5 EXPERIMENT: LOCAL SEARCH

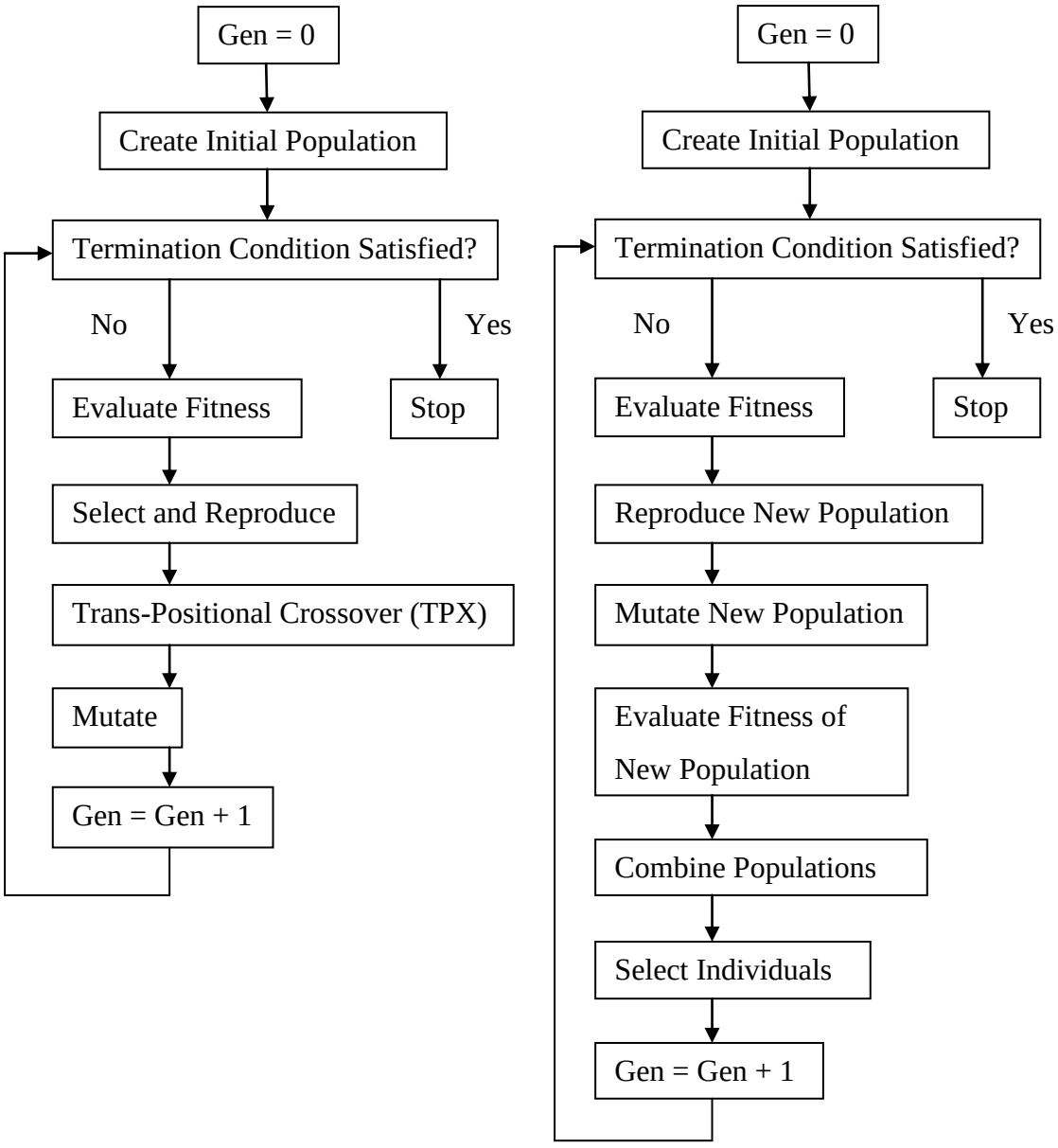
5.1 Background

Experiments in this chapter investigate two related research aims i.e. how to represent the design problem and early lifecycle design solution for effective computational search, and how to effectively explore and exploit the software design solution search space to arrive at useful and innovative UML class designs. Thus firstly, an initial set of experiments is conducted to trial the performance of the object-based representation using manual parameter 'tuning'. Building on the findings of initial experiments, a second set of experiments is then performed to investigate how dynamic parameter control might more effectively explore and exploit the local search space.

With respect to terminology, 'local search' is widely referred to as a metaheuristic for solving computationally hard optimisation problems. As such, local search can be applied to problems that can be formulated as finding a solution optimising a criterion among a number of candidate solutions. Starting at a candidate solution, the search typically moves iteratively to a neighbouring solution (e.g. 'hill-climbing'), hence the name local search. However, when using a discrete, object-based representation, the notion of candidate solution neighbourhood with fitness gradients is perhaps less applicable. Thus while the term local search is used in this thesis, its use in essence refers to '*localised'* single-objective search in order to contrast it with the multi-objective global search described in the following chapter.

With regard to the initial set of experiments, local search approaches inspired by genetic algorithms (e.g. De Jong, 1975, Goldberg, 1989) and evolutionary programming (e.g. Fogel *et al.*, 1966) are trialled. Figure 5.1 shows flow charts of the genetic algorithm (GA) and evolutionary programming (EP) inspired evolutionary approaches used in experiments.

The GA-inspired approach utilises the genetic operators described in chapter 4. To briefly recap, selection is performed by two techniques, namely tournament selection and fitness proportionate selection. Recombination is achieved by means of the trans-positional crossover (TPX), in which two individuals are chosen at random from the population, and their attributes and methods swapped between the two based on their class position. However, a constraint of the search space is that each design class must contain at least one attribute and one method. Thus positional swapping can only occur where swapping an attribute or method to another class would not leave the class



GA Inspired Process *EP Inspired Process*

Figure 5.1. Flowcharts of GA-inspired and EP-inspired Evolutionary Processes

lacking attributes or methods. Mutation, on the other hand, is achieved by relocating an attribute and a method from one class to another within a single design individual. Recombination rates and mutation rates are held at the population level i.e. the population selects individuals for recombination and mutation at random according to those rates.

The EP-inspired approach also draws upon the genetic operators described in chapter 4, but differs from the GA-inspired approach in algorithmic detail. For example,

potential offspring are first created by mutating all individual designs in the current population. Mutation is performed at two levels: class level mutation and element level mutation. At class level, all attributes or methods of a class in an individual are swapped as a group with another class selected at random. For element level mutation, on the other hand, elements (attributes and methods) in an individual are swapped at random from one class to another, except where this violates the 'at least one attribute and one method' constraint of the representation. To determine which individuals survive in the next generation, the current population (of size N) and the mutated population are combined to make of population of $2N$. Each individual then takes part in a fitness tournament against q individuals selected at random from the combined populations. For each tournament, a 'win' is assigned if the fitness of the individual is superior to its opponent. A score is awarded to each individual based on the number of tournaments won; each individual is then ranked in the population according to their score. Highest ranking individuals progress to the next generation. Drawing on the findings of Bäck (1996) and Eiben and Smith (2003), a value of $q = 10$ has been applied for the EP-inspired selection.[1]

5.2 Methodology

Initial parameter values have been derived from Goldberg (1989), Back (1996) and De Jong (2006) and tuned by empirical trial and error. Optimum settings for evolutionary algorithm parameters are thus as follows:

i. *selection* – either tournament or proportionate for GA-inspired operators, or no selection for the EP-inspired variant;
ii. *crossover and mutation probabilities* – (0.7, 0.03) for GA-inspired operators and (0.0, 1.0) for the EP-inspired variant; and
iii. *offspring creation and replacement strategy* – (100, 100) for GA-inspired operators (i.e. 100 parents generate 100 offspring and only those offspring become parents of the next generation) and (200 TS) replacement for the EP-inspired variant (i.e. 100 parents generate 100 offspring to produce a combined

[1] As both Bäck and Eiben and Smith note, this stochastic selection mechanism allows for individuals of inferior fitness to survive into the next generation. However, as the value of q is increased, this chance becomes more unlikely, until the mechanism becomes deterministic when $q = 2N$.

population of 200; tournament selection is then applied with q = 10 to select 100 individuals for the next generation).

In initial experiments, the performance of fitness functions (i.e. COM, external coupling) and selection mechanisms (tournament, fitness proportionate) are compared for the same example design problem, i.e. the Cinema Booking System.

The overall methodological strategy of the thesis is to focus on computational support for the design within an interactive context. Thus the speed of the evolutionary search is important and is measured. However, exploration is also important and so it is crucial to preserve a range of design solutions so that the final evaluative decision on individual design solutions can be made by the designer. Because of this, populations of designs are seeded at random and average population fitness with standard deviation is recorded as evolutionary search proceeds. With respect to the termination condition, in order to examine the full fitness performance characteristics, search is not halted until the performance of the search has reached a fitness 'plateau'. (Of course, in an interactive evolutionary search, halting of search is at the discretion of the designer). To examine repeatability, each search is run 50 times to provide average population fitness curves together with standard deviation. All software is implemented in the Java programming language and all runs are conducted on a standard desktop PC running the Microsoft Windows operating system.

Because the tournament selection is not deterministic in the EP-inspired variant (where q = 10), it seems likely that the selection pressure for the EP-inspired variant is less than that of the GA-inspired variant. Therefore there is an expectation that the EP-inspired variant is more explorative, while the GA-inspired variant is more exploitative. Results of average population fitness curves are shown in figures 5.2 and 5.3 in the following section.

5.3 Results

All results have been obtained using the Cinema Booking System example design problem domain. Figure 5.2 shows population average fitness curves and standard deviation achieved using the COM cohesion fitness function; figure 5.3 shows population average fitness curves and standard deviation using external coupling as the fitness function. It is evident from both figures that the GA-inspired approach achieves a fitness plateau inside 100 generations using both fitness functions. On the other hand, the EP-inspired approach achieves a fitness plateau inside 1000 generations using the

COM cohesion fitness function, and inside 200 generations using external coupling as the fitness function. It is also evident that for the GA-inspired approach, tournament selection arrives at the fitness plateau before proportionate selection for both fitness functions.

