11

Adapting Genetic Algorithms for Combinatorial Optimization Problems in Dynamic Environments

Abdunnaser Younes, Shawki Areibi*, Paul Calamai and Otman Basir
*University of Waterloo, *University of Guelph*
Canada

1. Introduction

Combinatorial optimization problems (COPs) have a wide range of applications in engineering, operation research, and social sciences. Moreover, as real-time information and communication systems become increasingly available and the processing of real-time data becomes increasingly affordable, new versions of highly dynamic real-world applications are created. In such applications, information on the problem is not completely known a priori, but instead is revealed to the decision maker progressively with time. Consequently, solutions to different instances of a typical dynamic problem have to be found as time proceeds, concurrently with the incoming information.

Given that the overwhelming majority of COPs are NP-hard, the presence of time and the associated uncertainty in their dynamic versions increases their complexity, making their dynamic versions even harder to solve than its static counterpart. However, environmental changes in real life typically do not alter the problem completely but affect only some part of the problem at a time. For example, not all vehicles break down at once, not all pre-made assignments are cancelled, weather changes affect only parts of roads, any other events like sickness of employees and machine breakdown do not happen concurrently. Thus, after an environmental change, there remains some information from the past that can be used for the future. Such problems call for a methodology to track their optimal solutions through time. The required algorithm should not only be capable of tackling combinatorial problems but should also be adaptive to changes in the environment.

Evolutionary Algorithms (EAs) have been successfully applied to most COPs. Moreover, the ability of EAs to sample the search space, their ability to simultaneously manipulate a group of solutions, and their potential for adaptability increase their potential for dynamic problems. However, their tendency to converge prematurely in static problems and their lack of diversity in tracking optima that shift in dynamic environments are deficiencies that need to be addressed.

Although many real world problems can be viewed as dynamic we are interested only in those problems where the decision maker does not have prior knowledge of the complete problem, and hence the problem can not be solved in advance. This article presents strategies to improve the ability of an algorithm to adapt to environmental changes, and more importantly to improve its efficiency at finding quality solutions. The first constructed

Source: Advances in Evolutionary Algorithms, Book edited by: Witold Kosiński, ISBN 978-953-7619-11-4, pp. 468, November 2008, I-Tech Education and Publishing, Vienna, Austria

model controls genetic parameters during static and dynamic phases of the environment; and a second model uses multiple populations to improve the performance of the first model and increases its potential for parallel implementation. Experimental results on dynamic versions of *flexible manufacturing systems* (FMS) and the *travelling salesman problem* (TSP) are presented to demonstrate the effectiveness of these models in improving solution quality with limited increase in computation time.

The remainder of this article is organized as follows: Section 2 defines the dynamic problems of interest, and gives the mathematical formulation of the TSP and FMS problems. Section 3 contains a survey of how dynamic environments are tackled by EAs. Section 4 presents adaptive dynamic solvers that include a diversity controlling EA model and an island-based model. The main goal of Section 5 is to demonstrate that the adaptive models presented in this article can be applied to realistic problems by comparing the developed dynamic solvers on the TSP and FMS benchmarks respectively.

2. Background

Dynamism in real-world problems can be attributed to several factors: Some are natural like wear and weather conditions; some can be related to human behaviour like variation in aptitude of different individuals, inefficiency, absence and sickness; and others are business related, such as the addition of new orders and the cancellation of old ones.

However, the mere existence of a time dimension in a problem does not mean that the problem is dynamic. Problems that can be solved in advance are not dynamic and not considered in this article even though they might be time dependent.

If future demands are either known in advance or predictable with sufficient accuracy, then the whole problem can be solved in advance.

According to Psaraftis (1995), Bianchi (1990), and Branke (2001), the following features can be found in most real-world dynamic problems:

- Time dependency: the problem can change with time in such a way that future instances are not completely known, yet the problem is completely known up to the current moment without any ambiguity about past information.
- A solution that is optimal or near optimal at a certain instance may lose its quality in the next instance, or may even become infeasible.
- The goal of the optimization algorithm is to track the shifting optima through time as closely as possible.
- Solutions cannot be determined in advance but should be computed to the incoming information.
- Solving the problem entails setting up a strategy that specifies how the algorithm should react to environmental changes, e.g. to resolve the problem from scratch at every change or to adapt some parameters of the algorithm to the changes.
- The problem is often associated with advances in information systems and communication technologies which enable the processing of information as soon as received. In fact, many dynamic problems have come to exist as a direct result of advances in communication and real-time systems.

Techniques that work for static problems may therefore not be effective for dynamic problems which require algorithms that make use of old information to find new optima quickly.

2.1 Representative dynamic combinatorial problems

Combinatorial problems typically assume distinct structures (for example vehicle routing versus job shop scheduling). Consequently, benchmark problems for COPs tend to be very specific to the application at hand. The test problems used for dynamic scheduling and sequencing with evolutionary algorithms are typical examples (Bierwirth & Kopfer 1994; Bierwirth et al. 1995; Bierwirth & Mattfeld 1999; Lin et al. 1997; Reeves & Karatza 1993). However, the *travelling salesman problem* has often been considered representative of various combinatorial problems. In this article, we use the dynamic TSP and a dynamic FMS to compare the performance of several dynamic solvers.

2.2 Travelling salesman problem

Although the TSP problem finds applications in science and engineering, its real importance stems from the fact that it is typical of many COPs. Furthermore, it has often been the case that progress on the TSP has led to progress on other COPs. The TSP is modelled to answer the following question: if a travelling salesman wishes to visit exactly once each of a list of cities and then return to the city from which he started his tour, what is the shortest route the travelling salesman should take?

As an easy to describe but a hard to solve problem, the TSP has fascinated many researchers, and some have developed time-dependent variants as dynamic benchmarks. For example, Guntsch et al. (2001) introduced a dynamic TSP where environmental change takes place by exchanging a number of cities from the actual problem with the same number from a spare pool of cities. They use this problem to test an adaptive ant colony algorithm. Eyckelhof and Snoek (2002) tested a new ants system approach on another version of the dynamic problem. In their benchmark, they vary edge length by a constant increment/decrement to imitate the appearance and the removal of traffic jams on roads. Younes et al. (2005) introduced a scheme to generate a dynamic TSP in a more comprehensive way. In their benchmarks, environmental changes take place in the form of variations in the edge length, number of cities, and city-swap changes.

2.2.1 Mathematical formulation

There are many different formulations for the travelling salesman problem. One common formulation is the integer programming formulation, which is given in (Rardin 1998) as follows:

$$min \quad \sum_{i} \sum_{j>i} d_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j<i} x_{ji} + \sum_{j>i} x_{ij} = 2 \quad \text{for all } i$$

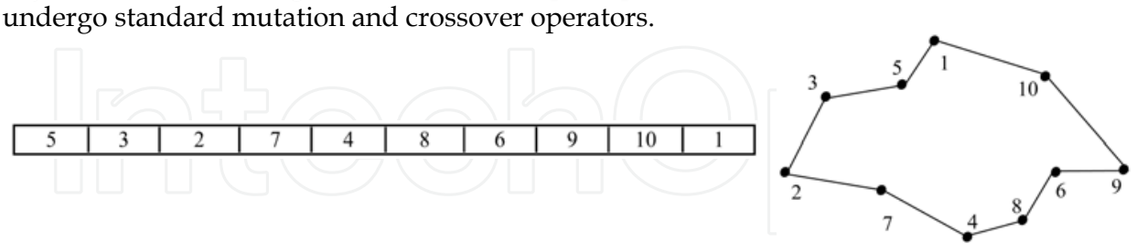
$$\sum_{i \in S} \sum_{j \notin S, j>i} x_{ij} + \sum_{i \notin S} \sum_{j \in S, j>i} x_{ij} \geq 2 \quad \text{for all proper point subsets } S, |S| \geq 3$$

$$x_{ij} = 0 \text{ or } 1 \quad \text{for all } i;\ j > i$$

where x_{ij}= 1 if link (*i; j*) is part of the solution, and $d_{ij}$is the distance from point i to point j. The first set of constraints ensures that each city is visited once, and the second set of constraints ensures that no sub-tours are formed.

2.2.2 Solution representation

In this article a possible TSP solution is represented in a straight forward manner by a chromosome; where values of the genes are the city numbers, and the relative position of the genes represent city order in the tour. An example of a chromosome that represents a 10 city tour is shown in Figure 1. With this simple representation, however, individuals cannot undergo standard mutation and crossover operators.



