergence is already reduced by the algorithm changes for Phase 2.

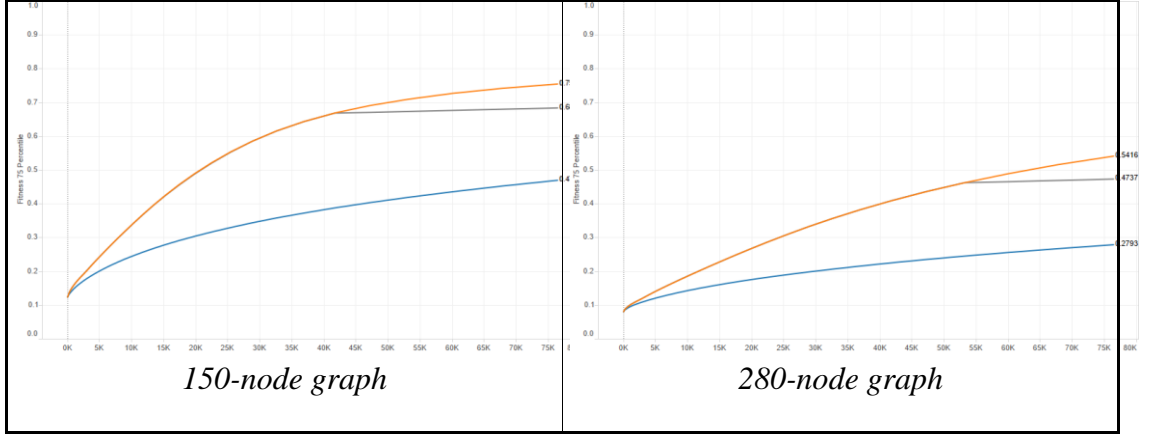


Figure 6.20. Hybrid process results.

The results are disappointing: the hybrid process barely improves fitness at all after gems are switched off. We thought this might be because gems were switched off too late (the switch-off stages are little more than educated guesses). Therefore we re-ran the experiments with earlier switch-off stages. Figure 6.21 shows the results for graph-sizes 150 and 280.

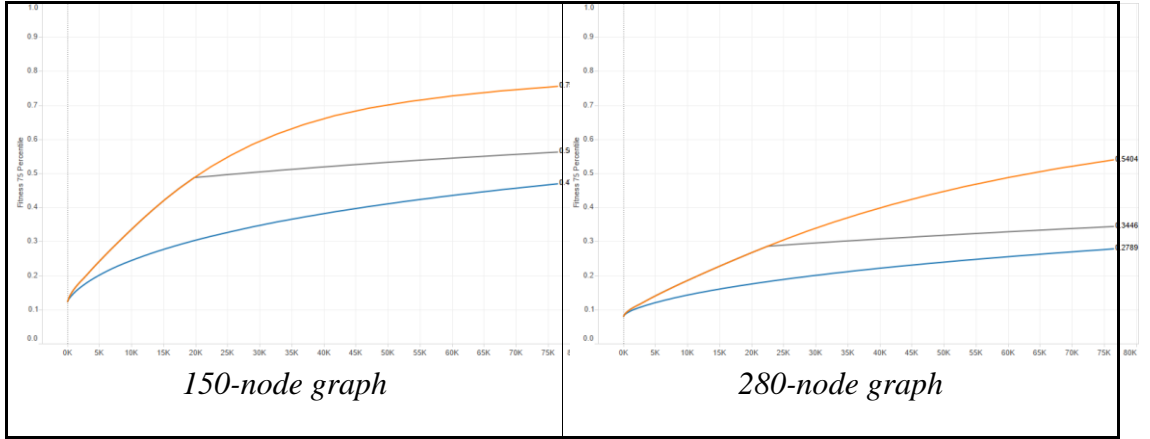


Figure 6.21. Hybrid with earlier switch-off

Even with earlier switch-off stages, the hybrid process has low fitness improvement. Therefore we decided to abandon the Hybrid algorithm.

We had considered another possibility: to toggle the method's status on/ off at every stage. However, figures 6.20 and 6.21 indicate that this is unlikely to work: the grey lines are clearly flattening even with early-stage switch-off, and the process does not recover even after many iterations.

We also considered a different approach to preserve diversity: to enforce a minimum iterations gap between two applications of the same gem. While we think this has potential, we did not implement this in Phase 2, due to scope constraints.

At a higher level, the gems' boost begs the question: "why worry about losing diversity, if the result is higher fitness?" With this consolation, we move on to an explanation of gem dynamics and impacts.

6.7 The Clan theory: an explanation of gem behaviours

6.7.1 Clans

Our explanation is based on the concept of *clans* of individuals. A clan is a group of individuals related by descent, and therefore likely to be similar through inheritance.

Figure 6.22 shows three seed individuals and their descendants. Individuals 1 and 3 are fitter than 2. They have more descendants, which are also fitter than average.

For example, the blue individuals form a clan descended from individual 1, and are similar. The closer the relationship, the closer the similarity. Thus individuals 111 and 112 are likely to be more similar to each other, than either is to individual 12.

Gem G1 is created when individual 111 is cloned and mutated to individual 1111.

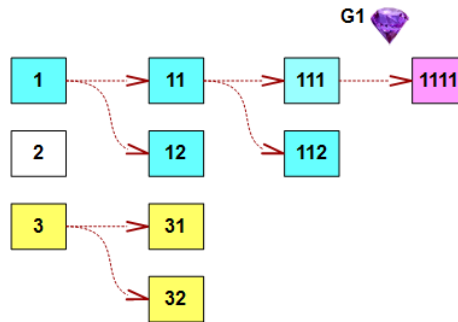


Figure 6.22. Clans formed by family trees.

Note that neither Individual 1111, nor any of its descendants, is compatible with G1. This is because their chromosomes contain the *post*-mutation alleles⁵⁴, whereas compatibility requires the *pre*-mutation alleles.