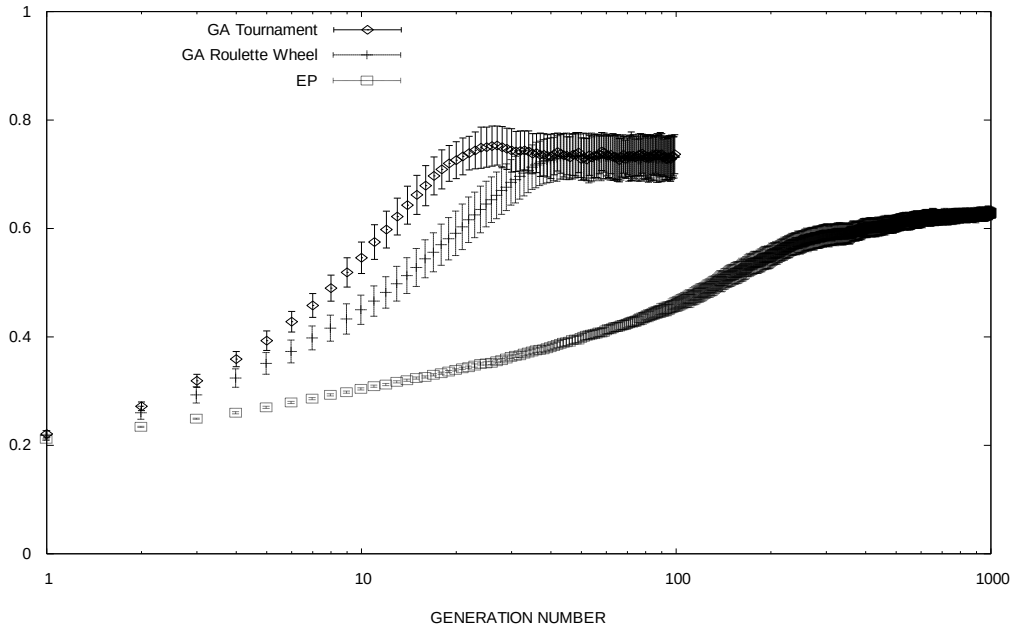


Figure 5.2. Population Average COM Cohesion Fitness Curves

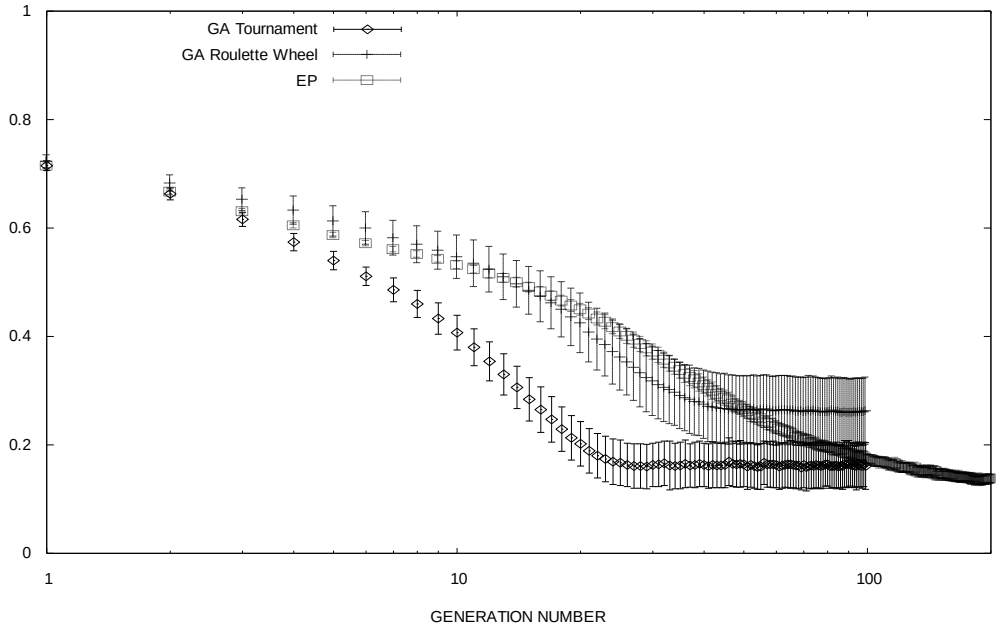


Figure 5.3. Population Average External Coupling Fitness Curves

Table 5.1 reveals the average run time in seconds and standard deviations (in parenthesis) of GA- and EP-inspired approaches for both fitness functions. The number of generations in a run is also shown for comparison. Table 5.2 reveals average population fitness when search has arrived at the fitness plateau for the three approaches trialled. Standard deviation values are provided in parentheses. Manual design cohesion and coupling values taken from the previous chapter are also provided for comparison.

Table 5.1. Search Average Run Times (seconds)

	GA-inspired (Tournament)		GA-inspired (Proportionate)		EP-inspired	
Cohesion	0.854	(0.140)	0.912	(0.163)	3.918	(0.394)
Generations	100		100		1000	
Coupling	0.820	(0.128)	0.964	(0.163)	1.943	(0.207)
Generations	100		100		200	

Table 5.2. Search Average Population Fitness Search Results

	Manual Design	GA-inspired (Tournament)		GA-inspired (Proportionate)		EP-inspired	
Cohesion	0.629	0.744	(0.040)	0.732	(0.043)	0.618	(0.014)
Coupling	0.154	0.163	(0.040)	0.264	(0.063)	0.157	(0.006)

Examples of early lifecycle software design visualisations for the Cinema Booking System (CBS) example design problem after fitness plateaus have been arrived at are shown in figures 5.4, 5.5 and 5.6.

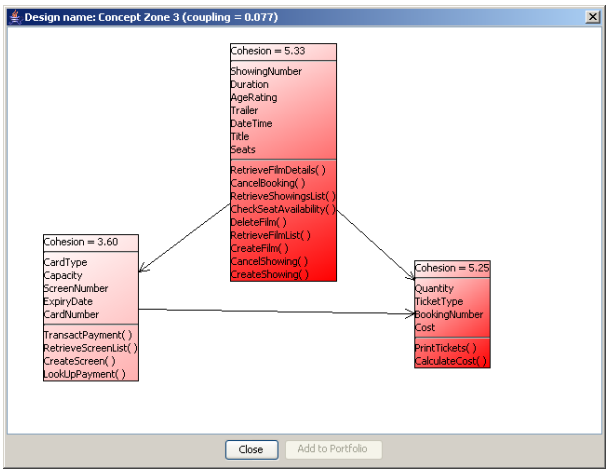


Figure 5.4. Example Software Design Visualisation with Three Classes

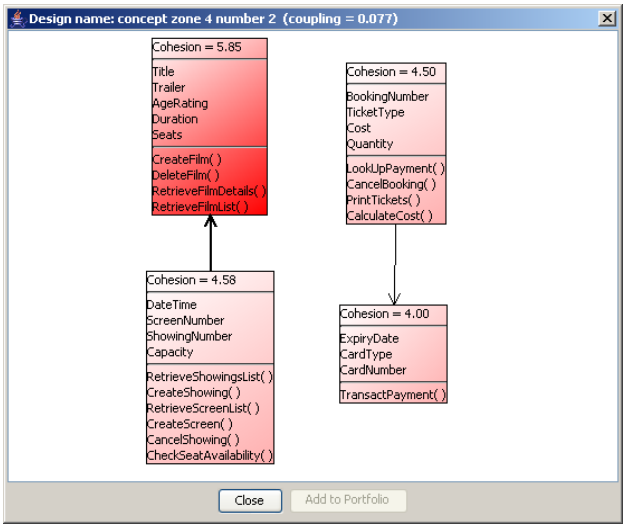


Figure 5.5. Example Software Design Visualisation with Four Classes

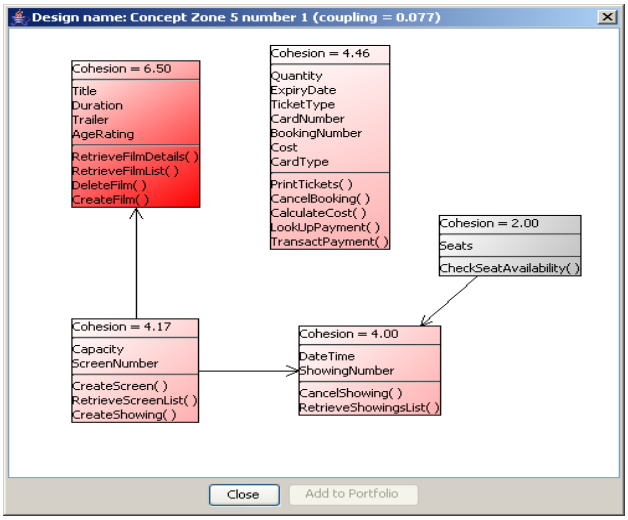


Figure 5.6. Example Software Design Visualisation with Five Classes

5.4 Analysis

With regard to computational speed of execution of evolutionary search, table 5.1 shows that the GA-inspired approach requires approximately one second to run to plateau fitness using average population fitness. The EP-inspired approach requires approximately three seconds using COM cohesion as a fitness measures and two seconds using eternal coupling. This suggests that with respect to computational speed at least, initial parameter tuning achieves sufficient performance with the object-based representation to provide a basis for interactive search.