(a) (b)

Fig. 1. Chromosome representation (a) of a 10 city tour (b) that starts and ends at city 5.

2.3 Flexible manufacturing systems

The large number of combinatorial problems associated with manufacturing optimization (Dimopoulos & Zalzala 2000) is behind the growth in the use of intelligent techniques, such as flexible manufacturing systems (FMS), in the manufacturing field during the last decade. An FMS produces a variety of part types that are flexibly routed through machines instead of the conventional straight assembly-line routing (Chen & Ho 2002). The flexibility associated with this system enables it to cope with unforeseen events such as machine failures, erratic demands, and changes in product mix.

A typical FMS is a production system that consists of a heterogeneous group of numerically controlled machines (machines, robots, and computers) connected through an automated guided vehicle system. Each machine can perform a specific set of operations that may intersect with operation sets of the other machines. Production planning and scheduling is more complicated in an FMS than it is in traditional manufacturing (Wang et al. 2005). One source of additional complexity is associated with machine-operation versatility, since each machine can perform different operations and an operation can be performed on different alternative machines. Another source of complexity is associated with unexpected events, such as machine breakdown, change of demand, or introduction of new products. A fundamental goal that is gaining importance is the ability to handle such unforeseen events. To illustrate the kind of FMS we are focusing on, we give the following example.

2.3.1 Example

A simple flexible manufacturing system consists of three machines, M_1, M_2 and M_3. The three respective sets of operations for these machines are $\{O_1, O_6\}$, $\{O_1, O_2, O_5\}$, and $\{O_4,O_6\}$, where O_i denotes operation i. This system is to be used to process three part types P_1, P_2, and P_3, each of which requires a set of operations, respectively, given as $\{O_1, O_4, O_6\}$, $\{O_1, O_2, O_5, O_6\}$, and $\{O_4, O_6\}$. There are several processing choices for this setting; here are two of them:

Choice (a) For part P_1: $(O_1 \rightarrow M_2; O_4 \rightarrow M_3; O_6 \rightarrow M_3)$; i.e, assign machine M_2 to perform operation O_1, and assign M_3 to process O_4 and O_6. For part P_2: $(O_1 \rightarrow M_1; O_2 \rightarrow M_2; O_5 \rightarrow M_2; O_6 \rightarrow M_1)$. For part P_3: $(O_4 \rightarrow M_3; O_6 \rightarrow M_3)$.

Choice (b) For part P_1: $(O_1 \rightarrow M_2; O_4 \rightarrow M_3; O_6 \rightarrow M_1)$. For part P_2: $(O_1 \rightarrow M_1; O_2 \rightarrow M_2; O_5 \rightarrow M_2; O_6 \rightarrow M_3)$. For part P_3: $(O_4 \rightarrow M_3; O_6 \rightarrow M_1)$.

By comparing both choices, one notices that the first solution tends to minimize the transfer of parts between machines. On the other hand the second solution is biased towards balancing the operations on the machines. However, we need to consider both objectives at the same time, which may not be easy since the objectives are conflicting.

2.3.2 Mathematical formulation

The assignment problem considered in this section is given in Younes et al. (2002) using the following notations:

i,l are machine indices (i,l = 1,2,3,...,n_m);

j is part index (j = 1,2,3,...,n_p);

$\hat{k}_j$ is processing choice for part j (j = 1,2,3,....,n_p);

k_j is the number of processing choices of P_j;

$n_{ij\hat{k}_j}$ is the number of necessary operations required by Pj on M_i in processing choice $\hat{k}_j$,

$1 \leq \hat{k}_j \leq k_j$

$t_{ij\hat{k}_j}$ is the work-load of machine M_i to process part P_j in processing choice $\hat{k}_j$;

$$x_{ji\hat{k}_j} = \begin{cases} 1 & \text{if } P_j \text{ requires } M_i \text{ in processing choice } \hat{k}_j \\ 0 & \text{otherwise;} \end{cases}$$

$$q_{j\hat{k}_j} = \begin{cases} 1 & \text{if processing choice } \hat{k}_j \text{ is selected for part } P_j \\ 0 & \text{otherwise.} \end{cases}$$

Using this notation, the three objective functions of the problem (f_1, f_2, and f_3) are given as follows:

1. Minimization of part transfer (by minimizing the number of machines required to process each part):

$$f_1 = min_{\hat{k}_j} \sum_{i=1}^{n_m} q_{j\hat{k}_j} x_{ji\hat{k}_j}, \forall_j \tag{2}$$

2. Load Balancing by minimizing the cardinality distance (measured in number of operations) between the workload of any pair of machines:

$$f_2 = min_{\hat{k}_j} \sum_{j=1}^{n_p} q_{j\hat{k}_j} \sum_{i=1}^{n_m} \sum_{l=(i+1)}^{n_m} |x_{ji\hat{k}_j} t_{ji\hat{k}_j} - x_{jl\hat{k}_j} t_{jl\hat{k}_j}| \tag{3}$$

3. Minimization of the number of necessary operations required from each machine over the possible processing choices:

$$f_3 = min_{\hat{k}_j} \sum_{i=1}^{n_m} q_{j\hat{k}_j} x_{ji\hat{k}_j} n_{ji\hat{k}_j}, \forall_j \tag{4}$$

An overall multi-objective mathematical model of FMS can then be formulated as follows:

$$Optimize(f_1, f_2, f_3)$$

s.t.

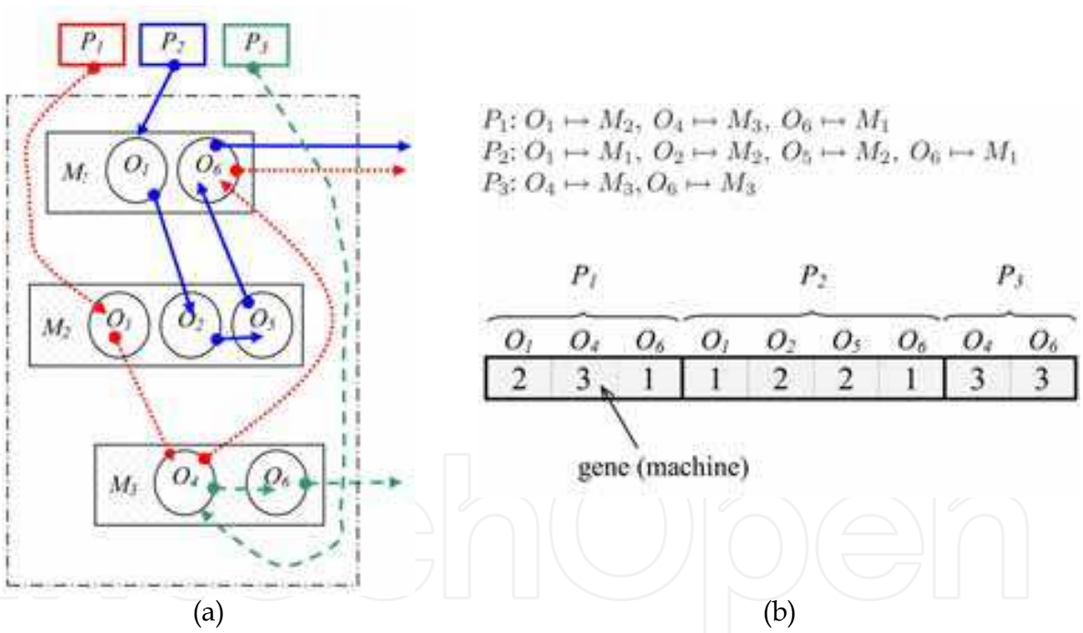
$$\sum_{\hat{k}_j=1}^{k_j} q_{j\hat{k}_j} = 1$$

$$x_{ji\hat{k}_j} \in \{0,1\};\ q_{j\hat{k}_j} \in \{0,1\};\ n_{ji\hat{k}_j} \geq 1;\ t_{ji\hat{k}_j} \geq 0$$

The first set of constraints ensures that only one processing choice can be selected for each part. The complexity and the specifics of the problem require revising several components of the conventional evolutionary algorithm to obtain an effective implementation on the FMS problem. In particular, we need to devise problem-oriented methods for encoding solutions, crossover, fitness assignment, and constraint handling.

2.3.3 Solution representation

An individual solution is represented by a series of operations for all parts involved. Each gene in the chromosome represents a machine type that can possibly process a specific operation. Figure 2 illustrates a chromosome representation of a possible solution to the example given in Section 2.3.1. The advantages of this representation scheme are the simplicity and the capability of undergoing standard operators without producing infeasible solutions (as long as parent solutions are feasible).