6.7.2 Clans as an explanation

The explanation is as follows:

⁵⁴ Gene values.

- a. Individual 111 was selected as a parent, so it is likely to be fitter than average. Therefore it is likely to be selected as a parent again, and since it is compatible with G1, the gem is likely to be applied again.
- b. Other individuals descended from 1 are also likely to be fitter than average, and therefore selected as parents. They are also likely to be similar to individual 111, and so are likely to be compatible with G1. Therefore G1 is likely to be applied again.
- c. Individual 1111 is even fitter than Individual 111, so it is even more likely to be selected as a parent. It soon creates a "sub-clan", both by descent and by re-application of G1 (a. and b. above). When a new gem, G2, arises in the sub-clan, the population already contains many compatible individuals. This combination of similarity by horizontal transmission, and by descent, is a positive feedback loop that accelerates the phenomenon.

In the SEA, we can imagine individuals moving around the fitness landscape (figure 2.3) independently. In clan theory, however, we imagine groups of similar individuals following each other around the landscape like sheep. An individual that undergoes a good mutation seems to pull its clan in the same direction. With hindsight, we can describe the change of gem-application logic as follows⁵⁵:

Phase 1: Individuals in a clan try to follow the leader, and usually they can. Only if they cannot, do they explore the landscape. Consequently, the individuals in a clan follow each other closely, and are very similar. This leads to premature convergence.

Phase 2: Individuals explore first, and only if unsuccessful do they try to follow the leader. This increases the opportunity to explore, so the clans are slightly looser. This reduces premature convergence.

The population is rapidly dominated by a few large clans. It seems likely that the jewellery-box becomes attuned to these clans, so that each clan has its own few gems in a "mini" jewellery-box.

Clans seem to explain some observed behaviours, notably:

⁵⁵ See §5.9.1 and figure 6.1.

The premature convergence seen in Phase 1.

The failure of the Hybrid process (figure 6.20). A few clans grow to dominate so quickly that when gems are switched off, the standard EA has insufficient diversity to work well on.

But the wisdom of hindsight is cheap: it is all too easy to find justification after the fact. Rigorous proof of the clan theory (*if* it is true) would require tracing the descent of each individual, which is out-of-scope here.

Fortunately, the theory has predictive power that can be tested empirically.

6.7.3 Clan theory predictions.

Clan theory predicts that several measures will differ from expectations without the clan phenomenon.⁵⁶ These predictions are qualitative: they predict the *direction* of difference, rather than the scale.

1. Gem application should be more frequent than expected by the match probabilities.

The clan concept includes positive feedback effects, which causes the phenomenon to grow rapidly as the process proceeds. Thus the number of applications per gem will be far more frequent than the match probabilities (table 3.1) would predict.

Many experimental results fulfil this prediction, including: Few natural individuals (§6.4.2), High saturation (§6.4.3), and Average applications per gem (§6.3.2 and §6.5.2, regardless of Flip vs. Hop).

2. Less gem application in the early stages.

The positive feedback takes time to grow. Therefore the theory predicts less gem application in the early stages, than later.

⁵⁶ Some behaviour arises from natural EA dynamics, such as impact of population size, different mutation types, and the root-cause analysis based on the probability of "good" mutations depending on fitness.

Table 6.3 shows the percentage of new individuals created by gems (rather than random mutations). As predicted, this percentage rises after the early stages.

Stages → Graph size	1 – 6	7 – 12	13 – 18	19 – 24	25 – 33
21	7%	11%	26%	33%	25%
42	11%	22%	42%	54%	35%
76	13%	24%	46%	66%	54%
150	16%	27%	46%	68%	76%
280	18%	30%	45%	64%	79%

Table 6.3: Percentage of individuals created by gems, at different stages.

3. Little difference between Hop and Flip gem application.

Gem application rates and saturation are so high, that they should dominate many other factors, including the difference between Flip and Hop gem application.

This prediction is borne out in Phase 1 (§5.5), and Phase 2 (§6.5). In fact, those results were one of the main motivations for the clan theory.

4. Population size.

Large populations have more diversity, which should dilute the clan effect to some extent. Therefore the Flip/ Hop ratio should be closer to the expected probability ratio (i.e. larger), in larger populations.

Tables 6.4 shows that this prediction is correct, although the differences between the population sizes are smaller than anticipated. This may be due to the strong positive feedback effect dominating other factors including population size.

Graph size →	21	42	76	150	280
Flip/Hop observed match/gem ratios, population = 100	0.999	0.808	0.897	0.995	0.899

Flip/Hop observed match/gem ratios, population = 1000	1.110	0.853	0.911	1.031	0.931
--	-------	-------	-------	-------	-------

Table 6.4. Ratio of applications per Flip / Hop gem, by Population-size

5. Clan effects should be larger with larger chromosome size.

Each Hop mutation perturbs 3 arcs (genes). Also, the Hop gem's compatibility test requires 3 arcs to match.⁵⁷

Let I be an individual tour of the smallest graph, with a chromosome of length 21. Suppose that I is compatible with gem G . Let $m(I) = I'$.

Probability(I' is compatible with G)

$$\begin{aligned}
 &= P(\text{mutation did not change any of the 3 arcs required to match}) \\
 &= P(\text{mutated gene 1} \neq \text{Arc 1, 2, or 3}) \times P(\text{mutated gene 2} \neq \text{Arc 1, 2, or 3}) \times \\
 &\quad P(\text{mutated gene 3} \neq \text{Arc 1, 2, or 3}) \\
 &= (18 / 21) \times (17 / 20) \times (16 / 19) = 61\%.
 \end{aligned}$$

The probability that an individual is still compatible after 10 independent mutations is $(61\%)^{10} = 0.76\%$ (shown as 1% in table 6.5).

Table 6.5 shows the match probabilities for all five graph sizes.