From table 5.2, it is observed that that the average population fitness values achieved by GA-inspired and EP-inspired evolutionary processes using both cohesion and coupling as fitness functions broadly correspond to the values obtained for the manual design. Figures 5.2 and 5.3 show that for both fitness functions, the population of the GA-inspired approach with tournament selection achieved a fitness plateau first, followed by the GA-inspired approach with fitness proportionate selection, followed by the EP-inspired approach. As was expected, this may be explained by the different balance between exploration and exploitation shown by the two approaches. In the GA-inspired approach, the explorative performance of the TPX crossover operator is restricted by the "one attribute and one method" constraint of the underlying design representation – thus it may be more exploitative in nature. On the other hand, the EP-inspired approach is highly explorative. In the EP approach, selection pressure is less and mutation is significantly more prominent than in GA approach due to the nature of the (200 TS) reproduction. Furthermore, because reproduction is mutation-based, using one parent for each offspring is less restricted by the "one attribute and one method" constraint. It seems likely that for the object-based representation, when compared to crossover, mutation more inherently respects the integrity of the sets of attributes and methods, and so is computationally more straightforward and efficient at preserving diversity in a population of software designs.

With respect to selection mechanisms in the GA-inspired approach, the results would appear to be consistent with earlier findings of Deb (2001) and Eiben and Smith (2003) in that fitness plateaus are achieved faster with tournament selection than fitness proportionate selection.

Regarding the performance of the two fitness functions, it is interesting to compare the results of the two approaches. Indeed, coupling appears to bring about a more consistent performance across the GA- and EP-inspired approaches. An

explanation for this may lie in the nature of the fitness functions. For instance, using the Cohesiveness of Methods (COM) metric, cohesion is expressed as an average of all individual class cohesion values. Thus it is possible for highly cohesive classes to be lost from the population if they co-exist in a software design with classes of inferior cohesion. On the other hand, the external coupling metric is a direct reflection of the design as a whole and does not address individual class fitness. This may explain why in the EP-approach, a fitness plateau is achieved in approximately 180 generations with coupling as opposed to 650 generations with COM. Thus it seems that coupling may be the more tractable and useful fitness function for single-objective local search.

Figures 5.4, 5.5 and 5.6 show UML class designs visualised in a manner suitable for human comprehension. Although printed in this chapter in monochrome, the visualisations are colourful when presented to the designer. Therefore, analysing the results of the initial experiments with the object-based representation and manual parameter tuning, it is observed that:

- speed of computational execution is satisfactory,
- tournament selection out-performs fitness proportionate selection,
- external coupling appears more tractable and useful than COM in local search, and
- class design visualisation is human comprehensible.

Taken in the round, this suggests that the object-based representation and its associated genetic operators can provide an effective basis for evolutionary search of early lifecycle software designs. Nevertheless, in order for the local search to be the basis of user-centred, interactive evolutionary search, it is crucial that search be capable of exploration and exploitation of different design problems and different scales of software design problems. Furthermore, in the course of an interactive software design episode, it is unreasonable to expect the software designer to "tune" control parameters empirically. In short, a more robust and scalable approach is required. Therefore, to enable such robustness and scalability, it seems logical to build upon:

- the superior performance of tournament selection,
- the more explorative and computationally efficient mutation operator, and
- the more tractable and useful external coupling metric.

Therefore, the following section describes further experimentation to evaluate such an approach with dynamic parameter control for all three example software design problems.

5.5 Dynamic Parameter Control

It has been observed that there are limitations to empirical parameter "tuning". For example, over many years of applying evolutionary algorithms, De Jong (2006) has observed that as each parameter to be tuned has a wide range of possible values resulting in an explosion of parameter value combinations due to the range of all possible interactions. Eiben *et al.* (1999) go further to suggest that as evolutionary algorithms are inherently an intrinsically dynamic adaptive process, *"static parameters go against this spirit and different values of parameters may be optimal for different stages of the evolutionary process"*. Indeed, more recently, Eiben *et al.* (2007) reflect that over the past 20 years, the evolutionary computation community has *"shifted from believing that evolutionary algorithm performance is to a large extent independent from the given problem instance to realizing that it is. In other words, it is acknowledged that EAs should be more or less fine-tuned to specific problems and problems instances. Ideally, it should be the algorithm that performs the necessary problem-specific adjustment".* Within early lifecycle software design, there exist many different and unique instances of software design problems. This reasoning thus suggests that each design problem instance requires its own individual fine-tuning of control parameters.

A comprehensive review of dynamic parameter control in evolutionary algorithms is provided by Meyer-Neiberg and Beyer (2007). In their review, Meyer-Neiberg and Beyer explain that dynamic parameter control is well understood in both evolutionary programming (EP) for graphs (Fogel *et al.* (1995)) and evolutionary strategies (ES) (e.g. Rechenberg (1965), Schwefel (1981)) where real-valued representations predominate. Dynamic parameter control has also been investigated in genetic algorithms that employ binary encoding. For example, Schaffer and Morishima (1987) and Spears and Anand (1991) report the use of self-adaptive crossover operators. In addition, examples of investigations into self-adaptive mutation in binary coded genetic algorithms include Back (1993), Smith and Fogarty (1996), Smith (2001), and Stone and Smith (2002). More recently, Serpell and Smith (2010) report the use self-adaptive mutation for permutation representations using benchmark Travelling Salesman Problems (TSPs), while Kramer (2010) surveys operators and strategy

parameters in evolutionary self-adaptation. Indeed, the number of promising reports on the use of dynamic parameter control suggests that considerable efficiency and robustness may be gained from such approaches. However, the representation of the software design search space proposed in this thesis is neither real valued nor binary encoded but object based. It has been found that research findings of dynamic parameter control to evolutionary search using object-based representations are not readily available in the research literature.

With regard to dynamic parameter control in the Interactive Evolutionary Computing (IEC) research literature, Caleb-Solly and Smith (2005) report promising findings on the incorporation of adaptive mutation based on subjective evaluation in an interactive evolutionary strategy within an automatic surface inspection system for classifying defects in sheet steel. In another report by Wenli (2008), adaptive interactive evolutionary computation has been applied to conceptual engineering design. Wenli implements a prototype plug-in for an existing computer-aided design (CAD) tool to assist the engineering designer in curve fitting for ship hull design, although experimentation is highly limited and further investigations would bolster the credibility of this approach. In a different field, Sannen *et al.* (2008) report a novel image classification framework for real-world images recorded during the CD imprint production process, wherein the image classification framework is able to automatically reconfigure and adapt its feature-driven classifiers and improve its performance based on user interaction during on-line processing mode.

In the field of Search-Based Software Engineering (SBSE) however, there appears to be relative lack of work on IEC and reports of dynamic parameter control in evolutionary search also occur infrequently. Nevertheless, there is one report of dynamic optimization in evolutionary test case generation provided by Xie *et al.* (2005). Xie *et al.* use adaptive parameter control by monitoring the number of individuals in each generation, wherein the "proportion of repetitious individuals" (PRI) is obtained. If the PRI exceeds a threshold, the probability of mutation is increased, and *vice versa*. Thresholds are fixed for an evolutionary run. Using adaptive parameter control for three test problems, the authors claim that the performance of the algorithm can be greatly increased, although RPI thresholds require manual tuning.

The evolutionary process used in investigations of dynamic parameter control with local search is shown in figure 5.7. As in the evolutionary processes of the previous section, after the population is initialised, the fitness of the population of

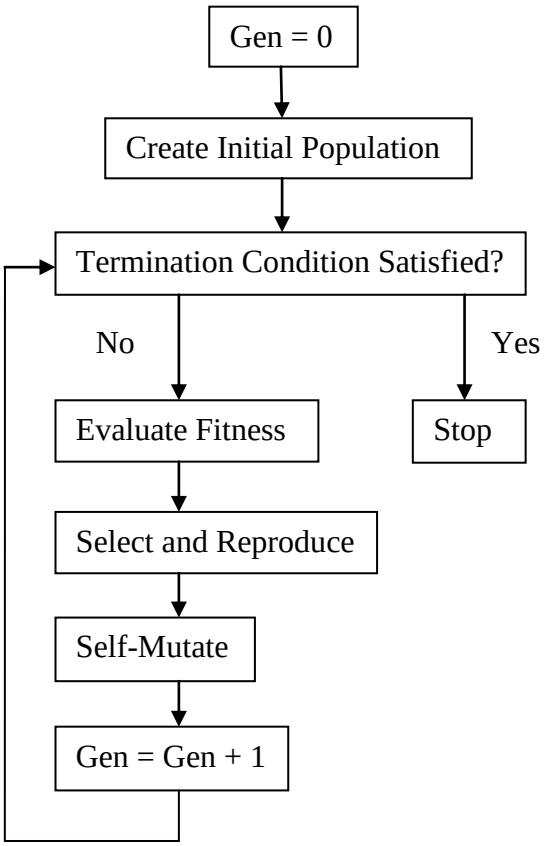


Figure 5.7. Flowchart of Evolutionary Process for Local Search

individual software designs is evaluated. Building on the findings of the previous section, external coupling is used as the fitness function and tournament selection is applied prior to reproduction due to its superior performance. Because of its relative computational complexity and inefficiency, crossover is not used to promote diversity in offspring individuals. Rather, mutation is more computationally straightforward and explorative, and so mutation has been incorporated into the evolutionary local search process. However, certain enhancements have been made to the mutation operation described in the previous section, and are as follows.

Mutation requires the relocation of attributes and/or methods among classes in individual software designs. However, unlike the previous GA and EP-inspired variants, attributes and/or methods are now swapped and moved in rotation. In other words, the mutation operator cycles through the following tactics on each operation:

- *swap attribute:* two classes are chosen at random in a software design individual. From each class, an attribute is chosen at random, and the resulting two attributes are the swapped between the classes.
- *swap_method:* two classes chosen at random in a software design individual. From each class, a method is chosen at random, and the resulting two methods are swapped between the classes.
- *move_attribute:* a single class with more than one attribute is chosen at random in a software design individual. An attribute from this class is removed, and then inserted into another class in the software design individual chosen at random.
- *move_method:* a single class with more than one method is chosen at random in a software design individual. A method from this class is removed, and then inserted into another class in the software design individual chosen at random.