(a) (b)

Fig. 2. Chromosome representation. A schematic diagram of the possible choice of part routing in (a) is represented by the chromosome in (b)

3. Techniques for dynamic environments

The limitation on computation time imposed on dynamic problems calls for algorithms that adapt quickly to environmental changes. We discuss some of the techniques that have been used to enhance the performance of the standard *genetic algorithm* (GA) in dynamic environments in the following paragraphs (we direct the interested reader to Jin and Branke (2005) for an extensive survey).

3.1 Restart

The most straightforward approach to increase diversity of a GA search is to restart the algorithm completely by reinitializing the population after each environmental change. However, any information gained in the past search will be discarded with the old population after every environmental change. Thus, if changes in the problem are frequent, this time consuming method will likely produce results of low quality. Furthermore, successive instances in the typical dynamic problem do not differ completely from each other. Hence, some researchers use partial restart: Rather than reinitializing the entire population randomly, a fraction of the new population is seeded with old solutions (Louis and Xu 1996; Louis and Johnson 1997). It should be noted here that for environmental changes that affect the problem constraints, old solutions may become infeasible and hence not be directly reusable. However, repairing infeasible solutions can be an effective approach that leads to suboptimal solutions.

3.2 Adapting genetic parameters

Many researchers have explored the use of adaptive genetic operators in stationary environments (see Eiben et al. (1999) for an extensive survey of parameter control in evolutionary algorithms). In fact, the general view today is that there is no fixed set of parameters that remain optimal throughout the search process even for a static problem.
With variable parameters (self adapting or otherwise) finding some success on static problems, it would be natural to investigate them on dynamic problems.
Cobb (1990) proposed *hyper-mutation* to track optima in continuously-changing environments, by increasing the mutation rate drastically when the quality of the best individuals deteriorates. Grefenstette (1992) proposed *random immigrants* to increase the population diversity by replacing a fraction of the population at every generation. Grefenstette (1999) compared *genetically-controlled mutation* with fixed mutation and hyper-mutation, and reported that genetically controlled mutation performed slightly worse than the hypermutation whereas fixed mutation produced the worst results.

3.3 Memory

When the problem exhibits periodic behaviour, old solutions might be used to bias the search in their vicinity and reduce computational time. Ng & Wong (1995) and Lewis et al. (1998) are among the first who used memory-based approaches in dynamic problems. However, if used at all, memory should be used with care as it may have the negative effect of misleading the GA and preventing it from exploring new promising regions (Branke 1999). This should be expected in dynamic environments where information stored in memory becomes more and more obsolete as time proceeds.

3.4 Multiple population genetic algorithms

The inherent parallel structure of GAs makes them ideal candidates for parallelization. Since the GA modules work on the individuals of the population independently, it is straightforward to parallelize several aspects of a GA including the creation of initial populations, individual evaluation, crossover, and mutation. Communication between the processors will be needed only in the selection module since individuals are selected according to global information distributed among all the processors.
Island genetic algorithms (IGA) (Tanese 1989; Whitley & Starkweather 1990) alleviate the communication load, and lead to better solution quality at the expense of slightly slower

convergence. They have showed a speedup in computation time. Even when an IGA was implemented in a serial manner (i.e., using a single processor), it was faster than the standard GA in reaching the same solutions.

Several multi-population implementations were specifically developed for dynamic environments, for example the *shifting balance genetic algorithm* (SBGA) by Wineberg and Oppacher (2000); the *multinational genetic algorithm* (MGA) by Ursem (2000); and the *self-organizing scouts* (SOS) by Branke et al. (2000).

In SBGA there is a single large *core* population that contains the best found individual, and several small colony populations that keep searching for new optima. The main function of the core population is to track the shifting optimal solution. The colonies update the core population by sending immigrants from time to time.

The SOS approach adopts an opposite approach to SBGA by allocating the task of searching for new optima to the *base* (main) population and the tracking to the *scout* (satellite) populations. The idea in SOS is that once a peak is discovered there is no need to have many individuals around it; a fraction of the base population is sufficient to perform the task of tracking that particular peak over time. By keeping one large base population, SOS behaves more like a standard GA—rather than an IGA—since the main search is allocated to one population. This suggests that the method will be more effective when the environment is dynamic (many different optima arise through time) and hence the use of scouts will be warranted. SOS is more adaptive than SBGA, which basically maintains only one good solution in its base.

MGA uses several populations of comparable sizes, each containing one good individual (the peak of the neighbourhood). MGA is also self-organizing since it structures the population into subpopulations using an interesting procedure called *hill-valley detection*, which causes the immigration of an individual that is not located on the same peak with the rest of its population and the merging of two populations that represent the same peak. The main disadvantage of MGA is the frequent evaluations done for valley detection.

3.5 Adapting search to population diversity

There is a growing trend of using population diversity to guide evolutionary algorithms. Zhu (2003) presents a *diversity-controlling adaptive genetic algorithm* (DCAGA) for the vehicle routing problem. In this model, the population diversity is maintained at pre-defined levels by adapting rates of GA operators to the problem dynamics. However, it may be difficult to set a single value as a target as there is no agreed upon accurate measure for diversity (Burke et al. 2004). Moreover, the contemporary notion that the best set of genetic parameters changes during the run can be used to reason that the value of the best (target) diversity also changes during the run.

Ursem (2002) proposes *diversity-guided evolutionary algorithms* (DGEA) which measures population diversity as the sum of distances to an average point and uses it to alter the search between an exploration phase and an exploitation phase. Riget & Vesterstroem (2002) use a similar approach but with particle swarm optimization. However, the limitation on runtime in dynamic problems may not permit alternate phases.

4. Efficient solvers for dynamic COPs

From the foregoing discussion, techniques based on parameter adaptation and multiple populations seem to be the most promising for tackling dynamic optimization problems.

These techniques, however, were designed for either static problems or dynamic continuous optimization problems, thus none can be used without modification for dynamic COPs. This section introduces two models that are specifically designed for dynamic COPs: the first model uses measured population diversity to control the search process, and the second model extends the first model using multiple populations.

4.1 Adaptive diversity model

The *adaptive diversity model* (ADM) is comparable in many ways to other diversity controlled models. ADM, like DCAGA, controls the genetic parameters. However, unlike DCAGA ADM controls the parameter during environmental changes, and without specifying a single target for diversity. ADM, like DGEA, uses two diversity limits to control the search process, however, it does not reduce the search to the distinct pure exploitation and pure exploration phases, and it does not rely on the continuity of chromosome representation.

In deciding on the best measure for population diversity, it is important to keep in mind that the purpose of measuring diversity is to assess the explorative state of the search process to update the algorithm parameters, rather than precisely determining variety in the population as a goal in itself. For this goal, diversity measures that are based on genotopic distances are convenient since genetic operators act directly on genotype.

Costs of computing diversity of a population of size n can be reduced by a factor of n by using an average point to represent the whole population. However, arithmetic averages can be used only with real-valued representations. Moreover, an arithmetic average does not reflect the convergence point of a population, since evolutionary algorithms are designed to converge around the population-best. Hence, it is more appropriate to measure the population diversity in terms of distances from the population-best rather than distances from an average point. By reserving individual v_n for the population-best, the aggregated genotypic measure (d) of the population can be expressed as

$$d = \sum_{i=1}^{n-1} \frac{dist(v_i, v_n)}{n-1}. \tag{5}$$

Considering the mutation operator for a start, ADM can be described as follows. When an environmental change is detected (at $t = t_m$), the mutation rate is set to an upper limit $\overline{\mu}$. While the environment is static ($t_m \leq t < t_{m+1}$), population diversity $d(t)$ is continually measured and compared to two reference values, an upper limit d_h and a lower limit d_l, and the mutation rate $\mu(t)$ is adjusted using the following scheme:

$$\mu(t) = \begin{cases} \overline{\mu}, & t = t_m \\ Z_l \cdot \big(\overline{\mu} - \mu(t-1)\big) + \mu(t-1), & t \neq t_m,\ d(t) < d_l \\ \mu(t-1) - Z_h \cdot \big(\mu(t-1) - \underline{\mu}\big), & t \neq t_m,\ d(t) > d_h \\ \mu(t-1), & t \neq t_m,\ d_l \leq d(t) \leq d_h \end{cases} \tag{6}$$

where $Z_l = min\left\{\frac{d_l - d(t)}{D}, 1\right\}$, $Z_h = min\left\{\frac{d(t) - d_h}{D}, 1\right\}$, $D = d_h - d_l$

The formula for adaptive crossover rate $\hat{A}(t)$ is similar to that of mutation. However, since high selection pressures reduce population diversity the selection probability $s(t)$ is adapted in an opposite manner to that used for mutation in Equation 6, as follows:

$$s(t) = \begin{cases} \underline{s}, & t = t_m \\ s(t-1) - Z_l \cdot \big(s(t-1) - \underline{s}\big), & t \neq t_m, d(t) < d_l \\ Z_h \cdot \big(\overline{s} - s(t-1)\big) + s(t-1), & t \neq t_m, d(t) > d_h \\ s(t-1), & t \neq t_m,\ d_l < d(t) < d_h \end{cases} \tag{7}$$

where $\underline{s}$ and $\overline{s}$ are the lower and the upper limits of selection probability respectively; and Z_l, and Z_h are as given earlier in the mutation formula 6.