Graph size	Pr(match = True) after 1 mutation	Pr(match = True) after 10 mutations
21	61%	1%
42	80%	10%
76	88%	29%
150	94%	54%
280	97%	72%

Table 6.5: Probability of matching a gem after 1 mutation, and after 10 mutations.

⁵⁷ Similar concepts apply to Flip gems, but it is more complicated because a Flip mutation also reverses all the arcs within the flipped section.

The larger the chromosome, the more likely it is to maintain compatibility after mutation(s). Therefore we predict that the clan effect is larger for larger chromosomes.

Figure 6.23 shows the average number of application per gem. As predicted, this is much smaller for the 21-node graph than for the 280-node graph. The differences may seem less than expected from the right-most column in table 6.3, but recall that the Hop match-probability for the 21-node graph is over 3,000 times higher than for the 280-node graph (table 3.1).

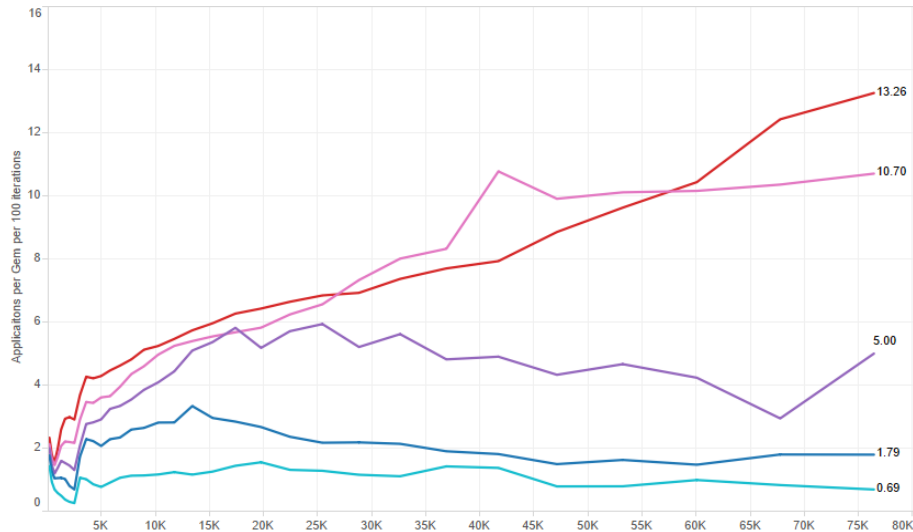


Figure 6.23. Average number of applications per gem.⁵⁸

Figure 6.15 showed the rapid decrease of "pure" individuals. As expected by the "chromosome size" effect, the decrease is far faster for larger graph sizes.

In summary, the clan theory explains some behaviour in hindsight, and its predictions are supported by empirical results. Although unproven, we propose it as the best explanation of observed gem behaviour.

⁵⁸ These lines are quite erratic, and interpolation between points is dubious. This highlights the smoothness of the lines in the other charts.

6.8 Performance

6.8.1 Proof-of-concept platform

The proof-of-concept experiments were re-run with 200 trials for each case, to record run-times. All ran on a Lenovo G50-70 laptop with:

OS: Windows 10 Home 64-bit. **Processor:** 64-bit Intel Core i3 CPU 1.7 GHz.

RAM: 8 GB.

All software was written in Java version 8 from Oracle Corporation.

All data was recorded in MySQL version 6.3.

6.8.2 Comparison of SEA and Gem methods' runtimes.

We ran time-trials on two machines for thoroughness. Figure 6.24 shows the mean number of milliseconds to run the process, with blue bars for the SEA, and orange bars for the Gem method.

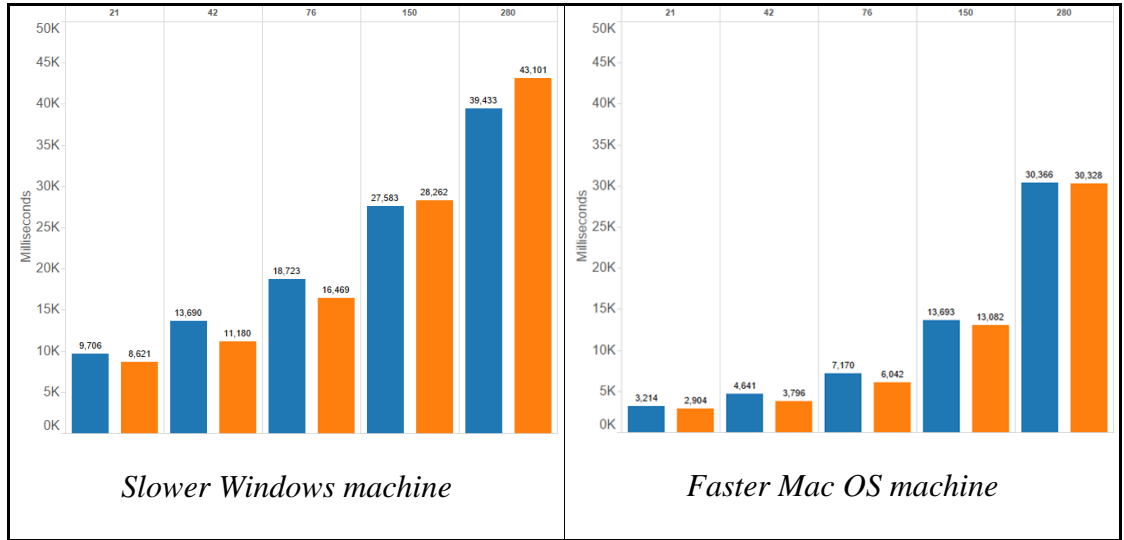


Figure 6.24. SEA and Gem methods' runtimes (Random seeds).

Random seed graph sizes:	21	42	76	150	280
A. Gem/SEA runtime ratio ⁵⁹ :	0.89	0.82	0.88	1.02	1.09

⁵⁹ Based on the Windows figures, which are less favourable to Gems, for caution.