Furthermore, each software design individual encodes its own mutation probability, rather than the probability being held at the population level as is the case with the initial approach reported in the previous section. When an individual acts as a parent and is cloned to produce an offspring, the offspring is mutated using one of the rotating tactics as described above. If the offspring is of a superior fitness to the parent, it is inserted back into the population by means of a 'Delete-Worst' strategy i.e. the individual of worst fitness in the population is replaced by the offspring. If, on the other hand, the mutated offspring is of inferior fitness, it is discarded and the process repeated. The mutation operation is allowed to repeat for up to 50 times. If mutation fails to produce a superior offspring after 50 attempts, mutation is abandoned. The mutation operation can be summarised as follows:

```
WHILE ( attemptCount < MAX_ATTEMPTS )
      Clone an offspring individual from a parent
      Mutate offspring by rotating mutation tactic
      Evaluate mutated offspring
      IF ( offspring is superior to parent )
            Replace individual in population with offspring
               using 'Delete-Worst'
            Exit
      ELSE
            attemptCount = attemptCount + 1
```

In one sense, the mutation operation is not dissimilar to a hill-climbing approach, insofar as offspring are always of superior fitness when compared to their parents. The elite-preserving mutation, when combined with the 'Delete-Worst' replacement strategy, results in an elitist, steady-state local evolutionary algorithm. Such a steady-state approach has been previously reported to be effective (Smith and Fogarty, 1996) and is appropriate for the requirement of an interactive evolutionary search algorithm wherein a robust algorithm must arrive at useful and interesting design solution individuals quickly and efficiently.

5.6 Methodology

Two adaptive and one self-adaptive parameter control mechanisms are compared against a baseline to assess performance within the evolutionary hybrid local search. The adaptive and self adaptive approaches evaluated include

- Annealing,
- 1/5 Success Rule, and
- Self-Adaptive Parameter Control.

For baseline performance, in initial 'tuning' trials of the rotating tactic mutation operation, fixed mutation probabilities of 0.01, 0.02, 0.03, 0.05, 0.10, 0.20 and 0.25 have been trialled, but it has been found that mutation probabilities of over 0.10 do not necessarily produce greater exploration and so greater average population fitness. Thus to provide a valid baseline for comparison with adaptive and self-adaptive approaches, each design solution in the population is assigned a mutation probability between 0.01 and 0.10 at random not only in the initial generation, but also in all subsequent generations.

In the annealing approach to adaptive parameter control, a typical annealing approach inspired by Davis (1987) has been trialled. In this approach, the number of generations drives mutation probability. Mutation probabilities typically start high, and decrease as the number of generations evolves. Thus, each software design solution individual in the population is assigned a mutation probability of 0.10 at the start of each evolutionary run, decreasing in a linear manner to 0.01 at 500 generations.

In the "1/5 Success Rule" proposed by Rechenberg (1973), the success of mutation achieved during evolution is used to drive adjustments of a solution individual's mutation probability. In the context of this experiment, mutation is taken to

be successful where the mutated offspring is of superior fitness when compared to the parent. Rechenberg's heuristic suggests that the ratio of successful mutations to all mutations should be 1/5. Thus if the ratio is less than 1/5, the mutation probability should be increased by a step size to increase local search exploration. On the other hand, if the ratio is greater than 1/5, then the mutation probability should be decreased to focus the local search around the region of the current individual. The rule is applied at periodic intervals, where the mutation probability is adjusted by:

$$p_m' = \begin{cases} p_m \,.\, c & \text{if } p_s > 1/5, \\ p_m \,/\, c & \text{if } p_s < 1/5, \\ p_m & \text{if } p_s = 1/5. \end{cases} \qquad (4.1)$$

where p_m is the probability of mutation, p_s is the frequency of successful mutations, and c is a parameter in the range $0.817 \leq c \leq 1.0$ (Rechenberg, 1973). In this investigation, the success of mutation is examined in every generation in order to enable maximum sensitivity in the face of the three differing example design problems. A mid-range value of 0.9 has been chosen for the constant c. The average population number of attempts to mutate is used to reflect the success or failure of the mutation.

In self-adapting mutation, mutation probability is encoded within the individual and modified prior to evaluation of the solution design individual; effectively the mutation probability is co-evolving with the solution individual. Building loosely on Evolutionary Strategy mutation (e.g. Schwefel, 1995), mutation probability values are mutated thus:

$$p_m' = p_m \,.\, (\, N(\, 0,\, 0.1\,) + 1) \qquad (4.2)$$

where $N(\, 0,\, 0.1\,)$ denotes a random number from a Gaussian distribution with a mean value of 0.0 and a standard deviation of 0.1. This effectively results in a mutation probability multiplicative factor within a range of approximately 0.8 to 1.2 (drawn from a Gaussian distribution around a mean of 1.0). Thus the mutation mechanism is straightforward computationally with a fixed step size.

All local searches use a population size of 100 individuals, and run for 500 generations. Local searches are replicated over 50 runs, after which, average population external coupling fitness, average number of mutation attempts and average population

mutation probability are recorded. All local searches have been implemented in the Java programming language and run on a standard Microsoft Windows office desktop PC. Results obtained are reported in the following section.

5.7 Results

External coupling is used as the fitness function for all dynamic parameter control experiments. For the baseline experiments specifically, each design solution in the population is assigned a mutation probability between 0.01 and 0.10 at random not only in the initial generation, but also in all subsequent generations. The results of baseline average population external coupling fitness (+/- one standard deviation) for the Cinema Booking System (CBS), Graduate Development Program (GDP) and Select Cruises (SC) example design problems are shown in figure 5.8. Figure 5.9 shows the baseline average population number of mutation attempts, while figure 5.10 shows the average mutation probability for the population as the local search evolves. Average population mutation probabilities vary from one generation to another due to noise – but broadly remain at an average of 0.5 as expected. Taking figures 5.9 and 5.10 together, the results reflect the scale of the example design problems. In terms of attributes, methods and uses, CBS is the smallest, GDP is larger and SC is larger still. Because of this, the amount of inherent coupling in each example design problem increases, which is reflected in the average coupling population fitness when the fitness plateau is reached -– 0.175, 0.330 and 0.455 for CBS, GDP and SC respectively. Of the three example design problems, CBS is the quickest to reach a fitness plateau followed by GDP and then SC, again reflecting the scale of the example design problems. The average