Figure 3 illustrates the general principle of the ADM, and how it drives genetic parameters toward exploration or exploiting in response to measured diversity. In this figure, P can be the value of any of the controlled genetic parameters μ, χ or s. P_r corresponds to maximum exploration values; i.e., $\overline{\mu}$, $\overline{\chi}$ or $\underline{s}$, whereas P_t corresponds to maximum exploitation values ($\underline{\mu}$, $\underline{\chi}$, or $\overline{s}$).

The pseudo code for a dynamic solver using ADM can be obtained from Figure 5, by setting the number of islands to one and cancelling the call to PerformMigration().

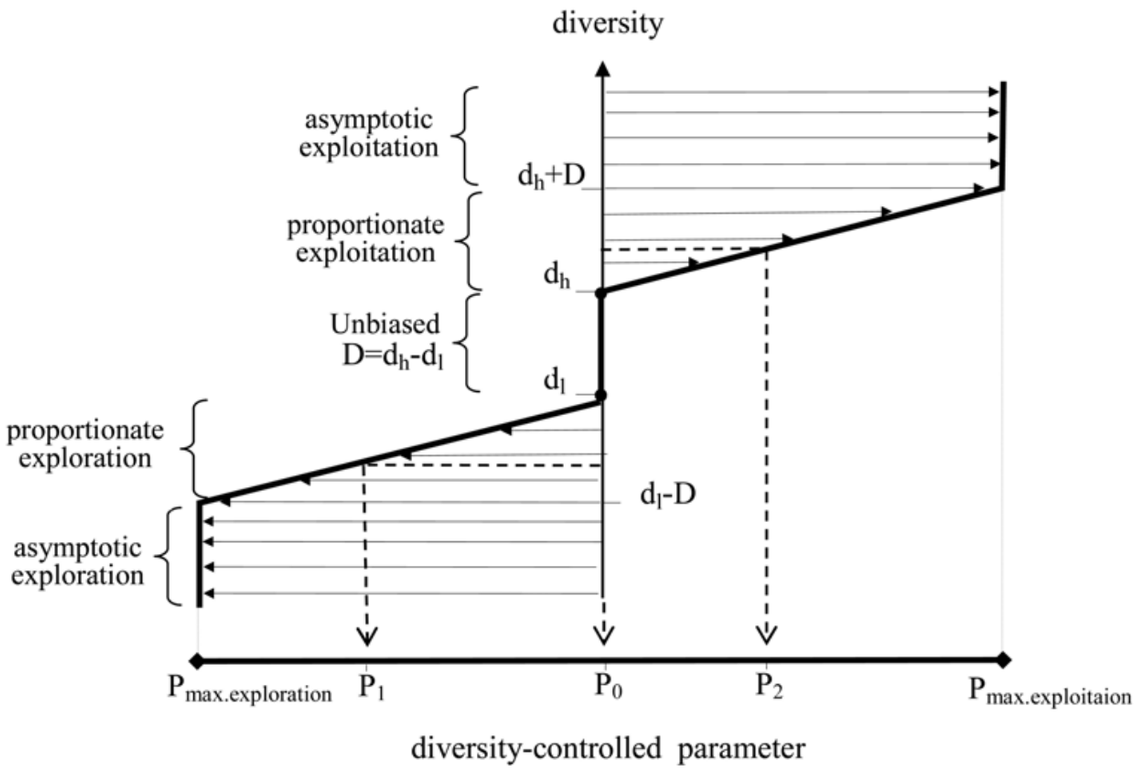


Fig. 3. Diversity range is divided into five regions.

Low diversity maps the genetic parameter into a more explorative value (e.g., P_1) and high diversity maps it into a less explorative value (e.g., P_2). Diversity values between d_l and d_h do not change the current values of the genetic parameters (the parameter is mapped into its original value P_0). The farther the diversity is from the unbiased range, the more change to the genetic parameter. Diversity in the asymptotic regions maps the parameter into one of its extreme values ($P_{\text{max.exploration}}$ or $P_{\text{max.exploitation}}$) .

4.2 Adaptive island model

The *adaptive island model* (AIM) shares many features with other multiple population evolutionary algorithms that have been mentioned previously. However, unlike SBGA and SOS, AIM uses a fixed number of equal-size islands. In addition, no specific island is given the role of base or core island in AIM: the island that contains population-best is considered the current base island. AIM maintains several good solutions at any time, each of which is the center of an island. Accordingly**,** all islands participate in exploring the search space and at the same time exploit good individuals. AIM is more like MGA, but still does not rely on the continuity nature of the variables to guide the search process. As well, AIM uses diversity-controlled genetic operators, in a way similar to that of ADM.

AIM extends the function of ADM to control a number of islands. Thus, two measures of diversity are used to guide the search: an island diversity measure and a population diversity measure. Island diversity is measured as the sum of distances from individuals in the island to the island-best, and population diversity is measured as the sum of the distances from each island best to the best individual in all islands.

Each island is basically a small population of individuals close to each other. It evolves under the control of its own diversity independently from other islands. The best individual in the island is used as an aggregate point for measuring island diversity and as a representative of the island in measuring inter-island diversity (or simply population diversity).

With the islands charged with maintaining population diversity, the algorithm becomes less reliant on the usual (destructive) high rates of mutation. Furthermore, mutation now is required to maintain diversity within individual islands (not within entire population), thus lower rates of mutation are needed. Therefore, mutation rate in AIM, though still diversity dependent, has a lower upper limit.

In order to avoid premature convergence due to islands being isolated from each other, individuals are forced to migrate from one island to another at pre-defined intervals in a ring-like scheme, as illustrated in Figure 4. This scheme helps impart new genetic material to destination islands and increase survival probability of high fitness individuals.

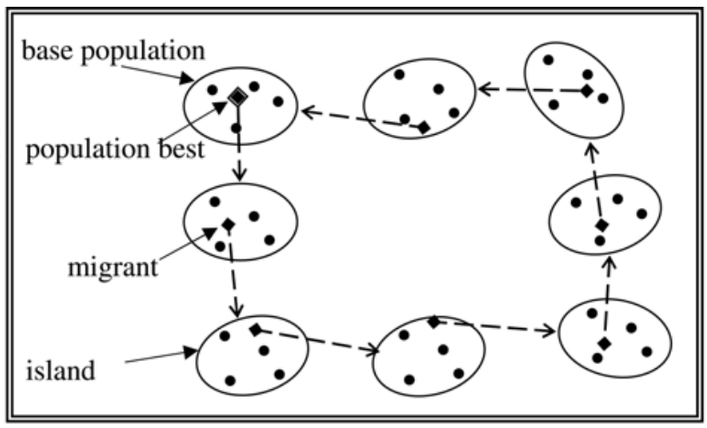


Fig. 4. Ring migration scheme, with the best individuals migrating among islands

On the global level, AIM is required to keep islands in different parts of the search space. This requirement is achieved by measuring inter-island diversity before migration and by mutating duplicate islands. If two islands are found very close to each other, one of them is