B. Maximum Gem boost: ⁶⁰	1.12	1.29	1.50	1.91	2.17
C. (Max-boost) / (Runtime-ratio):	1.26	1.58	1.70	1.87	1.98

Table 6.6. Gem / SEA run-times per graph size.⁶¹

The A. ratios are very favourable. The gem method takes a little longer for larger problem instances, but these are the problems that gain most from the method. The gains are much higher than the additional time cost. For example, the time-cost ratio is 1.09 for graph-size 280, yet its fitness is 2.17 times higher. Row C shows the "benefit-cost" ratio: the larger the problem instance, the better the ratio is.

Table 6.7 compares run-times for a selection of other cases:

Case	A. (Gem/ SEA) time	B. Max Gem boost	C. (Max-Boost) / (time-ratio)
1. 150-node, Random seed, Flip .	1.00	2.12	2.12
2. 150-node, Random seed, Hop .	1.08	1.74	1.61
3. 150-node, Random seed, Population: 100 .	1.02	1.44	1.41
4. 150-node, Random seed, Population: 1000 .	1.03	2.44	2.37
5. 42 -node, Greedy seed.	0.88	1.10	1.25
6. 280 -node, Greedy seed.	1.01	1.08	1.07

Table 6.7. Selected comparisons of SEA and Gem run-times.

Observations:

- Cases 1 and 2 compare runtimes for Flip and Hop gem cases. Again, the time cost is small compared to the fitness boost. Flip mutations and gems give

⁶⁰ See figure 6.4.

⁶¹ Each A. figure is a mean of 1,600 data points for Gems, and 800 for SEA.

Each B. figure is a mean of 800 data points.

better fitness results generally, so it is pleasing to see that Gem and SEA runtimes are identical in this case.

- b. Cases 3 and 4 compare small and large population sizes. The boost is significantly higher for large populations, and the runtime difference is small.
- c. Cases 5 and 6 focus on greedy-seed problems. Gems are faster in one case (0.88), and about the same in the other (1.01). The benefit ratios are positive, but not as high as in random-seed problems because the boosts are lower. Also, unlike Random-seed cases, the smaller problem has the best benefit ratio. Most of this difference is due to runtime rather than boost.

In summary, gems add little to the runtime, and the time cost is outweighed by the fitness boost.

7. Conclusions

7.1 Summary

This thesis proposed Gems as a novel method to augment standard EAs, and developed the concepts into well-defined models and algorithms.

A proof-of-concept system was implemented to test the new method, and compare it to the Standard EA process. Results showed that the method confers a significant fitness improvement, as shown in figure 6.4, repeated below:

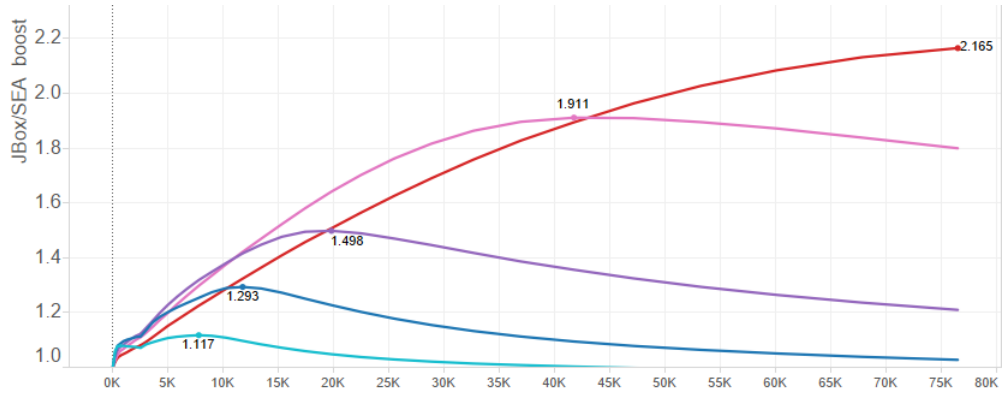


Figure 7.1. Fitness boost during process.

Results also showed that the method yields larger improvements for random-seed populations, and for larger problem instances. This is welcome, since these are the most difficult problems.

General problem characteristics were identified to indicate when gems may be used. Gem dynamics were investigated, and the behaviours were explained, culminating in the clan theory.

Runtime comparisons showed that the fitness improvements easily outweigh the small performance cost.

7.2 Contributions

7.2.1 To academic literature and theory

This thesis considerably extends EA theory by augmenting it with the new Gem method, describing the concepts, models, objects (almost to the level of Abstract Data Types), definitions, the importance of gem compatibility (and finding that

partial compatibility is not feasible), and detailed algorithms (including the improved gem application logic to reduce premature convergence).

It further contributes a check-list of characteristics that indicate if a problem-domain is amenable to the method.

The results also provide insight to other aspects, notably: factors that impact the method's behaviour, impact on fitness variance, gem dynamics, and saturation, and raise questions about the importance of diversity, which is only a means to an end.

Potential concerns about the probability of match, especially for larger problem instances with long chromosomes, were alleviated, at least for TSP.

Proof-of-concept results also bolster the literature comparing the merits of mutation and cross-over. Even with no cross-over, mutation provides good fitness improvements.

The fitness boost measure may be generalised for other EA research that compares fitness across problem instances with different optima, and scenarios with different dynamics.

The clan theory was presented as an explanation of observed behaviour. Although not proven, the theory is cohesive, quite simple, intuitively appealing, and its predictions are fulfilled by results.

We believe that gems may be related to, and provide insight about, the leading EA theory: Schema theory (Holland, 1975). Like schema, gems represent small groups of alleles that propagate rapidly through the population, and are most likely found in high-fitness individuals.

7.2.2 To practice

Future EA researchers may find the logging techniques of interest, particularly (a) the geometrically increasing stage widths and (b) the data design for multi-factor experiments, and algorithm 4.3 to loop through the experimental cases.

Gems certainly provide significant benefits when applied to the TSP, and runtime results show that the method's cost-benefit ratio is very favourable. This should encourage use, especially when only random seed populations are feasible, since the results are particularly favourable here.

The jewellery-box class implementation, gem representation, gem value formula, and algorithms implemented, may be used by other researchers, both in TSP and as guides for other problem domains⁶². The efficient algorithms shown in Appendix 9.1 may be used by others working on the TSP and similar graph problems.

The proof-of-concept highlighted important factors, and indicated how the process may be tuned with certain parameters (e.g. population size and tournament size to control selection pressure, and gem lifetime to control saturation).

Presenting the failed hybrid algorithm means that others can avoid it in future.

Results alleviated early concerns about too few applications for gems with complex mutation operators. Therefore others can feel reasonably confident in using the method with complex mutations.

7.2.3 Novelty

The literature search shows that this thesis introduces many novelties, most notably: (a) the gem concept, (b) the method and models in detail, and (c) clan theory as an explanation of observed behaviour.

The geometrically increasing log stages, and some efficient algorithms, may also be considered novel⁶³, but are of lesser importance.

7.3 Limitations and future work

The thesis scope was limited by time and space constraints. The proof-of-concept was primarily intended to show that gems work, and investigate the scenarios that it worked best in, using TSP problem instances.

The first limitation is that only TSP instances were tested. Proof-of-concepts in other problem domains would show if the method provides similar gains in other domains, and refine our knowledge of the characteristics of problems where the method can be applied fruitfully. These problems may be in the broader domain of graph-theory

⁶² Complete code is available on request.

⁶³ We did not see these in the literature review, but we were not searching for them.

(e.g. the Snowplough problem), combinatorial but outside graph-theory (e.g. the Knapsack problem), or beyond combinatorial (e.g. real-valued search-spaces), especially where no feasible greedy solutions are known.

Future research could investigate other TSP mutation operators, especially more complicated ones, and some with different numbers of parameters. This would guide design of generic gem objects, which could be implemented as Java interfaces, for example.

The next important limitation is that the proof-of-concept did not use cross-over. Implementing a proof-of-concept with cross-over could be very fruitful, not only because cross-over is widely used in EA practice, but also because recombining parents' genes should maintain more diversity in the population. It should also be quite simple, since gems do not effect cross-over operators. Performance comparisons would also be of interest here.

This thesis focused on improving EAs, on the assumption that EAs are suitable for solving certain classes of problems. However, for any given problem (such as TSP), there may be better solutions than EAs (such as Simulated Annealing). Therefore, before deciding to use the EA heuristic, the practitioner should ascertain if a different method would outperform EAs for the problem in hand. Given that we used TSP as the test problem, it would be worthwhile surveying other approaches to solving TSP.

The Phase 2 logic change helped to preserve diversity. Other ideas to preserve diversity may be as good, or better. For example, the minimum iteration-gap between gem applications has potential, and is easy to implement.⁶⁴

Several parameter settings can be used to tune the process' behaviour. Many of these were not tried in the proof-of-concept, so this may be a useful avenue of research. Examples of such settings include: tournament-size and population-size to tune selection pressure, and maximum gem usage and gem lifetime to tune saturation.

As already noted, gems may shed light on Holland's Schema theory, especially schema dynamics. Research in this area could be very interesting.

⁶⁴ This was implemented in an early version, but dropped as over-engineering for the proof-of-concept.

8. Selected bibliography

- Abdoun, Otman, Jaafar Abouchabaka, and Tajani Chakir. "Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem." *International Journal of Emerging Science and Engineering* 03, Bhopal, 2012.
- Baudrey, Benoit, Franck Fleurey, Jean-Marc Jezequel and Yves Le Traon. "From genetic to bacteriological algorithms for mutation-based testing." *Software Testing Verification and Reliability*. 06/2005; 15(2):73-96. Doi: 10.1002/stvr.313. John Wiley & Sons, NJ, 2005.
- Bessaou M. and Patrick Siarry. "A genetic algorithm with real-value coding to optimize multimodal continuous functions." *Structural and Multidisciplinary Optimization* 11/2001; 23(1):63-74, Springer, New York, 2001.
- Bhandari Dinabandhu, Murthy C. A., Sankar K. Pal. "Variance as a Stopping Criterion for Genetic Algorithms with Elitist Model." *Fundamenta Informaticae* 120: 145–164, IOS Press, Amsterdam, 2012.
- Blickle Tobias and Lothar Thiele. "A Mathematical Analysis of Tournament Selection." *Genetic Algorithms: Proceedings of the 6th International Conference (ICGA95)*, San Francisco, 1995.
- Brabazon, Anthony et al. "Natural Computing Algorithms", Springer, 2015.
- Cantu-Paz, E. "Selection intensity in genetic algorithms with generation gaps." *Genetic and Evolutionary Computation Conference*, Las Vegas, NV, July 2000.
- Chiong Raymond, Ooi Koon Beng. "A Comparison between Genetic Algorithms and Evolutionary Programming based on Cutting Stock Problem." *Engineering Letters*, 14:1, Hong Kong, February 2007
- Collingwood, Emma, David Corne, and Peter Ross "Useful Diversity via Multiploidy." *Proceedings of the IEEE International Conference on Evolutionary Computing*, Nagoya, Japan, 1996.
- Darwin, Charles. *The origin of species by means of natural selection*, London, 1859.
- Dawkins, Richard. *The Selfish Gene*. Oxford University Press, 1976.
- Deep, Kusum and Hadush Mebrahtu. "Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem." *International Journal of*

- Combinatorial Optimization Problems and Informatics, Vol. 2, No.3, 1-23.*
Moreles, Mexico, 2011.
- De Jong Kenneth A., Sarma Jayshree. Generation Gaps Revisited. *Foundations of Genetic Algorithms 2*, pp 19-28. Morgan Kauffmann, Massachusetts, 1992.
- De Jong, Kenneth A. "An Analysis of the Behavior of a Class of Genetic Adaptive Systems." Ph.D. diss., University of Michigan, 1975.
- Deng, Yong, Yang Liu, and Deyun Zhou. "An Improved Genetic Algorithm with Initial Population Strategy for Symmetric TSP." *Mathematical Problems in Engineering Volume 2015, Article ID 212794*, Hindawi, Cairo, 2015.
- Domingos, Pedro. "The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World". Basic Books, NY, 2015.
- Eshelman L. J. and J. D. Schaffer, "Preventing premature convergence in genetic algorithms by preventing incest", *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 115 – 122, University of California, San Diego, 1991.
- Falkenauer, E. and A. Delchambre. "A Genetic Algorithm for Bin Packing and Line Balancing." *Research Centre for Belgian Metalworking Industry*, Belgium, 1997.
- Ferreira Cândida. "Combinatorial Optimization by Gene Expression Programming: Inversion Revisited." *Proceedings of the Argentine Symposium on Artificial Intelligence*, pp 160-174, Santa Fe, Argentina, 2002.
- Fogel, Lawrence J., Owens A.J., and Walsh M.J.. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, NJ, 1966.
- Fogel, Lawrence J., *Evolutionary programming*, 1960.
- Friedrich, Tobias, Pietro S. Oliveto, Dirk Sudholt, and Carsten Witt. "Analysis of Diversity-Preserving Mechanisms for Global Exploration." *MIT Evolutionary Computation, Vol. 17, No. 4, Pp 455-476. (Doi:10.1162/evco.2009.17.4.17401)*, MIT, Massachusetts, 2009.
- Goldberg D. E. and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization", *Proceedings of the Second International Conference on Genetic algorithms and their application*, pp. 41-49. MIT, Massachusetts, 1987.

- Goldberg, D.E. and Deb, K. "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms." *Foundations of Genetic Algorithms, Volume 1*, pp 69–93. Doi:10.1016/B978-0-08-050684-5.50008-2. Springer, New York, 1991.
- Goldberg, D. E. "Real coded Genetic Algorithms, Virtual Alphabets and Blocking". *Complex Systems* 02/1995; 5(2). Champaign, IL, 1995.
- Gotshall, Stanley, and Bart Rylander. *Optimal Population Size and the Genetic Algorithm*. University of Portland, OR, USA. 1992.
- Grefenstette, John, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. "Genetic Algorithms for the Travelling Salesman Problem." *Proceedings of the First International Conference on Genetic Algorithms and their Applications*. Carnegie-Mellon University, PA. July 1985.
- Grefenstette John, "Incorporating Problem Specific Knowledge into Genetic Algorithms." *Genetic Algorithms and Simulating Annealing*, ed. Lawrence Davis. Morgan Kaufmann, San Francisco, 1987.
- Gupta, Deepti, and Shabina Ghafir. "An Overview of methods maintaining Diversity in Genetic Algorithms." *International Journal of Emerging Technology and Advanced Engineering*, ISSN 2250-2459, Volume 2, Issue 5, Bhopal, 2012.
- Hansen, Nikolaus, Dirk V. Arnold and Anne Auger. *Evolution Strategies* Doi: 10.1007/978-3-662-43505-2_44. February 2015.
- Haupt R. L. "Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors." *Antennas and Propagation Society International Symposium, 2000. IEEE, Volume: 2*. Salt Lake City, 2000.
- Herreraa Francisco, Manuel Lozanoa, and J.L. Verdegay. "Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis". *Artificial Intelligence Review* Volume 12, Issue 4, pp 265-319. Springer, New York, 1998.
- Herreraa Francisco, Manuel Lozanoa, and José Ramón Canob "Replacement strategies to preserve useful diversity in steady-state genetic algorithms." *Information Sciences*, Volume 178, Issue 23. Amsterdam, 2008.
- Holland, John. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, University of Michigan Press, 1975.

Jassadapakorn, Chaiwat and Prabhas Chongstitvatana. "Self adaptation mechanism to control the diversity of the population in Genetic Algorithm." *International Journal of Computer Science & Information Technology (IJCSIT) Vol 3, No 4*, Mauritius, 2011.

JCLEC: A public library of generic EA functionality, freely available through SourceForge under the GNU General Public License (GPL).
<http://jclec.sourceforge.net/> , 2016.

Koljonen, Janne and Jarmo T. Alander. "Effects of population size and relative elitism on optimization speed and reliability of genetic algorithms." *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*. Helsinki, 2006.

Larrañaga P., C. M. H. Kuijpers, R.H. Murga, I. Inza , and S. Dizdarevic. "Genetic Algorithms for the Travelling Salesman Problem: A review of representations and operators." *Artificial Intelligence Review, vol 13, 129-170*. Springer, New York, 1999.

Malthus, Thomas. *An Essay on the Principle of Population*. London. 1798.

McGarraghy, Sean. "Natural Computing" MIS40530 course notes, MSc Business Analytics, Smurfit Business School. Dublin. October 2014.

Mendel, Gregor Johann. *Experiments on Plant Hybridization*. 1865.

Mitchell, Melanie, Stephanie Forest, and John Holland. "The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance." *Proceedings of the First European Conference on Artificial Life*, Paris, 1991.

Noever, David, and Subbiah Baskaran. *Steady-State vs. Generational Genetic Algorithms : A Comparison of Time Complexity and Convergence Properties*. Santa Fe Institute Working Paper 1992-07-032, 1992.

Oliveto Pietro, Jun He, and Xin Yao. "Time Complexity of Evolutionary Algorithms for Combinatorial Optimization: A Decade of Results." *International Journal of Automation and Computing* 06/2007; 4(3):281-293. Doi: 10.1007/s11633-007-0281-3. Springer, New York, 2007.

- Oman Stephen, Cunningham Pádraig. "Using Case Retrieval to Seed Genetic Algorithms." *International Journal of Computational Intelligence and Applications, Vol 1, 71-82*. Atlantis Press, Paris, 2001.
- Osaba, Eneka, Roberto Carballedo, Fernando Diaz, Enrique Onieva, Idoia de la Iglesia, and Asier Perallos. "Crossover versus Mutation: A Comparative Analysis of the Evolutionary Strategy of Genetic Algorithms Applied to Combinatorial Optimization Problems." *The Scientific World Journal, Volume 2014, Article ID 154676*. Hindawi, Cairo, 2014.
- Rechenberg, Ingo. "Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution" (PhD. diss.). Technical University of Berlin, 1971.
- Roewa, Olympia. Fidanova, Stefka. Paprzycki, Marcin. "Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling." *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems pp. 371–376*. Kraków, 2013.
- Rogers Alex and Adam Priigel-Bennett. Modelling the Dynamics of a Steady State Genetic Algorithm. *Foundations of Genetic Algorithms - 5, 57-68*. Springer, New York, 1999.
- Shimodaira Hisashi, "DCGA: A Diversity Control Oriented Genetic Algorithm." *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence, Doi: 10.1109/TAI.1997.632277*. Herndon, VA, 1997.
- Smith Jim and Frank Vavak. "Replacement Strategies in Steady State Genetic Algorithms: Static Environments." Banzhaf, W. and Reeves, C., eds. *Foundations of Genetic Algorithms 5, pp. 219-234*. Morgan Kaufmann, San Francisco, 1999.
- Syswerda, Gilbert. "A Study of Reproduction in Generational and Steady-State Genetic Algorithms." *Foundations of Genetic Algorithms, Volume 1, Pp 94–101*. Morgan Kaufmann, San Francisco, 1999.
- TSPLIB, Reinelt Gerhard, Universität Heidelberg, Heidelberg, Germany. Accessed 2016. <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>

Ting C. K. and H. K. Buning, "A mating strategy for multi-Parent genetic algorithms by integrating Tabu search." *Proceedings of the 2003 Congress on Evolutionary Computation*, pp. 1259 - 1266. Canberra, 2003.

Vrajitoru Dana. "Large populations or Many generations for Genetic Algorithms? Implications in Information Retrieval." *Doi: 10.1007/978-3-7908-1849-9_9*. University of Neuchatel, Switzerland, 1999.

Wainwright, Roger. *Introduction to Genetic Algorithms Theory and Applications*. The University of Tulsa, OK, 1993.

Whitley, D. and Kauth, J. "Genitor: A Different Genetic Algorithm", *Proceedings 4th Rocky Mountain Conference on Artificial Intelligence*, Denver, 1988.

Xie, Huayang and Mengjie Zhang. "Tuning Selection Pressure in Tournament Selection." School of Engineering and Computer Science, Victoria University of Wellington, 2009.

9. Appendices

9.1 Order-of-Complexity workings

9.1.1 Efficient calculation of mutated offspring's fitness

Let I be the parent, $C(I)$ = cost of the parent tour, and $m(I) = I'$. The proof-of-concept calculates $f(I')$ using only the cost of the arcs changed by the mutation, as follows:

Let $C(A_x)$ = cost of Arc X .

By definition, $\text{cost}(I) = \sum C(A_x)$, over all arcs in the tour ($x = 1 \dots N$).

Let the number of arcs changed by a mutation = M .

$\Rightarrow C(I') = C(I) - \sum C(M \text{ old arcs removed}) + \sum C(M \text{ new arcs added})$.

$\Rightarrow F(I') = 1/C(I')$.

Algorithm 9.1. Efficient calculation of offspring's fitness.

Since $C(I)$ is known in advance at line 4, we only need $2M$ arc-costs to calculate $C(I')$. Since the number of arcs effected by a mutation is much smaller than the number in a tour⁶⁵, this is a major processing efficiency per mutation, and since it occurs in the innermost loop of the process, it is a major efficiency overall.⁶⁶

This may seem obvious, but it is worth pointing out, since Abdoun et al. (2012) sum *all* the arc-costs for every offspring.

Note that such efficiencies may not exist for other problems, even within the graph domain. Sometimes fitness must be completely re-calculated for each offspring.

9.1.2 Efficient finding of each arc's cost.

Suppose that the five USA cities in figure 1.2 are the complete graph. Conceptually, the arc-costs can be stored in a 5×5 matrix, as follows:

	Boston	New York	Philadelphia	Los Angeles	San Diego
--	--------	----------	--------------	-------------	-----------

⁶⁵ $M = 2$ or 3 versus $N = 42$ to 280 in the proof-of-concept.

⁶⁶ The proof-of-concept uses 76,500 iterations.

Boston	0	306	436	4,174	4,155
New York	306	0	130	3,940	3,912
Philadelphia	436	130	0	3,848	3,815
Los Angeles	4,174	3,940	3,848	0	180
San Diego	4,155	3,912	3,815	180	0

Table 9.1. Distances (miles) between US cities, as the crow flies

A triangular matrix is sufficient for a bidirectional graph, and the 0 "self-to-self" costs are redundant. The proof-of-concept stores all arc costs in advance, in an array of size $N(N-1)/2$

To find the cost of arc A_{xy} (the arc between nodes x and y):

```

Algorithm: double arcCost(x, y)
If x > y
    Swap x and y ;
arcIndex = (x * N) + y ;      // N = number of nodes.
return arcArray[arcIndex] ;  // Direct array access.

```

Algorithm 9.2. Efficient fetching of arc's cost.

Given (1) population size = P , (2) graph size = N , (3) the number of arcs in the seed population = PN , (4) number of arcs effected by a mutation = M , and (5) number of iterations = X , then:

The number of arc-costs used per EA run = $PN + 2MX$.

Typically M is small, X is large, and P and N lie between M and X .

The smallest proof-of-concept case has:

$P = 100$, $N = 21$, $M = 2$, and $X = 76,442$.

Count of arc-costs used = $2,100 + 152,884 = 154,984$.

Arc-cost array-size: $21 \times 20 / 2 = 210$.

The largest proof-of-concept case has:

$P = 1000$, $N = 280$, $M = 3$, and $X = 76,442$.

Count of arc-costs used = $280,000 + 229,326 = 509,326$.

Arc-cost array-size: $280 \times 279 / 2 = 39,060$.

In either case, pre-loading the arc-costs into an array for use in the algorithm, with $O(1)$ access to array values, is very efficient. Even with double values, the array does not take much memory on the proof-of-concept machine.

This does not scale well to larger graphs because the array would consume too much memory. For example, a graph with one million nodes would store about 500 billion values in memory.

9.1.3 Standard EA order-of-complexity

Table 9.2 shows the order-of-complexity workings for the SEA, with steps numbered from the flowchart in figure 2.1. The symbols used are:

- N = Number of nodes in the problem instance.
- P = Population size.
- T = Tournament size.
- R = Generate random number.
- M = Number of parameters for a mutation.
- X = Number of iterations.
- J = Maximum number of gems in the jewellery-box.

#	Algorithm step & notes	Computational cost	O()
1	Create seed population. Done once, before EA process iterations.	N^2P	$O(N^2P)$
2	Select parent.	TR	$O(T)^{67}$

⁶⁷ $O(R) = 1$.

3	Generate child as a clone with the same number of nodes.	N	O(N)
4	Mutate child. Flip: Generate 2 random numbers. Change N/2 arcs. Hop: Generate 2 random numbers. Change 3 arcs.	2 + N/2. 2 + 3.	O(N) O(1)
4.1	Compute child's fitness. Flip: $2 \times 2 = 4$ arc-lengths changed. Hop: $2 \times 3 = 6$ arc-lengths changed.	4. 6.	O(1)
5	Test $f(I') > f(I)$.	1	O(1)
6	Replace weakest individual with I'. Also refresh population's minimum fitness.	1 + N	O(N)
7	Test stop criteria (iteration > max iteration).	1	O(1)
Σ	Total SEA time complexity	$N^2P +$ $X (TR + N + 2$ $+ N/2 + 4 + 1 +$ $1 + N + 1)$	$O(N^2P + X(T + N))$

Table 9.2. Order-of-complexity workings for the SEA.

9.1.4 Additional work for gems

Numbering is from the gem method flowchart in figure 3.1.

#	Algorithm step & notes	Computational cost	O()
4.2	Set Mutation.Value = $f(I') - f(I)$ (both already known). Test if Mutation.Value > 0.	1 1	O(1)

4.3	Test if new gem. If true, do step 4.4.	1	O(1)
4.4	Insert new gem: (i) Make gem, (ii) Insert, (iii) refresh min value gem.	Flip: $4 + 1 + J$. Hop: $5 + 1 + J$.	O(J)
4.5	Gem match: test the nodes for compatibility with each gem. If match found, do step 4.6.	Flip: $2J$. Hop: $3J$.	O(J)
4.6	Apply existing gem from jewellery-box. Calculate fitness ($2 \times M$).	Flip: $N/2 + 4$. Hop: $3 + 6$.	O(N)
Σ	Total additional Gem time complexity.	$3 + 2J + N/2 + 4$ (worst case)⁶⁸	O(J + N)

Table 9.3. Order-of-complexity workings for Gems.

Gems are applied within the iterations loop, so the additional time complexity is $O(X(J+N))$. Therefore the gem method's time complexity is:

$$O(N^2P + X(T + N + J + N)) = O(N^2P + X(T + N + J)).$$

Since T and J are typically small compared to N, N dominates the complexity within the iteration loop. Therefore the time complexity for both the SEA and Gems is approximately $O(N^2P + XN)$.

Note that, although $O(R) = 1$, each random-number generated may be expensive, and each random mutation requires at least two random numbers. Gems reduce the frequency of random-number generation, because gem applications replace some random mutations.

9.2 General case for Gem Value = δF

Let I_1 and I_2 be two individuals, with I_1 created in an earlier iteration than I_2 , and $c(I_1) > c(I_2)$ (as is typical).

Write $\delta C_1 = c(I_1') - c(I_1)$ and $\delta C_2 = c(I_2') - c(I_2)$, and suppose that the cost-reduction is the same in both cases, i.e. $\delta C_1 = \delta C_2$.

⁶⁸ Worst case is steps 4.4 + 4.5 (rather than 4.2 + 4.3), with a Flip gem.

Write $\delta F_1 = f(I_1') - f(I_1)$ and $\delta F_2 = f(I_2') - f(I_2)$.

$$\Rightarrow \delta F_1 = 1/c(I_1') - 1/c(I_1) = (c(I_1) - c(I_1')) / (c(I_1') * c(I_1)) = \delta C_1 / (c(I_1') * c(I_1))$$

$$\text{And } \delta F_2 = 1/c(I_2') - 1/c(I_2) = (c(I_2) - c(I_2')) / (c(I_2') * c(I_2)) = \delta C_2 / (c(I_2') * c(I_2))$$

Now $\delta C_1 = \delta C_2$

$$\Rightarrow \delta F_2 = \delta C_1 / (c(I_2') * c(I_2)).$$

$$\Rightarrow \delta F_2 / \delta F_1 = (c(I_1') * c(I_1)) / (c(I_2') * c(I_2))$$

Recall that $c(I_1) > c(I_2)$

$$\Rightarrow c(I_1') > c(I_2')$$

$$\Rightarrow \delta F_2 / \delta F_1 = (x / y) \text{ with } x > y.$$

$$\Rightarrow \delta F_2 > \delta F_1.$$