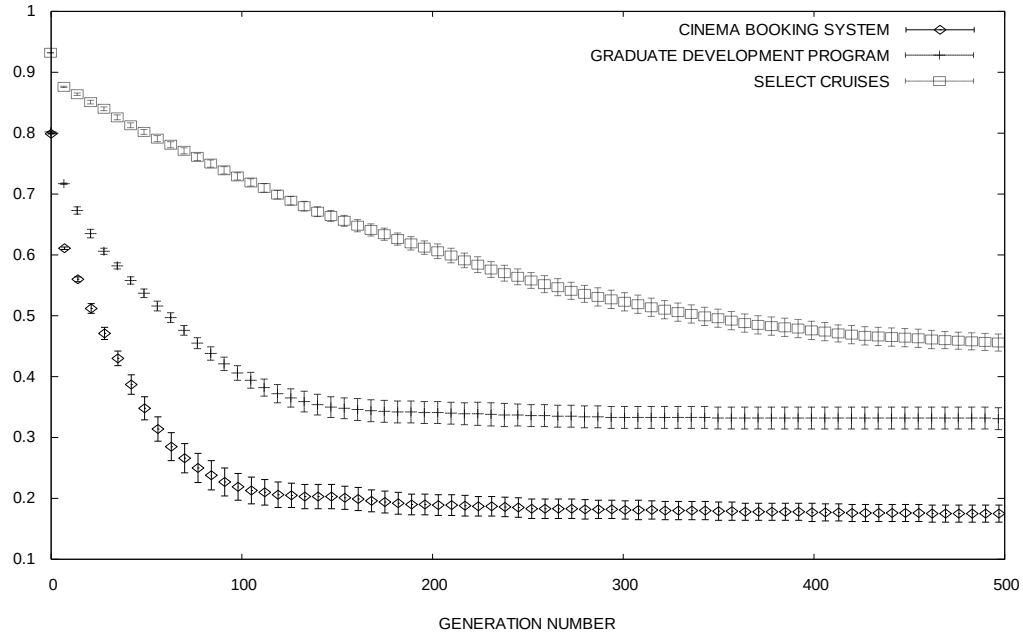


Figure 5.8. Baseline Average Population Fitness

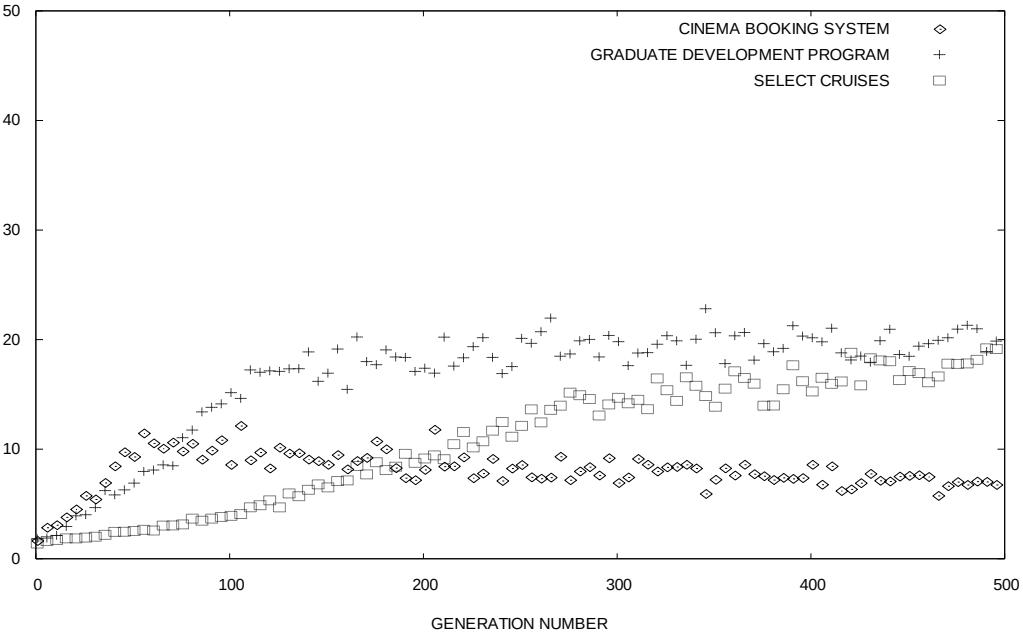


Figure 5.9. Baseline Population Average Number of Mutation Attempts

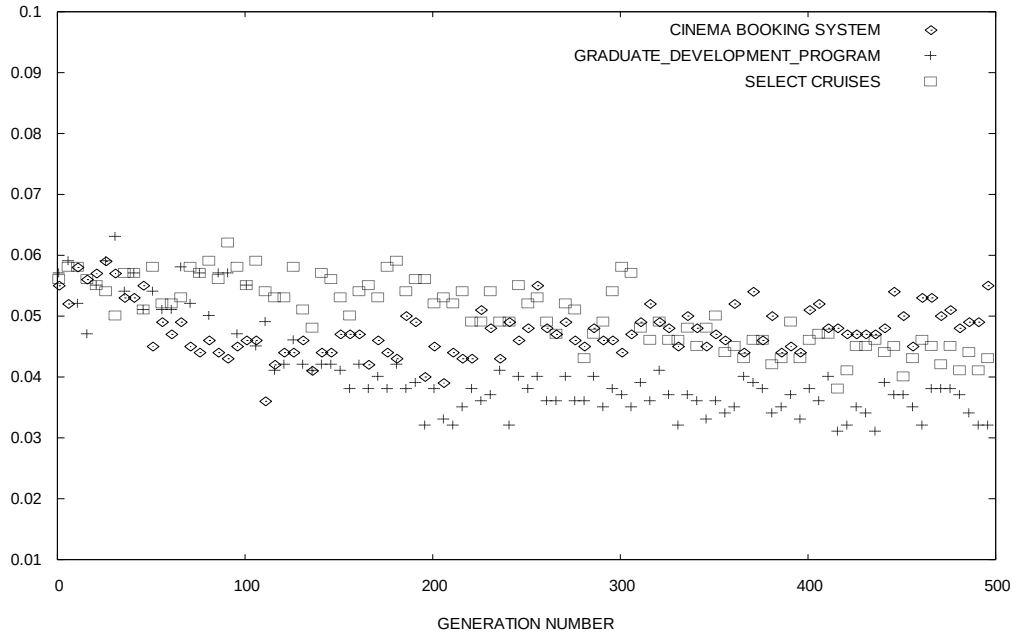


Figure 5.10. Baseline Average Population Mutation Probabilities

execution time for a single run of 500 generations of the baseline local search is 3.917, 14.159 and 12.033 seconds for CBS, GDP and SC respectively.

For experiments with adaptive parameter control using simulated annealing, the number of generations drives mutation probability; mutation probabilities start high and decrease as the number of generations evolve. Thus each software design solution individual is assigned a mutation probability of 0.10 at the start of each evolutionary run, decreasing in a linear manner to 0.01 at 500 generations. Figure 5.11 shows average population coupling results while figure 5.12 shows the population average number of mutation attempts for annealing. Figures 5.11 and 5.12 reveal that results obtained for local search using annealing are similar to those obtained for baseline local searches. Average population fitness values at fitness plateau for CBS, GDP and SC are

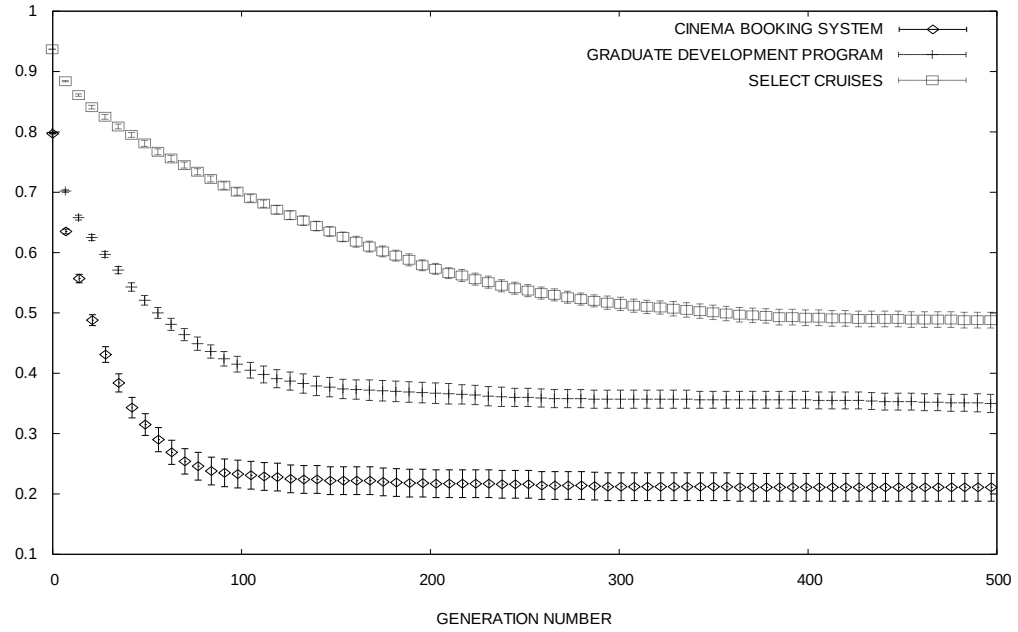


Figure 5.11. Annealing Average Population Fitness

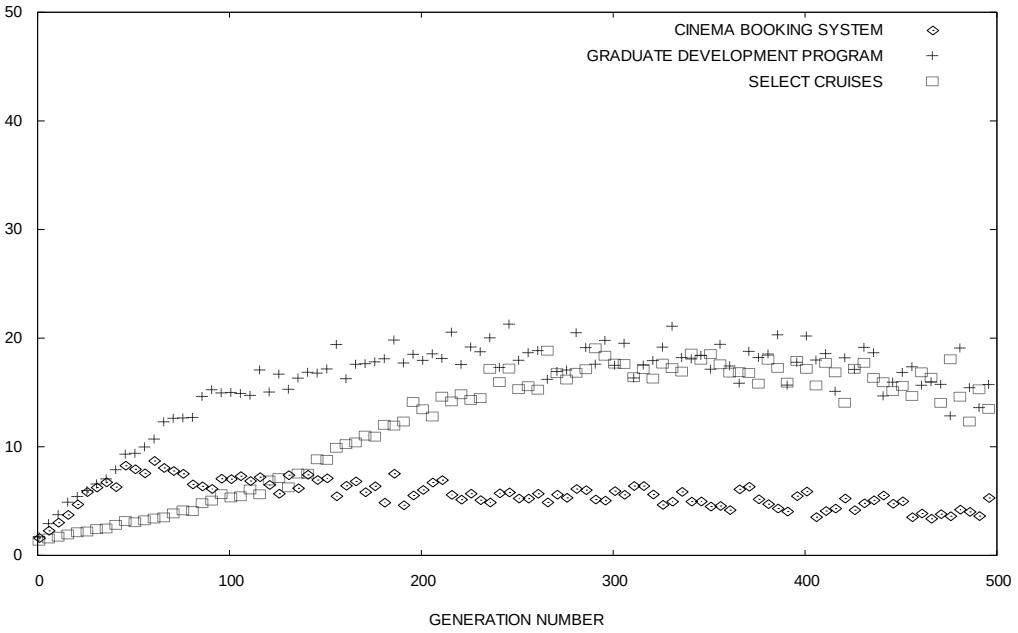


Figure 5.12. Annealing Average Number of Mutation Attempts

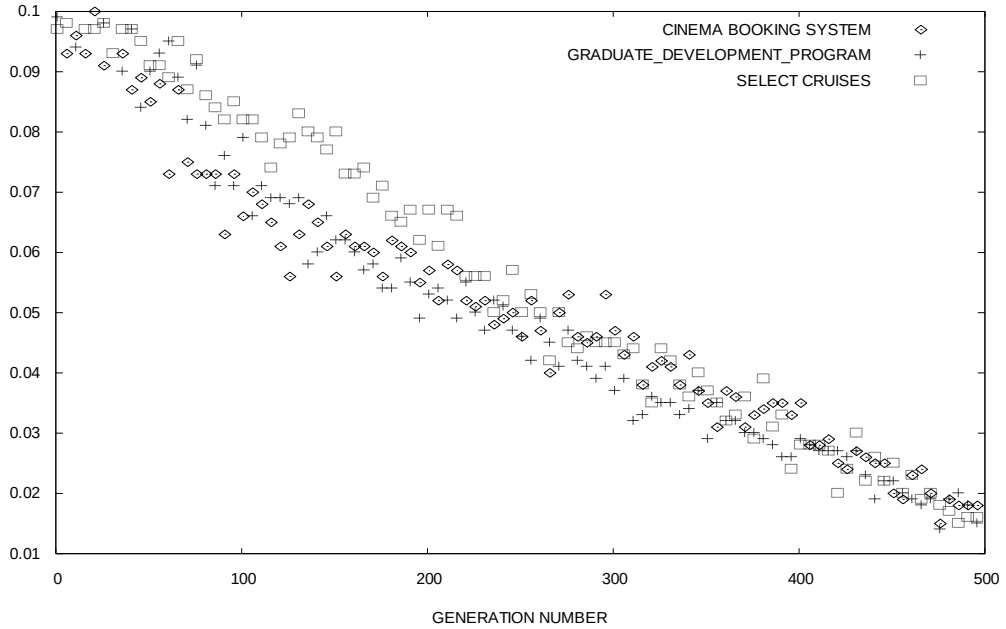


Figure 5.13. Annealing Average Population Mutation Probabilities

0.211, 0.349 and 0.487 respectively. Figure 5.13 shows how average population mutation probabilities decrease from 0.10 at the start of search to 0.01 after 500 generations. Variation in average population mutation probability from one generation to the next is observed, because only solution individuals that have undergone mutation are inserted back into the population.