considered a duplicate, and consequently its individuals are mutated to cover a different region of the search space. Elite solutions consisting of the best individual from each island are retained throughout the isolation period. During migration, elite solutions are not lost since best individuals are forced to migrate to new islands.
At environmental changes, each island is re-evaluated and its genetic parameters are reset to their respective maximum exploration limits. During quiescent phases of the environment, genetic parameters are changed in response to individual island diversity measures. A pseudo code for AIM is given in Figure 5.

```
Procedure AIM
   i = 0;   // initiate instances counter
   g = 0;   // initiate generations counter
   pop = Generate(isl[1], isl[2], . . . , isl[n_islands]);
   Evaluate( pop );
   repeat
        while quiescent environment
             EvolveIslands( pop , g );
             PerformMigration( pop );
        endwhile
        i = i+1;
        AdaptPopulation( pop , strategy );
   until terminating condition

Procedure EvolveIslands( pop , g )
 for  (k=1 to n_islands)
   repeat
      g = g+1;
      Select( isl[k] );
      Cross( isl[k] );
      Mutate( isl[k] );
      best_of_isl[k] = Evaluate( isl[k] );
      isl_div = MeasureDiversity( isl[k] );
        // use diversity to update genetic parameters
      params[k] = DiversityAdaptParam( params[k] , isl_div );
   until end of isolation period
      pop_best = Best(pop_best,  best_of_isl[k] );
 endfor

Procedure AdaptPopulation( pop , strategy )
 Evaluate(pop);
 Repair(pop);
 ApplyDynamicStrategy(pop);    // set parameters to max explorative limits

Procedure PerformMigration( pop )
 pop_div = MeasureDiversity( isl[1], isl[2], . . . , isl[n_islands] );
 for  (k=1 to n_islands)
      //mutate current island if it is too close to another island
   MutateIsland( isl[k], pop-div);
   Migrate( isl[k] );
 endfor
```

Fig. 5. Pseudo code for AIM. The model can be reduced to ADM by setting the number of islands to one, and cancelling the call to PerformMigration().

5. Empirical study and analysis

The main purpose of this section is to demonstrate the applicability of the adaptive models to realistic problems. First, this section describes the performance measure and the strategies under comparison. Benchmarks and modes of dynamics are then given for each problem together with the results of comparison. Statistical analysis of the significance of the comparisons is given in an appendix at the end of this article.

5.1 Standard strategies and measures of performance

The dynamic test problems are used to compare the proposed techniques against three standard models: a *fixed model* (FM) that uses a GA with fixed operator rates and does not apply any specific measures to tackle dynamism in the problem, a *restart model* (RM) that randomly re-generates the population at each environmental change, and a *random immigrants model* (RIM) that replaces a fraction (10%) of the population with random immigrants (randomly generated individuals) at each environmental change.

Since the problems considered in this article are minimization of cost functions, the related performance measures are directly based on the solution cost rather than on the fitness. First, a mean best of generation (MBG) is defined after G generations of the r_{th} run as:

$$MBG(G,r) = \frac{1}{G}\sum_{g=1}^{G}\left(\frac{\varepsilon_g^r}{\hat{c}_g}\right), \qquad \varepsilon_g^r = min\{e_\theta^r \mid t_g \le \theta < t_{g+1}\} \tag{8}$$

where e_θ^r is the cost associated with the individual evaluated at time step θ and run r, t_g is the time step at which generation g started, and $\hat{c}_g$ is the optimal cost (or the best known cost) to the problem instance at generation g. The algorithm's performance on the benchmark over R runs can then be abstracted as

$$RunsAverageMBG(G,R) = \frac{1}{R}\sum_{r=1}^{R} MBG(G,r). \tag{9}$$

With these definitions, smaller values of the performance measure indicate improved performance. Moreover, since *MBG* is measured relative to the value of the best solutions found during benchmark construction, it will in general exceed unity. Less than unity values, if encountered, indicate superior performance of the corresponding model in that the dynamic solver with limited (time per instance) budget outperforms a static solver with virtually unlimited budget.

5.2 Algorithm parameter settings

In all tested models, the underlying GA is generational with tournament selection in which selection pressure can be altered by changing a selection probability parameter. A population of fifty individuals is used throughout. The population is divided into five islands in the AIM model (i.e., ten individuals per island).

The FM, RM and RIM models use a crossover rate of 0.9 and a selection probability of 1.0. The mutation rate is set to the inverse of the chromosome length (Reeves & Rowe 2002). For the ADM and AIM models, the previous values represent the exploitation limits of their

corresponding operators, with the exploration limits being 1.0 for crossover, 0.9 for selection, and twice the exploitation limit for mutation.
For TSP, edge crossover (Whitley et al. 1991) and pair-wise node swap mutation are used throughout. The mutation operator sweeps down the list of bits in the chromosome, swapping each with a randomly selected bit if a probability test is passed.
For FMS, a simple single-point crossover operator and a standard mutation operator are used throughout (Younes et al. 2002).

5.3 TSP experimentation

5.3.1 TSP benchmark problems

Static problems of sizes comparable to those reported in the literature (Guntsch et al. 2001; Eyckelhof & Snoek 2002) are used in the comparative experiments of this section. These problems are given in the TSP library (Reinelt 1991) as berlin52, kroA100, and pcb442. In this article they are referred to as *be52*, *k100*, and *p442* respectively. Dynamic versions are constructed from these problems in three ways (modes): an edge change mode (ECM), an insert/delete mode (IDM) and a vertex swap mode (VSM).

Edge change mode The ECM mode reflects one of the real-world scenarios, a traffic jam. Here, the distance between the cities is viewed as a time period or cost that may change over time, hence the introduction and the removal of a traffic jam, respectively, can be simulated by the increase or decrease in the distance between cities. The change step of the traffic jam is the increase in the cost of a single edge. The strategy is as follows: If the edge cost is to be increased then that edge should be selected from the best tour. However, if the cost were to be reduced then the selected edge should not be part of the best tour.
The BG starts from one known instance and solves it to find the best or the near best tour. An edge is then selected randomly from the best tour, and its cost is increased by a user defined factor creating a new instance which will likely have a different best tour.

Insert/delete mode The IDM mode reflects the addition and deletion of new assignments (cities). This mode works in a manner similar to the ECM mode. The step of the change in this mode is the addition or the deletion of a single city. This mode generates the most difficult problems to solve dynamically since they require variable chromosome length to reflect the increase or decrease in the number of cities from one instance to the next.

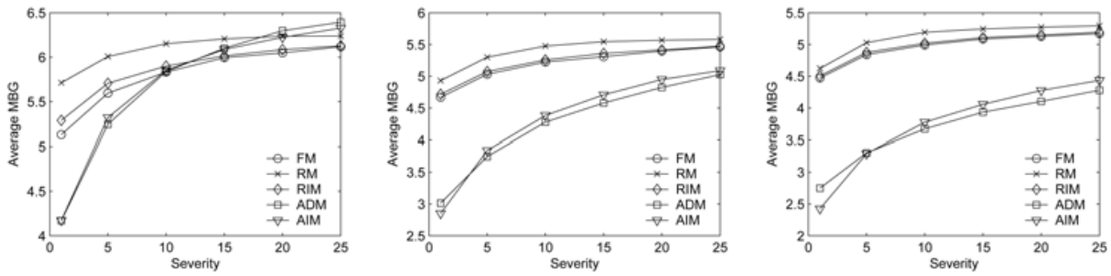
Vertex swap mode The VSM mode is another way to create a dynamic TSP by interchanging city locations. This mode offers a simple, quick and easy way to test and analyze the dynamic algorithm. The locations of two randomly selected cities are interchanged; this does not change the length of the optimal tour but does change the solution (this is analogous to shifting the independent variable(s) of a continuous function by a predetermined amount). The change step (the smallest possible change) in this mode is an interchange of costs between a pair of cities; this can be very large in comparison with the change steps of the previous two modes.

In the experiments conducted, each benchmark problem is created from an initial sequence of 1000 static problems inter-separated by single elementary steps. Depending on the specified severity, a number of intermediate static problems will be skipped to construct one test problem.

Each sequence of static problems is translated into 21 dynamic test problems by combining seven degrees of severity (1, 5, 10, 15, 20, 25 steps per shift, and random) and three periods of change (500, 2500, and 5000 evaluations per shift, which correspond to 10, 50, and 100 generations per shift based on a population of 50 individuals).