Results of experiments with adaptive parameter control using Rechenberg's "1 in 5 Success Rule" are shown in figures 5.14, 5.15 and 5.16. Figure 5.14 shows that the local search population using the Rechenberg approach achieves a fitness plateau with average fitness coupling values of 0.184, 0.345 and 0.430 for CBS, GDP and SC respectively. However, the Rechenberg approach achieves fitness plateau in fewer generations than both the baseline and the annealing approach. Using the Rechenberg approach, local search achieves fitness plateau after approximately 60 generations for both CBS and GDP, and 300 generations for SC. With respect to population average number of attempts to mutate for Rechenberg, figure 5.15 reveals similar results to those obtained for the baseline and annealing approaches. However, there are marked differences between mutation probabilities for baseline and annealing, and the Rechenberg approach. Figure 5.16 reveals that for the CBS example, mutation probability climbs steeply to 0.43 after 40 generations, then dropping to 0.10 after approximately 100 generations. By 500 generations, the average population mutation

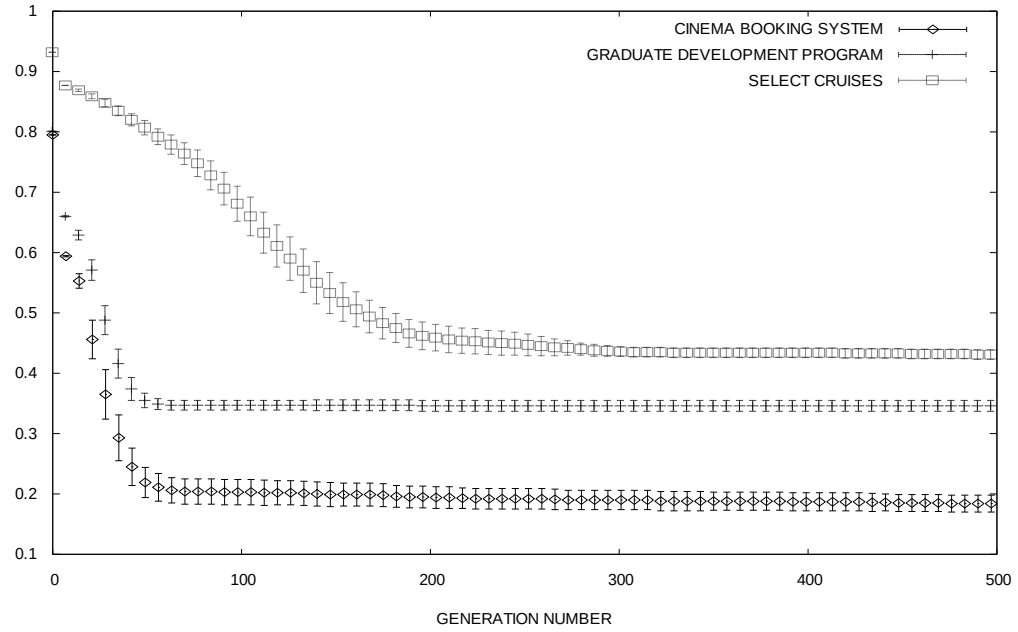


Figure 5.14. Rechenberg Average Population Fitness

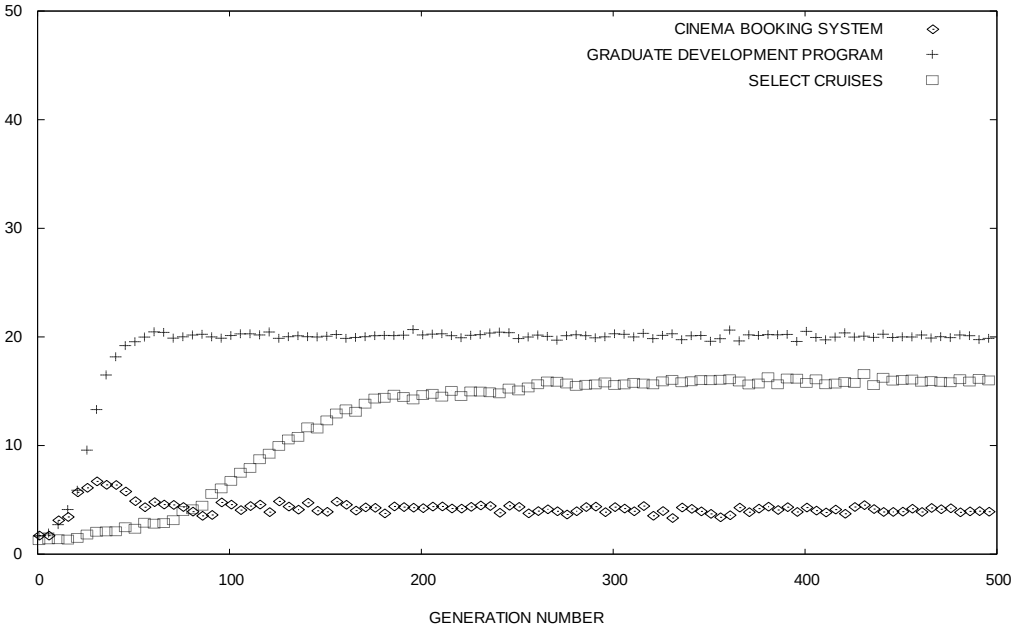


Figure 5.15 Rechenberg Average Number of Mutation Attempts

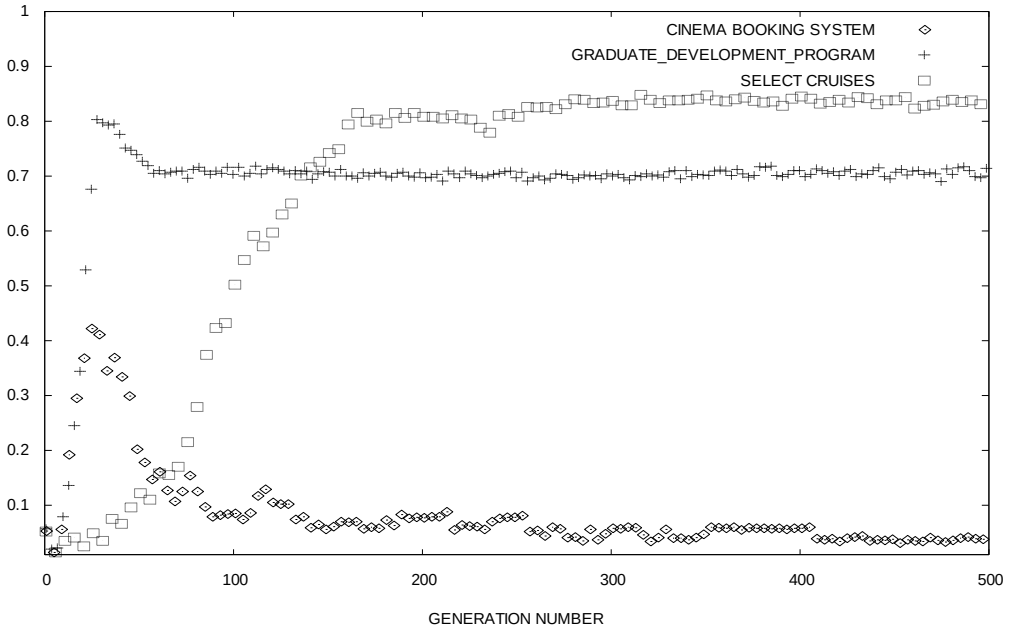


Figure 5.16. Rechenberg Average Population Mutation Probabilities

probability drops to 0.04. This finding is interesting as earlier investigations into empirical parameter tuning appeared to reveal that increasing mutation probability above 0.10 produced no discernable improvement in search performance. However, using the Rechenberg approach to adaptive mutation probability control indicates that dynamic control, peaking at a mutation probability of 0.40 before subsiding, is more effective in achieving population convergence in local search. However, average population mutation probabilities for GDP and SC show dissimilar behaviour. Mutation probabilities for the GDP design example rise quickly to 0.80 after 30 generations, but subside to 0.70 after 60 generations, and stay at that level for the remainder of the local search. Mutation probabilities for the SC design example also rise to 0.80, but less quickly i.e. after 165 generations. There is also no decrease in mutation probability for the remainder of the local search. The high values of mutation probabilities for the three design examples were somewhat unexpected in the light of the baseline and annealing approaches. However, it is conjectured that these findings provide an example of where dynamic parameter control achieves local search efficiencies that were not immediately apparent during manual parameter tuning. The average execution time for a single run of 500 generations of the Rechenberg approach local search is 8.861, 102.602 and 104.636 seconds for CBS, GDP and SC respectively.