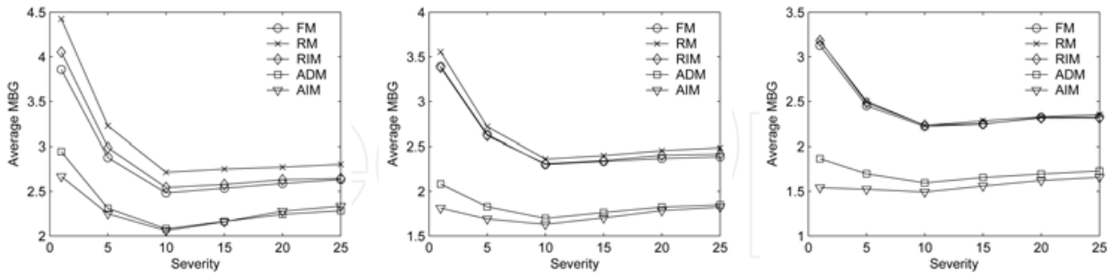
5.3.2 TSP results

Experimental results on the dynamic k100 problem in the VSM mode under three different periods of change are given in Figure 6, where the mean best of generation (averaged over ten runs) is plotted against severity of change. The ADM and AIM models outperform the other models in almost all cases. The other three models give comparable results to each other in general, with differences in solution quality tending to decrease as the severity of change increases. Only when the change severity is 10 steps per shift or more, may the other models give slightly better performance than ADM and AIM. Keep in mind that in this 100 vertex problem, a severity of 10 in the VSM mode amounts to changing (4 × 10) edges; that is, about 40% of the edges in an individual are replaced, which constitutes a substantial amount of change. As we are interested in small environmental changes (which are the norm in practice), we can safely conclude that the experiments attest to the superiority of the ADM and AIM over the other three models in the range of change of interest.



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 6. Comparison of evolutionary models (k100_VSM)

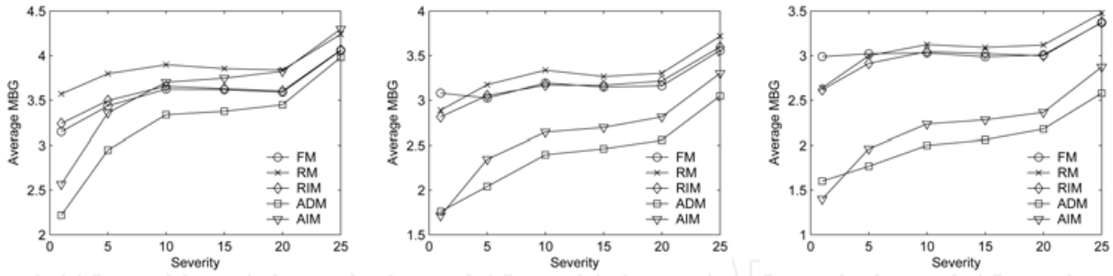


Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 7. Comparison of evolutionary models (k100_ECM)

Running the benchmark generator in either the ECM mode or the IDM mode gives similar results as illustrated in Figure 7 and Figure 8 respectively. It can be seen that ADM and AIM outperform the other models in most considered dynamics.

The RM model produces the worst results in all conducted experiments (even though this model has been modified to retain the best solution in the hope of obtaining better results than those obtainable using a conventional restart).



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 8. Comparison of evolutionary models (k100_IDM)

It is not easy to conclude from previous results the superiority of either model (ADM or AIM), since both give very comparable results in almost all cases. However, when more than one processor can be used, AIM is the best of the two models since it can be easily parallelized by allocating different islands to different processors and consequently reduce computation time drastically.

5.4 FMS experimentation

5.4.1 FMS benchmark problems

Four instances of sizes comparable to those used in the literature (Younes et al. 2002) are used in the comparative experiments of this section.

Three of these instances (20 agents, 200 jobs), (20 agents, 100 jobs) and (10 agents, 100 jobs) were used in Chu and Beasley (1997). The data describing these problems can be found in the *gapd* file in the OR-library (Beasley 1990). In this article they are referred to as *gap1*, *gap2*, and *gap3* respectively. As described in Chen & Ho (2002), agents are considered as machines, jobs are considered as operations, and each part is assumed to consist of five operations. In these instances, a machine is assumed capable of performing all the required operations. However, in general machines may have limited capabilities; that is, each machine can perform a specific set of operations that may or may not overlap with those of the other machines. To enable this feature, a machine-operation incidence matrix is generated for each instance as follows: If the cost of allocating a job to an agent is below a certain level, the corresponding entry in the new incidence matrix is equal to one to indicate that the machine is capable of performing the corresponding operation. Alternatively, if the cost is above this level, the corresponding entry in the incidence matrix is zero to indicate that the job is not applicable to the machine. The final lists that associate parts with operations and machines with operations are used to construct the dynamic problems.

The fourth problem instance is randomly generated. It was specifically designed and used to test FMS systems with overlapping capabilities in Younes et al. (2002). This instance consists of 11 machines, 20 parts, and 9 operations. In this article, it is referred to as *rnd1*.

In terms of the number of part operations (chromosome length) and the number of machines (alleles), the dimensions of these problems are 200×20, 100×20, 100×10,and 62×11 for *gap1*, *gap2*, *gap3*, and *rnd1* respectively.

Dynamic problems are constructed from these instances in three ways (modes): a machine delete mode (MDM), a part add mode (PAM), and a machine swap mode (MSM).

Machine delete mode The MDM mode reflects the real-world scenarios in which a machine suddenly breaks down. The change step of this mode is the deletion of a single machine.

Part add mode The PAM mode reflects the addition and deletion of new assignments (parts). The step of change in this mode is the addition or the deletion of a single part. This mode requires variable representation to reflect the increase or decrease in the number of operations associated with the changing parts.

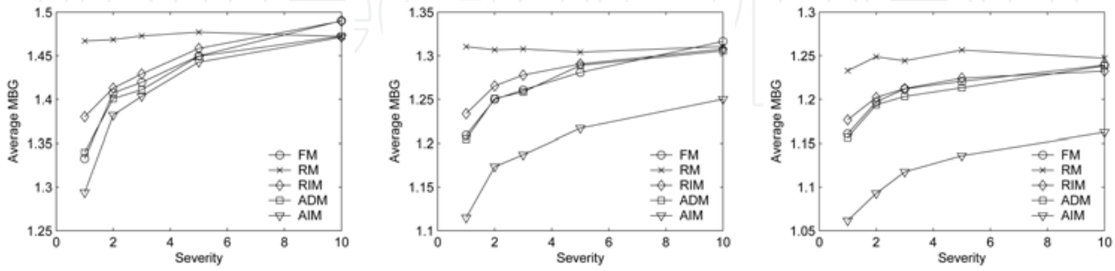
Machine swap mode The MSM mode is a direct application of the mapping-based benchmark generation scheme (Younes et al. 2005). By interchanging machine labels, a dynamic FMS can be generated easily and quickly. The change step in this mode is an interchange of a single pair of machines. As a mapping change scheme, this mode does not require computing a new solution after each change. We only need to swap the machines of the current optimal solution to determine the optimum of the next instance.

In the current experimentation, each benchmark problem is created from an initial sequence of 100 static problems inter-separated by single elementary steps. Depending on the specified severity, a number of intermediate static problems will be skipped to construct one test problem.

Each sequence of static problems is translated into 18 dynamic test problems by combining seven degrees of severity (1, 2, 3, 5, 10 steps per shift, and random) and three periods of change (500, 2500, and 5000 evaluations per shift, which correspond to 10, 50, and 100 generations per shift based on a population of 50 individuals).

5.4.2 FMS results

Experiments were conducted on the rnd1, gap1, gap2, and gap3 problems in the three modes of environmental change. In this section, we focus on the gap1 problem, the largest and presumably the hardest, and on the rnd1 problem, the most distinct. Results of comparisons in the MSM mode are shown in Figure 9, where the average MBG (over ten runs) is plotted against different values of severity. First, we notice that results of the RM model are inferior to those of the other models when the change severity is small. As severity increases, RM results become comparatively better, and at extreme severities RM outperforms the other models. This trend is consistent over different periods of environmental change confirming our notion that restart strategies are best used when the problem changes completely; i.e., when no benefits are expected from re-using old information.