Results of experiments with self-adaptive parameter control are shown in figures 5.17, 5.18 and 5.19. Figure 5.17 reveals that the local search population using the self-

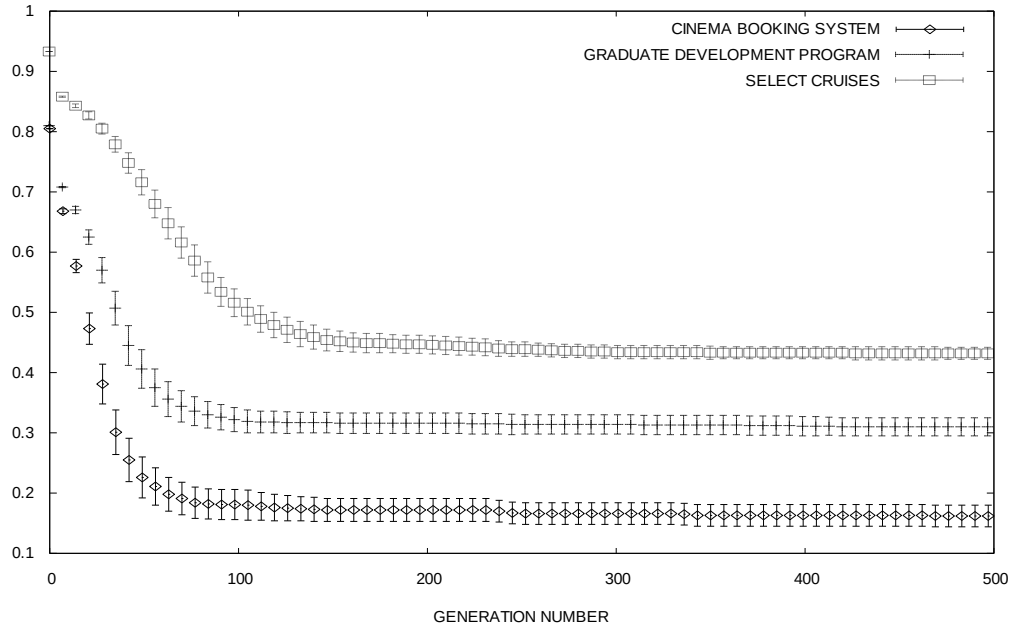


Figure 5.17. Self-adaptive Average Population Fitness

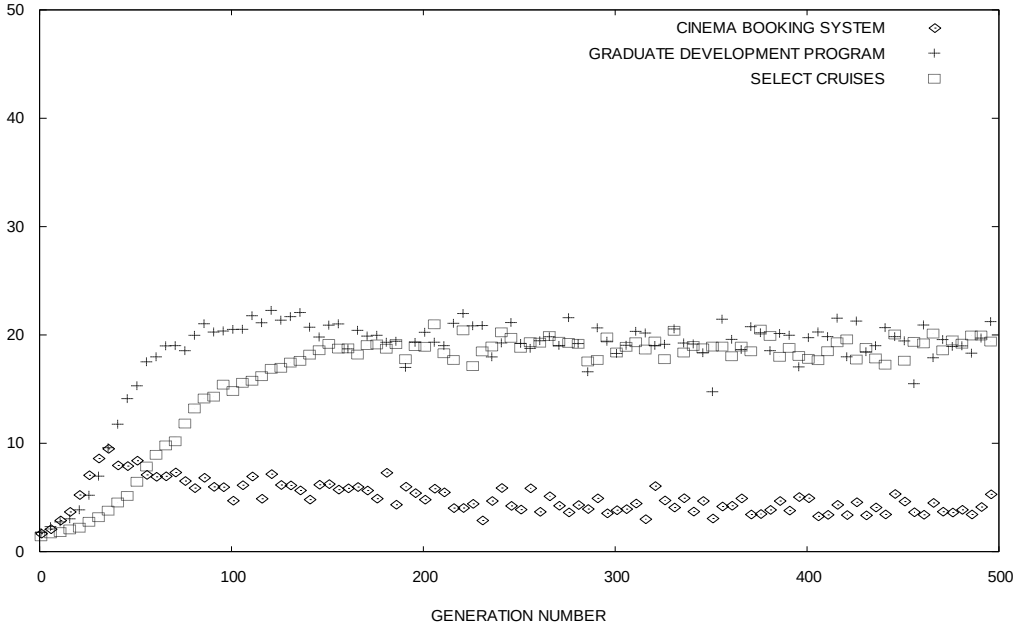


Figure 5.18. Self-adaptive Average Number of Mutation Attempts

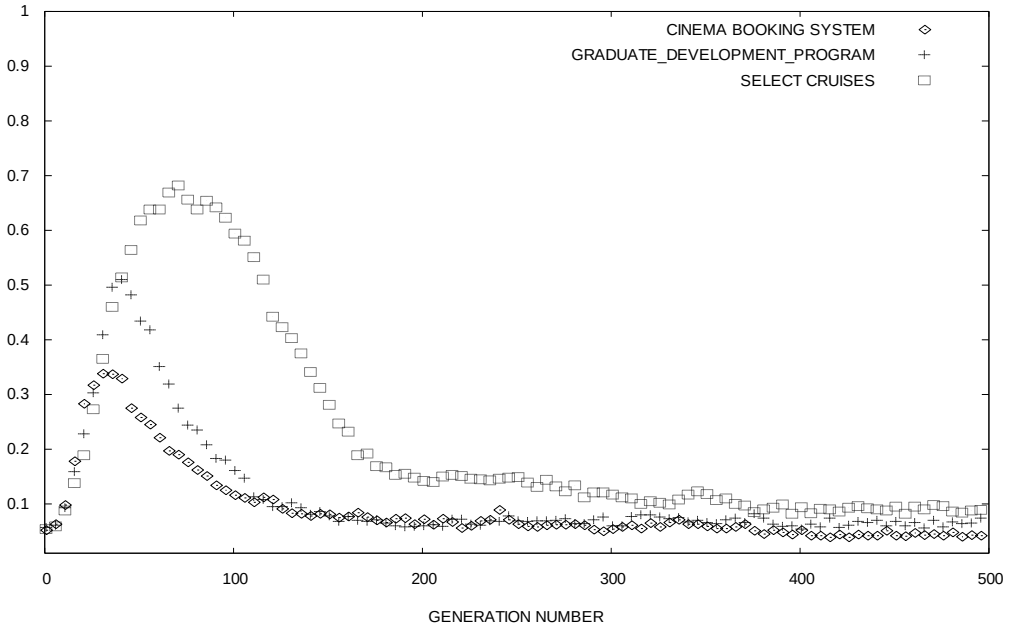


Figure 5.19. Self-adaptive Average Population Mutation Probabilities

adaptive approach converged upon average coupling fitness values of 0.162, 0.309 and 0.431 for CBS, GDP and SC respectively. While the rate of population convergence for example design problems CBS and GDP is similar to that achieved with the Rechenberg approach, the rate of convergence for the self-adaptive approach for the SC example design problem appears to be superior to all previous approaches. Figure 4.18 shows the average population number of attempts to mutate achieved by self-adaption which are similar to the results obtained by Rechenberg, except the number of attempts to mutate for self-adaptation are higher for SC.

Interestingly, figure 5.19 reveals average population mutation probability curves unlike those found for previous dynamic parameter control approaches. Mutation probabilities for CBS rise quickly and peak at 0.35 at 45 generations, thereafter dropping to 0.10 or less by 100 generations. Mutation probabilities for GDP follow a similar pattern, peaking at 0.50 at 50 generations before dropping away to 0.10 or less. This falling mutation probability is dissimilar to Rechenberg, where mutation probability remained high at 0.70. Mutation probabilities for SC also followed the same pattern, although peaking at 0.65 at approximately 75 generations before dropping to 0.10. Again, this is unlike result obtained Rechenberg, where mutation probabilities remain high at 0.80. The results for self-adapting mutation probabilities suggest that self-adaptation performs well in the face of increasing example design problem scale, and this efficient local search is reflected in favourable computational execution times.

The average execution time for a single run of 500 generations of the self-adapting local search is 5.314, 37.791 and 45.351 seconds for CBS, GDP and SC respectively.

5.8 Analysis

When comparing the performance of the dynamic mutation probability approaches, it is important to bear in mind the interactive context of the software design local search. Ideally, dynamic parameter control approach should be robust in the face of a range of example design problems and their differing scale, and be computationally efficient. Thus firstly, mean population coupling fitness for the three example design problems at fitness plateau is compared; values and standard deviations (in parenthesis) are given in Table 5.3 together with coupling values of manual designs for comparison. Tables 5.4, 5.5 and 5.6 show T-test actual confidence levels for comparison of mean values to determine if they are significantly different from each other. Actual confidence levels above 95% are shown in bold.

Table 5.3. Plateau Mean Population Coupling Fitness

	CBS		GDP		SC	
Manual Design	0.154		0.297		0.452	
Baseline	0.175	(0.014)	0.330	(0.018)	0.455	(0.014)
Annealing	0.211	(0.023)	0.349	(0.015)	0.487	(0.013)
Rechenberg	0.184	(0.014)	0.345	(0.009)	0.430	(0.008)
Self Adaptive	0.162	(0.018)	0.309	(0.015)	0.431	(0.010)

Table 5.4. Cinema Booking System T-Test Actual Confidence Levels for Means

	Annealing	Rechenberg	Self-Adaptation
Baseline	**99.99%**	**99.82%**	**99.99%**
Annealing		**99.99%**	**99.99%**
Rechenberg			**99.99%**

Table 5.5. Graduate Development Program T-Test Actual Confidence Levels for Means

	Annealing	Rechenberg	Self-Adaptation
Baseline	**99.99%**	**99.99%**	**99.99%**
Annealing		89.09%	**99.99%**
Rechenberg			**99.99%**

Table 5.6. Select Cruises T-Test Actual Confidence Levels for Means

	Annealing	Rechenberg	Self-Adaptation
Baseline	**99.99%**	**99.99%**	**99.99%**
Annealing		**99.99%**	**99.99%**
Rechenberg			41.79%

The T-Test actual confidence levels shown in tables 5.4, 5.5 and 5.6 reveal that the mean values for plateau population coupling fitness are significantly different from each other for all three design problems, with two exceptions. The only mean values that are not significantly different from each other are Annealing and Rechenberg for Graduate Development Program, and Rechenberg and Self-Adaptation for Select Cruises. Therefore, table 5.3 shows that with respect to plateau mean values of population coupling:

- The performance of Annealing is inferior to other dynamic parameter control approaches. Annealing mean coupling values are inferior to other baseline mean values for all design problems, although the annealing mean coupling value is not significantly different from the mean value for Rechenberg for Graduate Development Program.
- Self-Adaptation out-performs other dynamic parameter control approaches. Self-adaptation mean coupling values are superior to other mean values for Cinema Booking System and Graduate Development Program. However, for Select Cruises, the mean value for self-adaptation is not significantly different to the

mean value for Rechenberg, although both self-adaptation and Rechenberg are superior to annealing and baseline.

However, in an interactive local search situation, the time taken to achieve population convergence is crucial too, and so secondly, initial population fitness curves for the three example design problems are compared in further detail.

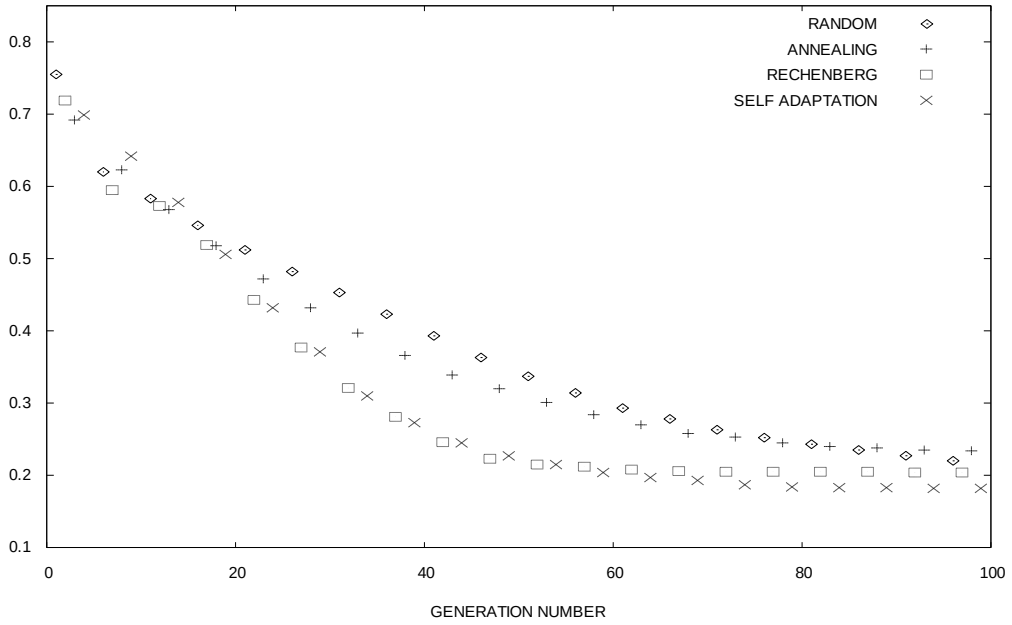


Figure 5.20. Average Population Fitness for CBS

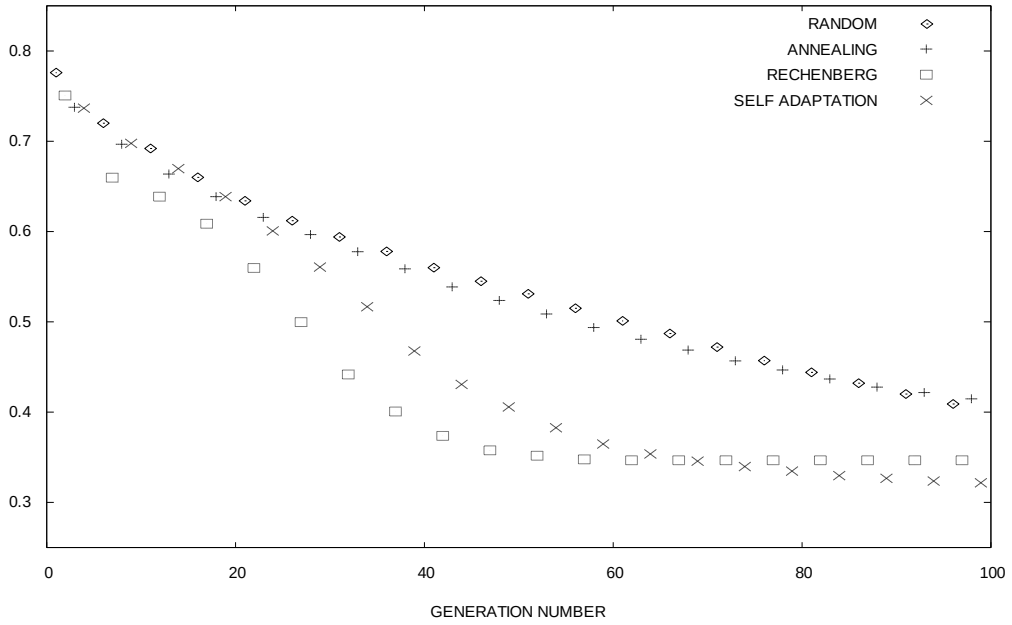


Figure 5.21. Average Population Fitness for GDP

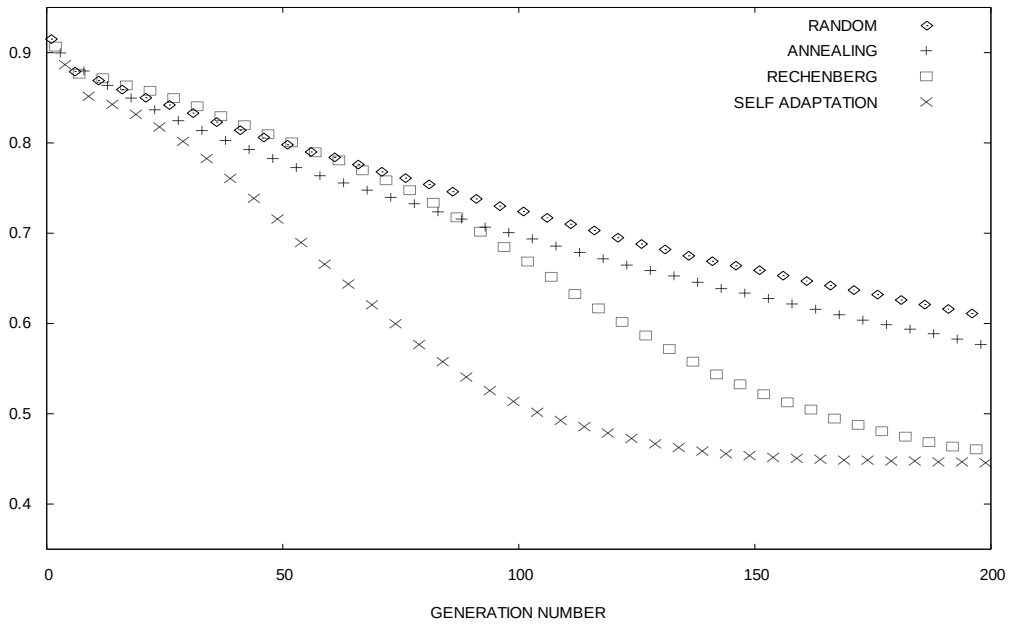


Figure 5.22. Average Population Fitness for SC

Figures 5.20, 5.21 and 5.22 show that for all example design problems, Rechenberg and Self-Adaptation out-perform baseline and Annealing with respect to the rate of achieving fitness plateau. It seems likely that the higher mutation probabilities achieved in Rechenberg and Self-Adaptation promote better exploration in local search, resulting in faster achievement of fitness plateau. However, the effect of design problem scale is also apparent. For the small scale design problem (CBS), Rechenberg and self-adaptation searches achieve fitness plateau at similar rates. In the medium scale design problem (GDP), Rechenberg achieves fitness plateau more quickly, due to steadily high mutation probabilities. However, in the large scale design problem (SC), self-adaptation brings about faster fitness plateau without persistently high mutation probabilities, indicating that self-adaptation copes best with increasing scale of software design problems. Thus overall, self-adaptation provides the fastest fitness plateau and robustness with respect to differing design problems for the object-based representation used in this thesis.

In terms of computational execution times of local search, average execution time for evolutionary runs of 500 generations is shown in Table 5.7. Table 5.7 shows that baseline and annealing mechanisms are fastest to execute, while Rechenberg is clearly the slowest. For the Rechenberg mechanism, it seems likely that high mutations probabilities contribute to longer search execution times. However, in interactive local search, it is not always necessary to allow local search to evolve to plateau fitness;

Table 5.7. Average Execution Time for Run of 500 Generations

	CBS (secs)	GDP (secs)	SC (secs)
Baseline	3.971	14.159	12.033
Annealing	3.049	13.633	16.081
Rechenberg	8.861	102.602	104.636
Self Adapt	5.314	37.791	45.351

indeed, useful and interesting solution individuals may be found in an optimum area of the local search space rather than a single point. Thus the superior early-generation performance of the Rechenberg and Self-Adaptation mechanisms is beneficial. For example, using Self-Adaptation, only 50 generations are required for fitness plateau in the small and medium scale design problems, and 100 generations in the large scale design problem. The execution time required to achieve this is thus estimated to be 0.531, 3.779 and 9.070 seconds for CBS, GDP and SC respectively. While this is slower than the Baseline and Annealing mechanism, the benefits of fast convergence and robustness in the face of differing design problems and their scale suggests that overall, self-adaptation is the superior mutation probability control mechanism for local search of early lifecycle software designs.

5.9 Conclusions

Results of initial empirical parameter tuning indicate that evolutionary local search arrives at design cohesion and coupling values that broadly correspond to values for the manually produced example designs. In terms of fitness functions, external coupling appears more tractable and useful than COM cohesion, while tournament selection out-performs fitness proportionate selection. It is also found that colourful class design visualisation is human comprehensible. Of the two approaches trialled in the initial parameter tuning experiments, the genetic algorithm inspired approach appears more exploitative while the evolutionary programming inspired approach appears to be more explorative.

Building on these findings, results of subsequent experiments reveal that incorporation of dynamic parameter control evolutionary within the local search algorithm is robust and scalable across the three example design problems, with self-adaptation producing the most robust and favourable results overall. Execution speed also appears satisfactory. However, it is interesting to note that the range of mutation probabilities obtained by dynamic parameter control is much wider than that first obtained by empirical parameter tuning. This is consistent with the known limitations of parameter tuning (e.g. Eiben *et al.*, 1999, De Jong, 2006).

Overall, the findings suggest that the object-based representation and its associated genetic operators, with self-adapting mutation probabilities incorporated, can provide an effective and robust basis for evolutionary search of early lifecycle software designs. Nevertheless, when considering interactive evolutionary search as a whole, it is necessary to investigate ways to facilitate collaborative designer / computer interaction. Thus the following chapter describes experiments into multi-objective search of the global search space as a starting point for designer / computer interaction, and also how such multi-objective search might be used to narrow and focus the search to local zones of interest to the designer for later local search.