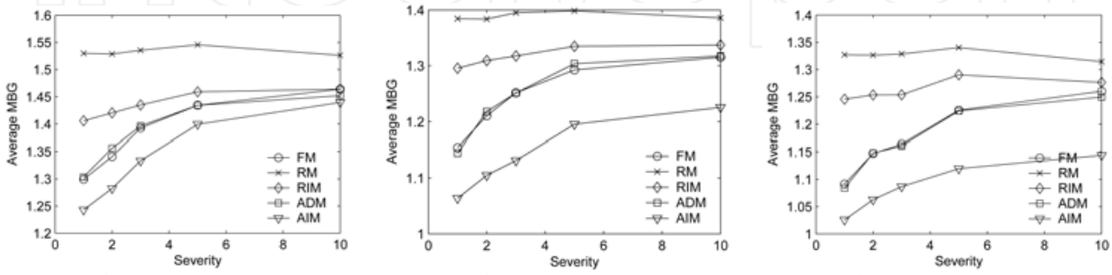
Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 9. Comparison of evolutionary models (rnd1_MSM)

Starting with the ten generation period, we notice that models that reuse old information (all models except for RM) give comparable performance. However, as the period of change

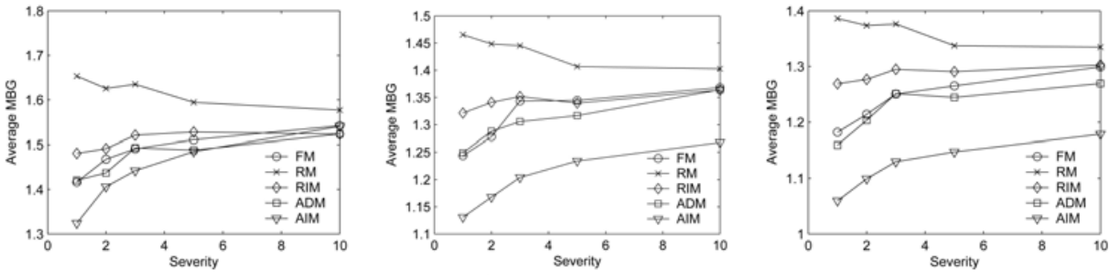
increases, differences between their performance become more apparent. This trend can be explained as follows: when the environmental change is fast, the models do not have sufficient time to converge, and hence they give nearly the same results. When allowed more time, the models start to converge, and those using the best approach to persevere after obsolete convergence produce the best results. The AIM model clearly stands out as the best model.

Comparing the five models on the PAM and MDM modes confirms the results obtained on the MSM mode. The inferiority of the RM model and the superiority of the AIM model persist, as can be seen in Figure 10 and Figure 11.



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 10. Comparison of evolutionary models (rnd1_PAM)

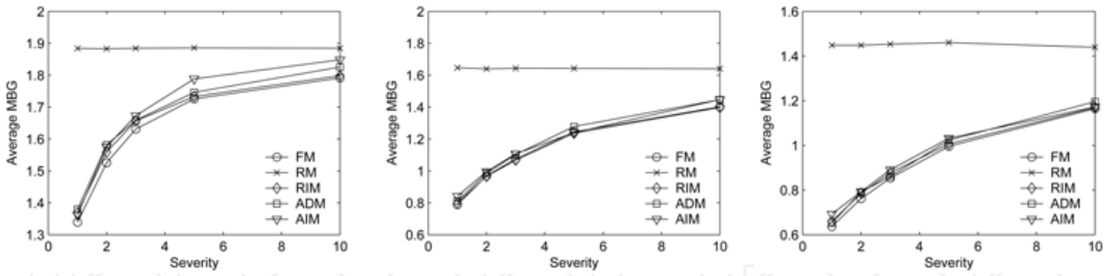


Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 11. Comparison of evolutionary models (rnd1_MDM)

The inferior performance of the RM model is more apparent in the other, large, test problems: the performance of the RM model is consistently poor across the problem dynamics whereas the performance of the other models deteriorates as the severity of environmental change increases. Figure 12 shows the case of gap1 in the MSM mode (other modes show similar behaviour). Comparing the gap1 results to those of rnd1, the apparent deterioration of RM (relative to the other models) in the case of gap1 can be explained by examining change severity. Although values of severity are numerically the same in both cases, relative to problem size they are different, since gap1 is larger than rnd1. In other words, the severity range used in the experiments on gap1 is virtually less than that used on rnd1.

In summary, we can conclude that AIM is the best of the five models, as illustrated clearly in the rnd1 experiments. For other problems in which AIM seems to produce comparable results to those of the other models, we can still opt for the AIM model as it offers the additional advantage of being easy to parallelize, as mentioned in the TSP results section.



Period = 10 generations Period = 50 generations Period = 100 generations

Fig. 12. Comparison of evolutionary models (gap1 MSM)

6. Conclusions and future work

The island based model proves to be effective under different dynamics. Although statistical analysis suggest that these benefits are not significant under some problem dynamics, this model can be more rewarding if several processors are employed. With each island allocated to a different processor, the per processor computational costs are reduced significantly.

The problem of parameter tuning is aggravated with dynamic environments, as a result of the increased problem complexity and the increased number of algorithm parameters; however, by using diversity to control the EA parameters, the models developed in this article had significantly reduced tuning efforts.

There are several ways in which the developed models can be applied and improved:

- The effectiveness of the developed methods on the TSP and FMS problems encourages their application to other problems, such as intelligent transportation systems, engine parameter control, scheduling of airline maintenance, and dynamic network routing.
- Diversity controlled models can use operator-specific diversity measures so that each operator is controlled by its respective diversity measure, i.e., based on algorithmic distance. Future work that is worth exploring involves using adaptive limits of diversity for the models presented in this article.

7. Appendix. Statistical analysis

Statistical t-tests that are used to compare the means of two samples can be used to compare the performance of two algorithms. The typical t-test is performed to build a confidence interval that is used to either accept or reject a null hypothesis that both sample means are equal. In applying this test to compare the performance of two algorithms, the measures of performance are treated as sample means, the required replicates of each sample mean are obtained by performing several independent runs of each algorithm, and the null hypothesis is that there is no significant difference in the performance of both algorithms. However, when more than two samples are compared, the probability of multiple t-tests incorrectly finding a significant difference between a pair of samples increases with the number of comparisons. Analysis of variance (ANOVA) overcomes this problem by testing the samples as a whole for significant differences. Therefore, in this article, ANOVA is performed to test the hypothesis that measures of performance of all the models under considerations are equal. Then, a multiple post ANOVA comparison test, known as Tukey's

test, is carried out to produce 95% confidence intervals for the difference in the mean best of generation of each pair of models.

Statistical results reported here are obtained using a significance level of 5% to construct 95% confidence intervals on the difference in the mean best of generation. Tables in this section summarize the statistical computations of the results reported in Section 5: Table 1, Table 2, and Table 3 are for TSP K100 problem in the three modes of change (respectively, ECM, IDM, and VSM); Table 4 and Table 5 are for the FMS rnd1 and gap1 problems in the MSM mode.

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	0	-1	0	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	1	0	0	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	0	0	-1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	0	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1
RM−ADM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	0	0	-1	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	0	0	0	0	0	0	0	-1	0	-1	0	0	0	1	0	-1	-1	-1	-1	0

Table 1. Multiple comparison test of evolutionary models (k100-VSM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1

Table 2. Multiple comparison test of evolutionary models (k100-ECM)

period	10							100							1000						
severity	1	5	10	15	20	25	r	1	5	10	15	20	25	r	1	5	10	15	20	25	r
FM−RM	-1	-1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
FM−ADM	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
FM−AIM	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1
RM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−ADM	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−AIM	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ADM−AIM	0	-1	0	-1	-1	0	-1	0	-1	0	0	-1	0	0	0	0	0	0	0	0	0

Table 3. Multiple comparison test of evolutionary models (k100-IDM)

Each table covers the combinations of problem dynamics (periods of change and levels of severity of change) described earlier, and an additional column for a random severity) The entries in these tables are interpreted as follows. An entry of 1 signifies that the confidence interval for the difference in performance measures of the corresponding pair consists entirely of positive values, which indicates that the first model is inferior to the second

model. Conversely, an entry of -1 signifies that the confidence interval for the corresponding pair consists entirely of negative values, which indicates that the first model is superior to the second one. An entry of 0 indicates that there is no significant difference between the two models.

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	0	0	-1	-1	-1	-1	0	0	-1	-1	-1	0	0	0	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RM−RIM	1	1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1
RM−ADM	1	1	1	0	0	1	1	1	1	0	0	1	1	1	0	0	0	1
RM−AIM	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1
ADM−AIM	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 4. Multiple comparison test of evolutionary models (rnd1-MSM)

Statistical analysis confirms the arguments made on the graphical comparisons in the previous section. As can be seen in Table 1, 2, and 3, there are significant differences between the performance of the adaptive models (ADM and AIM) and the other three models (FM, RM, and RIM), while there is no significant difference between ADM and AIM. Collectively, the statistical tables confirm the graphical comparisons presented in the previous section. As can be seen in Table 4, and 5, there are significant differences between the performance of the RM model and all others.

period	10						100						1000					
severity	1	2	3	5	10	r	1	2	3	5	10	r	1	2	3	5	10	r
FM−RM	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FM−RIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RM−RIM	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−ADM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RM−AIM	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RIM−ADM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RIM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADM−AIM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5. Multiple comparison test of evolutionary models (gap1-MSM)

8. References

Beasley, J. E. 1990. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society 41(11)*, 1069–1072.

Bianchi, L. 1990. Notes on dynamic vehicle routing - the state of the art. Tech. Rep. idsia 05-01, Italy.

Bierwirth, C. and Kopfer, H. 1994. Dynamic task scheduling with genetic algorithms in manufacturing systems. Tech. rep., Department of Economics, University of Bremen, Germany.

Bierwirth, C., Kopfer, H., Mattfeld, D. C., and Rixen, I. 1995. Genetic algorithm based scheduling in a dynamic manufacturing environment. In *Proc. of IEEE Conference on Evolutionary Computation*. IEEE Press.

Bierwirth, C. and Mattfeld, D. C. 1999. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation 7*, 1, 1–18.

Branke, J. 1999. Memory enhanced evolutionary algorithms for changing optimization problems. In *1999 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ, 1875–1882.

Branke, J. 2001. *Evolutionary Optimization in Dynamic Environments*. Environments. Kluwer Academic Publishers.

Branke, J., Kaussler, T., Schmidt, C., and Schmeck, H. 2000. A multi-population approach to dynamic optimization problaqw2 nh ems. In *Adaptive Computing in Design and Manufacturing 2000*. Springer.

Burke, E. K., Gustafson, S., and Kendall, G. 2004. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation 8*, 1, 47–62.

Chen, J.-H. and Ho, S.-Y. 2002. Multi-objective evolutionary optimization of flexible manufacturing systems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2002)*. Morgan Koffman, New York, New York, 1260–1267.

Chu, P. C. and Beasley, J. E. 1997. A genetic algorithm for the generalised assignment problem. *Computers and Operations Research 24*, 17–23.

Cobb, H. G. 1990. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Tech. Rep. 6760 (NLR Memorandum), Navy Center for Applied Research in Artificial Intelligence,Washington, D.C.

Dimopoulos, C. and Zalzala, A. 2000. Recent developments in evolutionary computation for manufacturing optimization: Problems, solutions, and comparisons. *IEE Transactions on Evolutionary Computation 4*, 2, 93–113.

Eiben, A. E., Hinterding, R., and Michalewicz, Z. 1999. Parameter control in evolutionary algorithms. *IEEE Trans. on Evolutionary Computation 3*, 2, 124–141.

Eyckelhof, C. J. and Snoek, M. 2002. Ant systems for a dynamic tsp. In *ANTS'02: Proceedings of the Third InternationalWorkshop on Ant Algorithms*. Springer Verlag, London, UK, 88–99.

Grefenstette, J. J. 1992. Genetic algorithms for changing environments. In *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)*, R. M¨anner and B. Manderick, Eds. Elsevier, Amsterdam, 137–144.

Grefenstette, J. J. 1999. Evolvability in dynamic fitness landscapes: a genetic algorithm approach. In *1999 Congress on Evolutionary Computation*. IEEE Service Center, Piscataway, NJ, 2031–2038.

Guntsch, M., Middendorf, M., and Schmeck, H. 2001. An ant colony optimization approach to dynamic tsp. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, L. Spector, E. D. Goodman, A.Wu,W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds. Morgan Kaufmann, San Francisco, California, USA, 860–867.

Jin, Y. and Branke, J. 2005. Evolutionary optimization in uncertain environments-a survey. *IEEE Trans. Evolutionary Computation 9*, 3, 303–317.

Lewis, J., Hart, E., and Ritchie, G. 1998. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In *Parallel Problem Solving from Nature –*

PPSN V, A. E. Eiben, T. B¨ack, M. Schoenauer, and H.-P. Schwefel, Eds. Springer, Berlin, 139–148. Lecture Notes in Computer Science 1498.

Lin, S.-C., Goodman, E. D., and Punch, W. F. 1997. A genetic algorithm approach to dynamic job shop scheduling problems. In *Seventh International Conference on Genetic Algorithms*, T. B¨ack, Ed. Morgan Kaufmann, 481–488.

Louis, S. J. and Johnson, J. 1997. Solving similar problems using genetic algorithms and case-based memory. In *Proc. of The Seventh Int. Conf. on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 283–290.

Louis, S. J. and Xu, Z. 1996. Genetic algorithms for open shop scheduling and rescheduling. In *ISCA 11th Int. Conf. on Computers and their Applications*, M. E. Cohen and D. L. Hudson, Eds. 99–102.

Ng, K. P. and Wong, K. C. 1995. A new diploid scheme and dominance change mechanism for non-stationary function optimization. In *Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 159–166.

Psaraftis, H. 1995. Dynamic vehicle routing: Status and prospects. *Annals Operations Research 61*, 143–164.

Rardin, R. 1998. *Optimization In Operation Research*. Prentice-Hall, Inc.

Reeves, C. and Karatza, H. 1993. Dynamic sequencing of a multi-processor system: a genetic algorithm approach. In *Artificial Neural Nets and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. Springer, 491–495.

Reeves, C. R. and Rowe, J. E. 2002. *Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory*. Kluwer Academic Publishers, Norwell, MA, USA.

Reinelt, G. 1991. TSPLIB — a traveling salesman problem library. *ORSA Journal on Computing 3*, 376 – 384.

Riget, J. and Vesterstroem, J. 2002. A diversity-guided particle swarm optimizer – the arpso.

Tanese, R. 1989. Distributed genetic algorithm. In *Proc. of the Third Int. Conf. on Genetic Algorithms*, J. D. Schaffer, Ed. Morgan Kaufmann, San Mateo, CA, 434–439.

Ursem, R. K. 2000. Multinational GAs: Multimodal optimization techniques in dynamic environments. In *Proc. of the Genetic and Evolutionary Computation Conf. (GECCO-00)*, D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, Eds. Morga Kaufmann, San Francisco, CA, 19–26.

Ursem, R. K. 2002. Diversity-guided evolutionary algorithms. In *Proceedings of Paralle Problem Solving from Nature VII (PPSN-2002)*. Springer Verlag, 462–471.

Wang, C., Ghenniwa, H., and Shen, W. 2005. Heuristic scheduling algorithm for flexibl manufacturing systems with partially overlapping machine capabilities. In *Proc. Of 2005 IEEE International Conference on Mechatronics and Automation*, IEEE Press, Niagara Falls, Canada, 1139 1144.

Whitley, D. and Starkweather, T. 1990. Genitor ii.: a distributed geneti algorithm. *J Exp. Theor. Artif. Intell. 2*, 3, 189–214.

Whitley, D., Starkweather, T., and Shaner, D. 1991. The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In *Handbook o Genetic Algorithms*, L. Davis, Ed. Van Nostrand Reinhold, New York, 350–372.

Wineberg, M. and Oppacher, F. 2000. Enhancing the ga's ability to cope with dynamic environments. In *GECCO*. 3–10.

Younes, A., Calamai, P., and Basir, O. 2005. Generalized benchmark generation for dynamic combinatorial problems. In *Genetic and Evolutionary Computation Conference (GECCO2005) workshop program*. ACM Press,Washington, D.C., USA, 25–31.

Younes, A., Ghenniwa, H., and Areibi, S. 2002. An adaptive genetic algorithm for multi objective flexible manufacturing systems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2002)*. Morgan Koffman, New York, New York, 1241–1249.

Zhu, K. Q. 2003. A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In *ICTAI*. 176–183.

Advances in Evolutionary Algorithms

Edited by Xiong Zhihui

ISBN 978-953-7619-11-4
Hard cover, 284 pages
Publisher InTech
Published online 01, November, 2008
Published in print edition November, 2008

With the recent trends towards massive data sets and significant computational power, combined with evolutionary algorithmic advances evolutionary computation is becoming much more relevant to practice. Aim of the book is to present recent improvements, innovative ideas and concepts in a part of a huge EA field.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Abdunnaser Younes, Shawki Areibi, Paul Calamai and Otman Basir (2008). Adapting Genetic Algorithms for Combinatorial Optimization Problems in Dynamic Environments, Advances in Evolutionary Algorithms, Xiong Zhihui (Ed.), ISBN: 978-953-7619-11-4, InTech, Available from: http://www.intechopen.com/books/advances_in_evolutionary_algorithms/adapting_genetic_algorithms_for__combinatorial_optimization_problems_in_dynamic_environments

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen