Single and Multi-Objective Dynamic Optimization: Two Tales from an Evolutionary Perspective

Kalyanmoy Deb
Kanpur Genetic Algorithms Laboratory (KanGAL)
Department of Mechanical Engineering
Indian Institute of Technology Kanpur, PIN 208016, India
deb@iitk.ac.in
http://www.iitk.ac.in/kangal/deb.htm

February 14, 2011
KanGAL Report Number 2011004

Abstract

Many real-world optimization problems involve objectives, constraints, and parameters which constantly change with time. However, to avoid complications, such problems are usually treated as static optimization problems demanding the knowledge of the pattern of change a priori. If the problem is optimized in its totality for the entire duration of application, the procedure can be computationally expensive, involving a large number of variables. Despite some studies on the use of evolutionary algorithms in solving single-objective dynamic optimization problems, there has been a lukewarm interest in solving dynamic multi-objective optimization problems. In this paper, we discuss two different approaches to dynamic optimization for single as well as multi-objective problems. Both methods are discussed and their working principles are illustrated by applying them to different practical optimization problems. The off-line optimization approach in arriving at a knowledge base which can then be used for on-line applications is applicable when the change in the problem is significant. On the other hand, an off-line approach to arrive at an minimal time window for treating the problem in a static manner is more appropriate for problems having a slow change. Further approaches and applications of these two techniques remain as important future work in making on-line optimization task a reality in the coming years.

1 Introduction

A dynamic optimization problem involves objective functions, constraint functions and problem parameters which can change with time. Such problems often arise in real-world problem solving, particularly in optimal control problems or problems requiring an on-line optimization. There are two computational procedures usually followed. In one approach, optimal control laws or rules (more generally a knowledge base) are evolved by solving an off-line optimization problem formed by evaluating a solution on a number of real scenarios of the dynamic problem [9, 10]. This approach is useful in problems which changes frequently and are also computationally expensive for any optimization algorithm to be applied on-line. The other approach is a direct optimization procedure on-line in which an off-line study is suggested for finding a minimal time window within which the problem will be treated as unchanged. In the latter case, the problem is considered stationary for some time period and an optimization algorithm be allowed to find optimal or near-optimal solution(s) within the time span in which the problem remains stationary. Thereafter, a new problem is constructed based on the current problem scenario and a new optimization is

performed for the new time period. Although this procedure is approximate due to the static consideration of the problem during the time for optimization, efforts are made to develop efficient optimization algorithms which can track the optimal solution(s) within a small number of iterations so that the required time period for fixing the problem is small and the approximation error is reduced.

Both the approaches are applicable for single as well as multi-objective optimization problems. In the case of single-objective dynamic optimization problems, the optimal solution change during the optimization procedure and the task of an efficient optimization algorithm would be to track the optimum solution as closely as possible with a minimal computational effort. Although single-objective dynamic optimization has received some attention in the past [2], the dynamic multi-objective optimization is yet to receive a significant attention. When a multi-objective optimization problem changes with time, the task of an dynamic evolutionary multi-objective optimization (EMO) procedure is to find or track the Pareto-optimal front as and when there is a change. Since a front of trade-off solutions change with time, dynamic multi-objective optimization is expected to be harder than dynamic single-objective optimization. A previous study [6] illustrated different possibilities of a change in the optimal front. But since this study, there has been a lukewarm interest on this topic [8, 7].

In the remainder of this paper, we discuss in details the philosophies of both approaches in Section 2. The first approach in which an off-line optimization study is needed to obtain an optimal knowledge base for on-line optimization is described next in Section 3. This section also shows how the procedure can be applied to a dynamic robot navigation problem for a single objective function of minimizing the overall time of travel and satisfying constraints related to avoidance of collision with moving obstacles. Due to the uncertain and imprecision nature of the associated variables, a fuzzy knowledge base is developed by an off-line application of an evolutionary algorithm. Later, the obtained fuzzy rule base is used to navigate on-line in unseen test scenarios. Section 4 then describes the second approach, in which an idea of the minimal time window for considering the problem as a static problem is determined based on an off-line study. The approach is applied to a hydro-thermal power dispatch problem in which power demand is considered as a changing parameter with time. The study shows how the obtained minimal time window allows the approach to be used on-line in a multi-objective version of the problem. The issue of automated decision-making required in the case of dynamic multi-objective optimization is discussed in Section 4.4. Conclusions are drawn in Section 5.

2 Solving Dynamic Optimization Problems

Many search and optimization problems in practice change with time and therefore must be treated as an on-line optimization problems. The change in the problem with time t can be either in its objective functions or in its constraint functions or in its variable boundaries or in any problem parameters or in any combination of above. Such an optimization problem ideally must be solved instantly at every time instant t or whenever there is a change in any of the above functions with t. However, practically speaking, an optimization task requires a finite amount of computational time (τ_{opt}) to arrive at a solution reasonably close to the true optimum. In such problems, there are two time frames which are intertwined: (i) computational time in arriving at a solution (denoted as τ) and (ii) real time in which the problem undergoes a change (denoted as t). While an optimization run is underway in the time frame of τ, the problem gets also changed in the time frame of t. Here, we shall assume equivalence of both time frames and any time spent in one frame affects the same amount in the other time frame.

It now becomes obvious that in an on-line optimization task, as an optimization task is performed (taking a finite time) the optimization problem gets changed and the optimization task is

not solving the same problem with which it started. The relevance and accuracy of the obtained optimum in the current context largely depends on the rate at which the problem changes with time. If the rate of change in the problem is fast compared to the time taken by the optimization algorithm in arriving at the optimal solution, the relevance of the optimal solution of earlier problem to the current context may be questionable. In such a situation, performing an optimization task on-line may not make much sense. On the other hand, if the rate is change is slow, an optimization task can be performed and the obtained optimal solution can still be meaningful. Based on these two scenarios, we suggest two different techniques for a possible on-line optimization task:

1. Develop an optimal rule base off-line and use it for on-line application, and
2. Develop an on-line optimization procedure by considering the problem to be static for a minimal time window.

We discuss each of these techniques in the following sections.

3 Approach 1: Off-line Development of an Optimal Rule Base

This approach is more suitable to problems which change quickly with time or which require a computationally expensive evaluation procedure. In this approach, a number of instantiations of the dynamically changed problem are first collected. An off-line optimization task is then used to find a set of optimal rules (other classifier based approaches can also be adopted here) that would correctly work in the chosen instantiations. It is then believed that since the obtained optimal rule base worked on a number of cases in solving the task optimally, it would also work on new cases on-line. Thus, the obtained optimal rule base can be used to quickly find a reasonable solution to the changing problem. Since new instantiations can be somewhat different from the earlier chosen instantiations, the optimization task of finding the optimal rule base can be repeated in a regular interval during the on-line application process. For this purpose, the new and structurally different instantiations can be stored in an archive and the optimization task can applied after a certain number of new instantiations are stored in the archive. The optimization task can continue in the background without disturbing the on-line application process. We describe one such application through an optimization based approach applied to dynamic robot navigation problem [4].

3.1 Off-line Optimization Approach Applied to a Robot Navigation Problem

Figure 1 shows the suggested off-line optimization based approach. On a set of instantiations, an optimization algorithm is applied to find a knowledge base using rules or by other means. The optimization task would find a set of rules or classifiers which will determine the nature of the outcome based on the variable values at any time instant. In the following, we describe the procedure in the context of an on-line robot navigation problem.

The purpose of the dynamic motion planning (DMP) problem of a robot is to find an obstacle-free path which takes a robot from a point A to a point B with minimum time. There are essentially two parts of the problem:

1. Learn to find *any* obstacle-free path from point A to B, and
2. Learn to choose that obstacle-free path which takes the robot in a minimum possible time.

Both these problems are somewhat similar to the learning phases a child would go through while solving a similar obstacle-avoidance problem. If a child is kept in a similar (albeit hypothetical)

situation (that is, a child has to go from one corner of a room to another by avoiding a few moving objects), the child learns to avoid a incoming obstacle by taking detour from his/her path. It is interesting that while taking the detour he/she never calculates the precise angle of deviation. This process of avoiding an object can be thought as if the child is using a rule of the following sort:

If an object is very near and is approaching, then turn right to the original path.

Because of the imprecise definition of the deviation in this problem, it seems natural to use a fuzzy logic technique here.

The second task of finding an optimal obstacle-free path arises from a simile of solving the same problem by an experienced versus an inexperienced child. An inexperienced child may take avoidance of each obstacle too seriously and deviate by a large angle each time he/she faces an obstacle. This way, this child may lead away from the target and take a long winding distance to reach the target. Whereas, an experienced child may deviate barely from each obstacle, thereby taking the quickest route. If we think about how the experienced child has learned this trick, the answer is through experience of solving many such problems in the past. Previous efforts helped find a set of good rules to do the task efficiently. This is precisely the task of an optimizer which needs to discover the optimal set of rules needed to avoid obstacles and reach the target point in a minimum possible time. This is where the genetic algorithm (GA) is a natural choice.

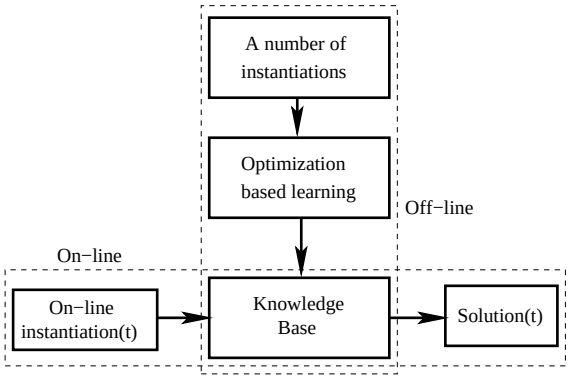

Figure 1: Approach 1 is illustrated.

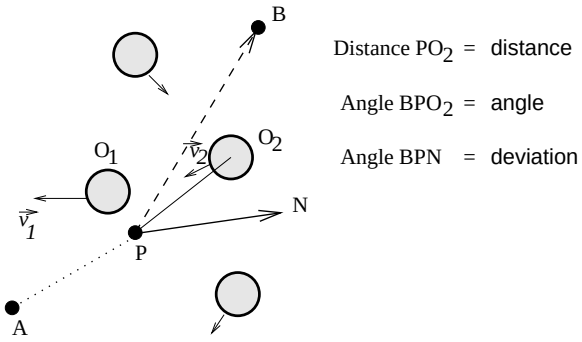

Figure 2: A schematic showing condition and action variables for the robot navigation problem.

In the proposed genetic-fuzzy approach, a GA is used to create the knowledge base comprising of fuzzy rules for navigating a robot off-line. For on-line application, the robot uses its optimal fuzzy rule base to find an obstacle-free path for a given input of parameters depicting the state of moving obstacles and the state of the robot.

3.2 Representation of a Solution in a GA

A solution to the DMP problem is represented by a set of rules which a robot will use to navigate from point A to point B (Figure 2). Each rule has three conditions: distance, angle, and relative velocity. The distance is the distance of the nearest obstacle forward from the robot. Four fuzzy values of distance is chosen: very near (VN), near (N), far (F), and very far (VF). The angle is the relative angle between the path joining the robot and the target point and the path to the nearest obstacle forward. The corresponding fuzzy values are left (L), ahead left (AL), ahead (A), ahead right (AR), and right (R). The relative velocity is the relative velocity vector of the nearest obstacle forward with respect to the robot. In our approach, we eliminate this variable by using a practical incremental procedure. Since, a robot can sense the position and velocity of each

obstacle at any instant of time, the critical obstacle ahead of the robot can always be identified. In such a case (Figure 2), even if an obstacle O_1 is nearer compared to another obstacle O_2, and the relative velocity $\vec{v}_1$ of O_1 directs away from robot's path towards the target point B, whereas the relative velocity $\vec{v}_2$ of O_2 directs towards the robot (Position P), the obstacle O_2 is assumed to be the critical obstacle forward.

The action variable is deviation of the robot from its path towards the target (Figure 2). This variable is considered to have five fuzzy values: L, AL, A, AR, and R. Triangular membership functions are considered for each membership function (Figure 3). Using this rule base, a typical rule will look like the following:

If distance is VN and angle is A, then deviation is AL.

With four choices for distance and five choices for angle, there could be a total of 4×5 or 20 valid rules possible. For each combination of condition variables, a suitable action value (author-defined) is associated, as shown in Table 1.

Table 1: All possible rules are shown.

distance \ angle	L	AL	A	AR	R
VN	A	AR	AL	AL	A
N	A	A	AL	A	A
F	A	A	AR	A	A
VF	A	A	A	A	A

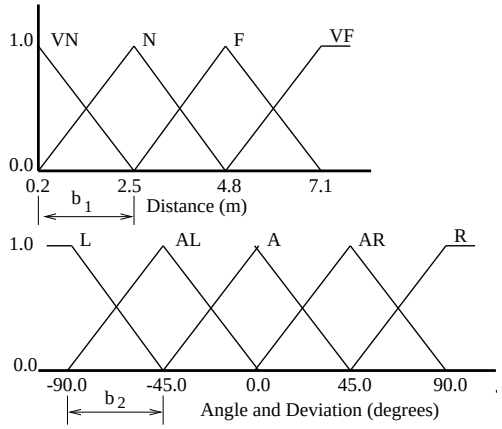

Figure 3: Author-defined membership functions.

The task of GAs is to find which rules (out of 20) should be present in the optimal rule base. We represent the presence of a rule by a `1` and the absence by a `0`. Thus, a complete solution will have a 20-bit length string of `1` and `0`. The value of i-th position along the string marks the presence or absence of the i-th rule in the rule base.

3.3 Evaluating a Solution in the GA

A rule base (represented by a 20-bit binary string) is evaluated by simulating a robot's performance on a number of scenarios and keeping track of the travel time, T. Since a robot may not reach the destination using a lethal rule base, the robot is allowed a maximum travel time. An average of travel times in all scenarios is used as the *fitness* of the solution.

Now, we shall discuss some details which will be necessary to calculate the actual travel time T. As mentioned earlier, the robot's total path is a collection of a number of small straight line paths traveled for a constant time ΔT in each step. To make the matter as practical as possible, we have assumed that the robot starts from zero velocity and accelerates during the first quarter of the time ΔT and then maintains a constant velocity for the next one-half of ΔT and decelerates to zero velocity during the remaining quarter of the total time ΔT. For constant acceleration and deceleration rates (a), the total distance covered during the small time step ΔT is $3a\Delta T^2/16$. At the end of the constant velocity travel, the robot senses the position and velocity of each obstacle

and decides whether to continue moving in the same direction or to deviate from its path. This is achieved by first determining the predicted position of each obstacle, as follows:

$$P_{\text{predicted}} = P_{\text{present}} + (P_{\text{present}} - P_{\text{previous}}). \quad (1)$$

The predicted position is the linearly extrapolated position of an obstacle from its current position P_{present} along the path formed by joining the previous P_{previous} and present position. Thereafter, the nearest obstacle forward is determined based on $P_{\text{predicted}}$ values of all obstacles and fuzzy logic technique is applied to find the obstacle-free direction using the rule base dictated by the corresponding 20-bit string. If the robot has to change its path, its velocity is reduced to zero at the end of the time step; otherwise the robot does not decelerate and continues in the same direction with the same velocity $a\Delta T/4$. It is interesting to note that when the latter case happens (the robot does not change its course) in two consecutive time steps, there is a saving of $\Delta T/4$ second in travel time per such occasion. Overall time of travel (T) is then calculated by summing all intermediate time steps needed for the robot to reach its destination. This approach of robot navigation can be easily incorporated in a real-world scenario[1].

3.4 Results on Robot Navigation Problem

We consider five different techniques:

Technique 1: Author-defined fuzzy-logic controller. In this approach, a fixed set of 20 rules and author-defined membership functions are used. No optimization method is used to find optimal rule base or to find the optimal membership function distributions.

Technique 2: Optimizing membership functions alone. A set of all 20 author-defined rule base is assumed and the membership function distributions of condition and action variables are optimized. The shape of the membership functions is assumed to be triangular. The base of the membership functions are considered as variables. The bases b_1 and b_2 (refer Figure 3) are coded in 10 bit substrings each, thereby making a GA string equal to 20 bits. The base b_1 is decoded in the range (1.0, 4.0) cm and the base b_2 is decoded in the range (25.0, 60.0) degrees. Symmetry is maintained in constructing other membership function distributions. In all simulations here, the membership function distribution for deviation is kept the same as that in angle.

Technique 3: Optimizing rule base alone. The rule base is optimized in this study, while using an author-defined membership functions. Here, the GA string is a 20-bit string (of 1 and 0 denoting presence or absence of rules).

Technique 4: Optimizing membership functions and rule basesimultaneously. In this study, both optimization of finding optimized membership functions and finding an optimized rule base are achieved simultaneously. Here, a GA string is a 40-bit string with first 20 bits denoting the presence or absence of 20 possible rules, next 10 bits are used to represent the base b_1 and the final 10 bits are used to represent the base b_2.

In all runs, we use binary tournament selection (with replacement), the single-point crossover operator with a probability p_c of 0.98 and the bit-wise mutation operator with a probability p_m of 0.02. A maximum number of generations equal to 40 is used. In every case, a population size of 100 is used. In all cases, 10 different author-defined scenarios are used to evaluate a solution.

We now apply all five techniques to eight-obstacle problems (in a grid of 20×24 m^2). The optimized travel distance and time for Techniques 1 to 4 are presented in Table 2. The first three

[1]In all simulations here, $\Delta T = 4$ sec and $a = 1$ m/s^2 are chosen. These values make the velocity of the robot in the middle portion of each time step equal to 1 m/sec.

Table 2: Travel distance D (in meter) and time T (in sec) obtained by five approaches for the eight-obstacle problem.

	Scenario	Technique 1		Technique 2		Technique 3		Technique 4	
		D	T	D	T	D	T	D	T
Training	1	27.203	28.901	26.077	27.769	26.154	27.872	26.154	27.872
	2	26.957	28.943	25.966	27.622	26.026	26.564	26.026	26.546
	3	29.848	36.798	28.623	35.164	26.660	34.547	27.139	35.000
Testing	4	33.465	43.364	26.396	27.907	26.243	27.512	26.243	29.512
	5	32.836	41.781	27.129	33.000	26.543	32.390	27.041	33.000
	6	33.464	43.363	28.001	31.335	27.164	31.000	27.164	31.000

rows in the table show the performance of all approaches on scenarios that were used during the optimization process and the last three rows shows their performance on new test (unseen) scenarios. The table shows that in all cases, Techniques 2, 3 and 4 have performed better than Technique 1 (no optimization). Paths obtained using all four approaches for scenario 4 (unseen) are shown in Figure 4. It is clear that the paths obtained by Techniques 3 and 4 are shorter and quicker than that obtained by Techniques 1 and 2. The optimized rule bases obtained using Techniques 3 and 4 are shown in Tables 3 and 4. The optimized membership functions obtained using

Table 3: Optimized rule base (having nine rules only) obtained using Techniques 3 for eight-obstacle problem.

distance \ angle	L	AL	A	AR	R
VN					
N		A		A	
F	A	A			
VF	A	A	A	A	A

Table 4: Optimized rule base (having five rules only) obtained using Technique 4 for eight-obstacle problem.

distance \ angle	L	AL	A	AR	R
VN		AR			
N	A	A			
F				A	
VF				A	

Techniques 2 and 4 are shown in Figures 5 and 6, respectively. Here, Technique 4 (simultaneous optimization of rules and membership functions) has elongated the membership function distribution so that classification of relative angle is uniform in the range of $(-90, 90)$ degrees. Because only 10 scenarios are considered during the optimization process, it could have been that in most cases the critical obstacles comes in the left of the robot, thereby causing more rules specifying L or AL to appear in the optimized rule base. By considering more scenarios during the optimization process, such bias can be avoided and equal number of rules specifying left and right considerations can be obtained. From Table 2, it can be observed that Technique 3 (optimization of rule base only) has resulted in a much quicker path than Technique 2 (optimization of membership function only). This is because finding a good set of rules is more important for the robot than finding a good set of membership functions. Thus, the optimization of rule base is a rough-tuning process and the optimization of the membership function distributions is a fine-tuning process. Among both the tables, in only one case (Scenario 6 in Table 2) the optimization of membership function for a optimized rule base has improved the solution slightly (Technique 4). In all other cases, the optimized solutions are already obtained during the optimization of rule-base only and optimization of membership function did not improve the solution any further.

Although the performance of Techniques 3 and 4 are more-or-less similar, we would like to highlight that Technique 4 is a more flexible and practical approach. The similarity in the performances of Techniques 3 and 4 reveals that optimizing rule base has a significant effect and

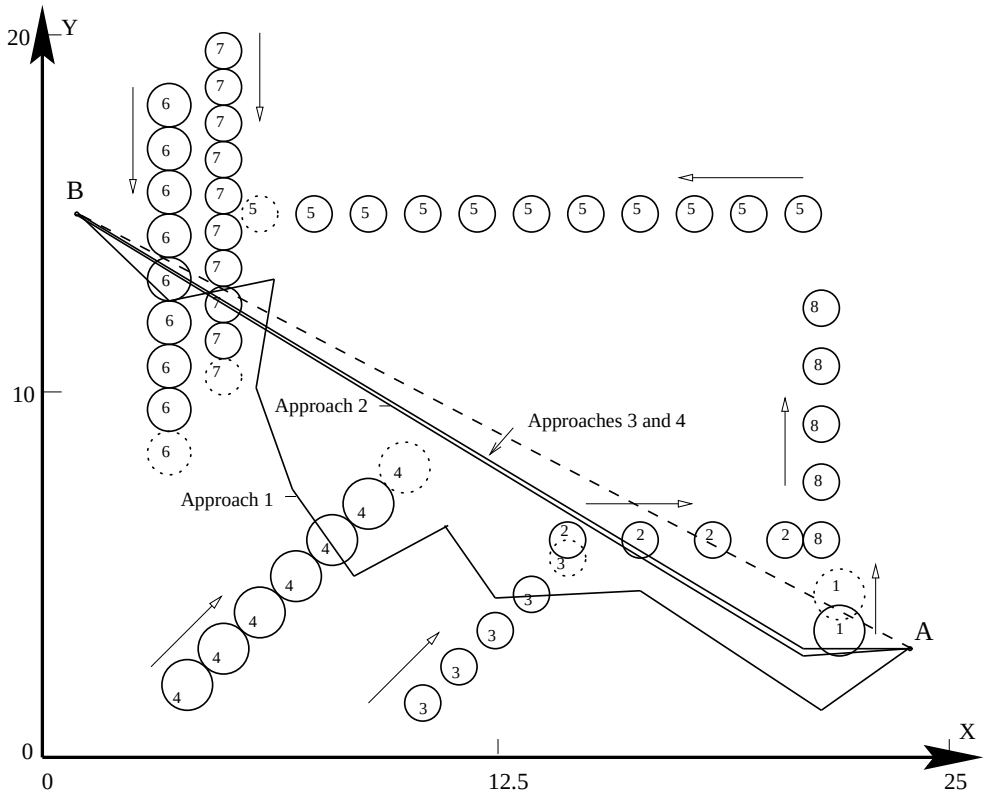

Figure 4: Optimized path found by all four approaches for the eight-obstacle problem are shown. There are seven obstacles and their movements are shown by an arrow. The location of the critical obstacle is shown by a dashed circle.

the optimization of the membership functions is only a secondary matter. Since the membership functions used in Technique 3 are developed by the authors and are reasonably good in these two problems, the performance of Technique 3 turns out to be good. However, for more complicated problems, we recommend using Technique 4, since it optimizes both the rule base and membership functions needed in a problem.

4 Approach 2: On-line Optimization with a Minimal Time Window

For a steady change in a problem (which is most usual in practice), we suggest an on-line optimization technique which we discuss next.

Let us assume that the change in the optimization problem is gradual in t. Let us also assume that each optimization iteration requires a finite time G and that τ_T iterations are needed to track the optimal frontier within an allowable performance level. An assumption we make here is that the problem does not change (or assumed to be constant) within a time interval t_T, such that $G\tau_T < t_T$. Thus, an initial $G\tau_T$ time is taken up by the optimization algorithm to track the new trade-off frontier and to make a decision for implementing a particular solution from the frontier. We expect that only a fraction of overall time is taken by the optimization algorithm, that is $\alpha = G\tau_T/t_T$ is expected to be a small value (say 0.25). After the optimal frontier is tracked, $(1-\alpha)t_T$ time is spent on using the optimized solution for the rest of the time period. Figure 7 illustrates this dynamic procedure. The objective function $f(x)$, hence also the optimum of $f(\mathbf{x})$,

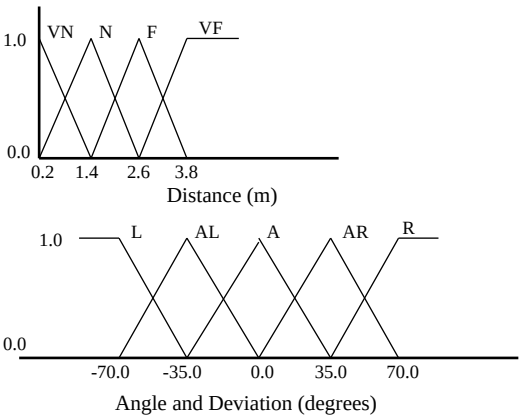

Figure 5: The optimized membership function obtained using Technique 2 for eight-obstacle problem.

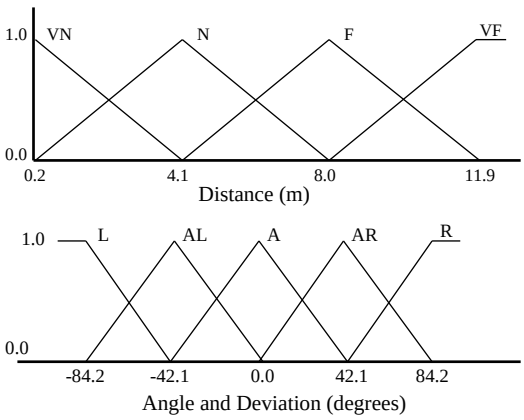

Figure 6: The optimized membership function obtained using Technique 4 for eight-obstacle problem.

changes with time.

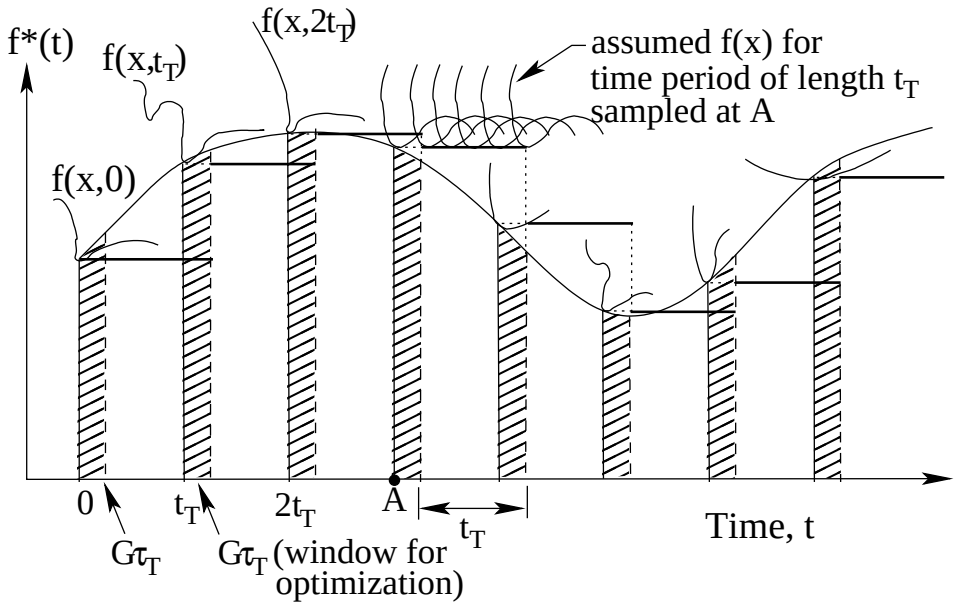

Figure 7: The on-line optimization procedure adopted in this study. For simplicity, only one objective is shown.

The choice of the time window is a crucial matter. If we allow a large value of t_T (allowing a proportionately large number of optimization iterations τ_T), a large change in the problem is expected, but the change occurs only after a large number of iterations of the optimization algorithm. Thus, despite the large change in the problem, the optimization algorithm may have enough iterations to track the trade-off optimal solutions. On the other hand, if we choose a small τ_T, the change in the problem is frequent (which approximates the real scenario more closely), but a lesser number of iterations are allowed to track new optimal solutions for a problem which has also undergone a small change. Obviously, there lies a lower limit to τ_T below which, albeit a small change in the problem, the number of iterations are not enough for an optimization algorithm

to track the new optimal solutions adequately. Such a limiting τ_T will depend on the nature of the dynamic problem and the chosen algorithm, but importantly allows the best scenario (and closest approximation to the original problem) which an algorithm can achieve. We suggest using an off-line study to find the limiting time window for an on-line optimization problem.

4.1 Dynamic NSGA-II for Handling Dynamic Multi-objective Optimization Problems

Here, we illustrate the working of the above on-line optimization approach on a dynamic multi-objective optimization problem. For this purpose, we suggested a modified NSGA-II procedure in an earlier study [5].

First, we introduce a test to identify whether there is a change in the problem at every generation. For this purpose, we randomly pick a few solutions from the parent population (10% population members used here) and re-evaluate them. If there is a substantial change in any of the objectives and constraint function values, we establish that there is a change in the problem. In the event of a change, all parent solutions are re-evaluated before merging parent and child population into a bigger pool. This process allows both offspring and parent solutions to be evaluated using the changed objectives and constraints.

In the dynamic NSGA-II, we introduce new randomly created solutions whenever there is a change in the problem. A ζ% of the new population is replaced with randomly created solutions. This helps to introduce new (random) solutions whenever there is a change in the problem.

4.2 Application to Bi-Objective Hydro-Thermal Power Scheduling

In a hydro-thermal power generation systems, both the hydro-thermal and thermal generating units are utilized to meet the total power demand. The optimum power scheduling problem involves the allocation of power to all concerned units, so that the total fuel cost of thermal generation and emission properties are minimized, while satisfying all constraints in the hydraulic and power system networks [11]. The problem is dynamic due to the changing nature of power demand with time. Thus, ideally the optimal power scheduling problem is truly an on-line dynamic optimization problem in which solutions must be found as and when there is a change in the power demand. In such situations, what can be expected of an optimization algorithm is that it tracks the new optimal solutions as quickly as possible, whenever there is a change.

The original formulation of the problem was given in Basu [1]. Let us also assume that the system consists of N_h number of hydro-thermal (P_{ht}) and N_s number of thermal (P_{st}) generating units sharing the total power demand, such that $\mathbf{x} = (P_{ht}, P_{st})$. The bi-objective optimization problem is given as follows:

$$\begin{array}{rl}
\text{Minimize} & f_1(\mathbf{x}) = \sum_{t=1}^{M} \sum_{s=1}^{N_s} t_T[a_s + b_s P_{st} + c_s P_{st}^2 + |d_s \sin\{e_s(P_s^{min} - P_{st})\}|], \\
\text{Minimize} & f_2(\mathbf{x}) = \sum_{t=1}^{M} \sum_{s=1}^{N_s} t_T[\alpha_s + \beta_s P_{st} + \gamma_s P_{st}^2 + \eta_s \exp(\delta_s P_{st})], \\
\text{subject to} & \sum_{s=1}^{N_s} P_{st} + \sum_{h=1}^{N_h} P_{ht} - P_{Dt} - P_{Lt} = 0, \quad t = 1, 2 \ldots, M, \\
& \sum_{t=1}^{M} t_T(a_{0h} + a_{1h} P_{ht} + a_{2h} P_{ht}^2) - W_h = 0, \quad h = 1, 2, \ldots, N_h, \\
& P_s^{min} \le P_{st} \le P_s^{max}, \quad s = 1, 2, \ldots, N_s, t = 1, 2, \ldots, M, \\
& P_h^{min} \le P_{ht} \le P_h^{max}, \quad h = 1, 2, \ldots, N_h, t = 1, 2, \ldots, M.
\end{array} \tag{2}$$

The transmission loss P_{Lt} term at the t-th interval is given as follows:

$$P_{Lt} = \sum_{i=1}^{N_h+N_s} \sum_{j=1}^{N_h+N_s} P_{it} B_{ij} P_{jt}. \tag{3}$$

This constraint involves both thermal and hydro-thermal power generation units. Due to its quadratic nature, it is handled directly to repair solution [5]. The problem is dynamic due to

changing nature of demand P_{Dt}. To make the demand varying in a continuous manner, we make a piece-wise linear interpolation of power demand values with the following (t, P_{dm}) values: (0, 1300), (12, 900), (24, 1100), (36, 1000) and (48, 1300) in (Hrs, MW). We keep the overall time window of $T = 48$ hours, but increase the frequency of changes (that is, increase M from four to 192, so that the time window t_T for each demand level varies from 12 hours to 48/192 hours or 15 minutes. It will then be an interesting task to find the smallest time window of statis which our dynamic NSGA-II can solve successfully. We run the dynamic NSGA-II procedure for $960/M$ (M is the number of changes in the problem) generations for each change in the problem, so as to have the overall number of function evaluations identical.

4.3 Results on Hydro-Thermal Power Dispatch Problem

We apply a dynamic NSGA-II procedure – an elitist non-dominated sorting genetic algorithm [3] – discussed above to solve the dynamic optimization problem. In this case, we have considered $\alpha = 1$, that is, the time window is equal to the time required for completion of the NSGA-II optimization run. To evaluate the performance of the dynamic NSGA-II procedure at the end of each time window, we initially treat each problem as a static optimization problem and apply the original NSGA-II procedure [3] for a large number (500) of generations so that no further improvement is likely. We call these fronts as ideal fronts and compute the hyper-volume measure using a reference point which is the nadir point of the ideal front. Thereafter, we apply our dynamic NSGA-II procedure and find an optimized non-dominated front after each time window. Then for each front, we compute the hyper-volume using the same reference point and then compute the ratio of this hyper-volume value with that of the ideal front. This way, the maximum value of the ratio of hyper-volume for an algorithm is one and as the ratio becomes smaller than one, the performance of the algorithm gets poorer.

Figures 8 to 11 show the hyper-volume ratio for different number of changes ($\tau_T = 4$ to 192) in the problem with different proportion of addition of random solutions, ζ. The figures also mark the 50th, 90th, 95th and 99th percentile of hyper-volume ratio, meaning the cut-off hyper-volume ratio which is obtained by the best 50, 90, 95, and 99 percent of M frontiers in a problem with M changes in power demand. Figures reveal that as M increases, the performance of the algorithm gets poorer due to the fact that a smaller number of generations ($\tau_T = 960/M$) was allowed to meet the time constraint. If a 90% hyper-volume ratio is assumed to be the minimum required hyper-volume ratio for a reasonable performance of the dynamic NSGA-II and if we base our confidence on the 95-th percentile performance, the figures indicate that we can allow a maximum of $M = 96$ changes (with a time window of 30 min.) in the problem. For this case, about 20 to 70% random solutions can be added whenever there is a change in the problem to start the next optimization and an acceptable performance of the dynamic NSGA-II can be obtained. Too low addition of random points does not introduce much diversity to start the new problem and too large addition of random solutions destroys the population structure which would have helped for the new problem. The wide range of addition for a successful run suggests the robustness of the dynamic NSGA-II procedure for this problem.

4.4 Automated Decision Making in a Dynamic Multi-Objective Optimization

A decision-making task is essential in a multi-objective optimization task to choose a single preferred solution from the obtained trade-off solution set. In a dynamic multi-objective optimization problem, there is an additional problem with the decision-making task. A solution is to be chosen and implemented as quickly as the trade-off frontier is found, and before the next change in the problem has taken place. This definitely calls for an automatic procedure for decision-making with some pre-specified utility function or some other procedure. Automated decision-making

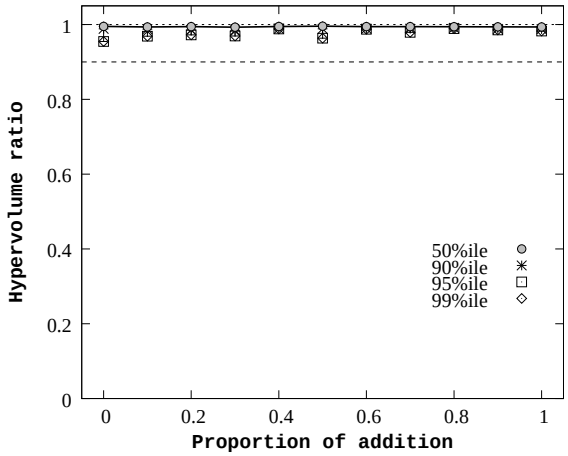

Figure 8: 3-hourly ($M = 16$) change with dynamic NSGA-II (Acceptable).

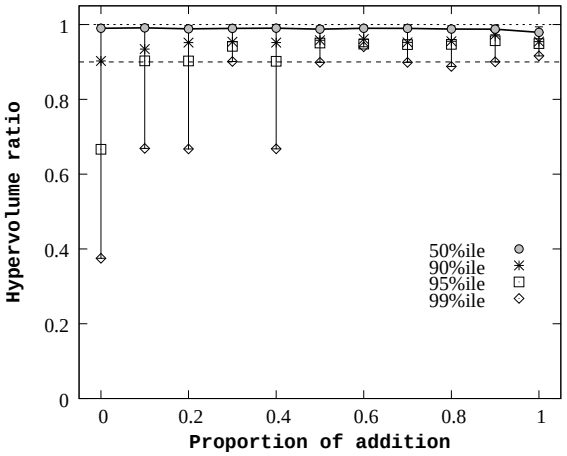

Figure 9: 1-hourly ($M = 48$) change with dynamic NSGA-II (Acceptable).

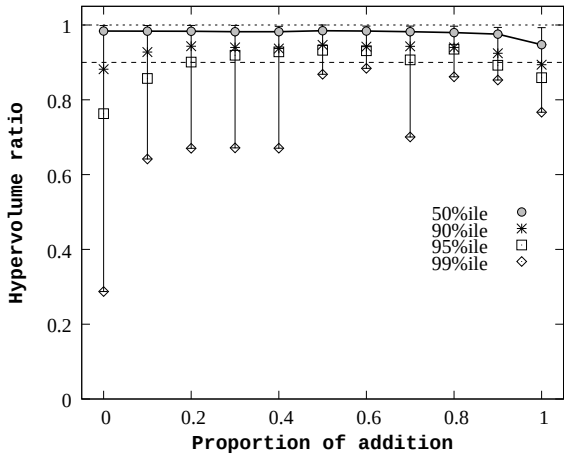

Figure 10: 30-min. ($M = 96$) change with dynamic NSGA-II (Acceptable and minimal).

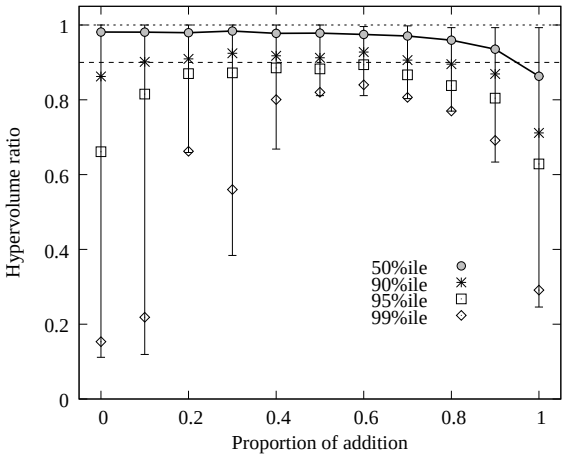

Figure 11: 15-min.($M = 192$) change with dynamic NSGA-II (Not acceptable).

is not available even in the multi-criteria decision making (MCDM) literature and is certainly a matter of future research, particularly if dynamic EMO is to be implemented in practice.

Here, we suggest a utility function based approach which works by providing different importance to different objectives. First, we consider a case in which equal importance to both cost and emission are given. As soon as a frontier is found for the forthcoming time period, we compute the pseudo-weight w_1 (for cost objective) for every solution $\mathbf{x}$ using the following term:

$$w_1(\mathbf{x}) = \frac{(f_1^{\max} - f_1(\mathbf{x}))/(f_1^{\max} - f_1^{\min})}{(f_1^{\max} - f_1(\mathbf{x}))/(f_1^{\max} - f_1^{\min}) + (f_2^{\max} - f_2(\mathbf{x}))/(f_2^{\max} - f_2^{\min})}. \quad (4)$$

Thereafter, we choose the solution with $w_1(\mathbf{x})$ closest to 0.5. A little thought will reveal that this task is different from performing a weighted-sum approach with equal weights for each objective. The task here is to choose the middle point in the trade-off frontier providing a solution equidistant from individual optimal solutions (irrespective of whether the frontier is convex or non-convex). Since the Pareto-optimal frontier is not known a priori, getting the frontier first and then choosing the desired solution is the only viable approach for achieving the task and such information cannot be utilized a priori.

To demonstrate the utility of this automated decision-making procedure, we consider the hydro-thermal problem with 48 time periods (meaning an hourly change in the problem). Figure 12 shows the obtained frontiers in solid lines and the corresponding preferred (operating) solution with a circle. It can be observed that due to the preferred importance of 50-50% to cost and emission, the solution comes nearly in the middle of each frontier. To meet the water availability constraint, the hydro-thermal units of $T_{h1} = 219.76$ MW and $T_{h2} = 398.11$ MW are computed and kept constant over time. However, four thermal power units must produce power to meet the remaining demand and these values for all 48 time periods are shown in Figure 13. The

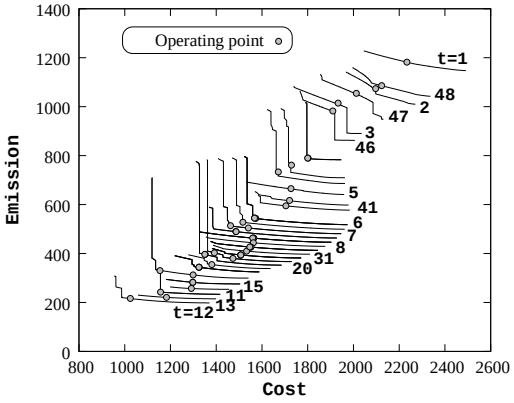

Figure 12: Operating solution for 50-50% cost-emission case.

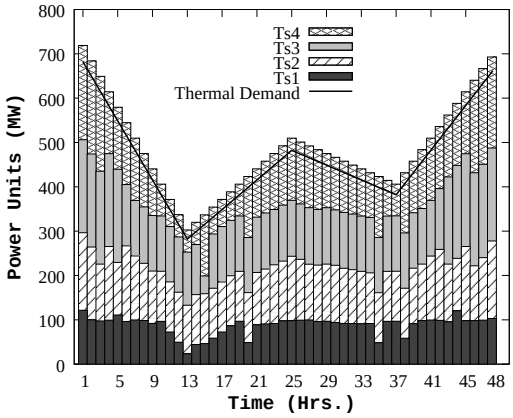

Figure 13: Variation of thermal power production for 50-50% cost-emission case.

changing pattern in overall computation of thermal power varies similar to that in the remaining demand in power. The figure also shows a slight over-generation of power to meet the loss term P_{Lt} given in equation 3.

Next, we compare the above operating schedule of power generation with two other extreme cases: (i) 100-0% importance to cost and emission and (ii) 0-100% importance to cost and emission. Table 5 shows the trade-off between cost and emission values for the three cases. Although this

Table 5: Different emphasis between cost and emission used in the automated decision-making process produces equivalent power generation schedules.

Case	Cost	Emission
50-50%	74239.07	25314.44
100-0%	69354.73	27689.08
0-100%	87196.50	23916.09

is one approach for executing an automated decision-making task, further ideas of choosing a preferred solution from an obtained set of trade-off solutions instantly using some previously defined principles must be worked on.

5 Conclusions

In this paper, we have dealt with solving on-line optimization problems. Specifically, we have suggested and demonstrated the working of two approaches: (i) an off-line optimization task

in which a set of optimal knowledge base is developed as guiding rules for handling changing problems on-line, and (ii) an on-line optimization approach in which the problem is considered static for a minimal amount of time window. It has been argued that for a problem having a rapid change in any of its parameters, objectives and constraints, the first approach may be more suitable. For a slow changing problem, the latter approach is more appropriate.

The working of the first approach is demonstrated by solving a robot navigation problem in which the obstacles move while the optimization is underway. A fuzzy rule base system is used to store the knowledge of an optimal action based on two given input parameters. The off-line optimization is able to find an optimal rule-base for achieving the task adequately on on-line scenarios.

The working of the second approach is demonstrated by solving a power dispatch problem that changes due to the ever-changing need of demand of power with time. To illustrate the generic nature of the approach, a two-objective version of the problem (minimizing cost and minimizing emission) have been considered. Based on a permissible performance limit, a dynamic NSGA-II approach has been able to identify a minimum time window (of 30 min.) for which the power demand can be considered static. Any faster consideration in the change of the problem was found to be too fast for the chosen algorithm to track the Pareto-optimal solutions of the problem.

Dynamic multi-objective optimization raises an important issue: an automated decision making task that must be performed as soon as the optimized trade-off front is found, as otherwise a delay in decision making may cause the problem to change significantly before the next round of optimization is performed. This paper has demonstrated one such automated decision-making approach, but this remains to be an open area of further research.

The approaches of this paper provides hope for applying single and multi-objective optimization techniques to on-line optimization tasks. A true implementation of these ideas in practice and further research for more sophisticated approaches would be the way forward.

Acknowledgments

The results and some texts of this paper are borrowed from author's earlier publications (references [4] and [5]). For details, readers are encouraged to refer these papers.

References

[1] M. Basu. A simulated annealing-based goal-attainment method for economic emission load disptch of fixed head hydrothermal power systems. *Electric Power and Energy Systems*, 27(2):147–153, 2005.

[2] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Heidelberg, Germany: Springer, 2001.

[3] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[4] K. Deb, D. K. Pratihar, and A. Ghosh. Learning to avoid moving obstacles optimally for mobile robots using a genetic-fuzzy approach. In *Parallel Problem Solving From Nature V*, pages 583–592, 1998.

[5] K. Deb, U. B. Rao, and S. Karthik. Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling bi-objective optimization problems. In *Proceedings of the Fourth International Conference on Evol. Multi-Criterion Optimization (EMO-2007)*, 2007.

[6] M. Farina, K. Deb, and P. Amato. Dynamic multiobjective optimization problems: Test cases, approximations, and applications. *IEEE Transactions on Evolutionary Computation*, 8(5):425–442, 2000.

[7] I. Hatzakis and D. Wallace. Dynamic multi-objective optimization with evolutionary algorithms: A forward-looking approach. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 1201–1208, 2006.

[8] Y. Jin and B. Sendoff. Constructing dynamic optimization test problems using the multiobjective optimization concept. In *Lecture Notes in Computer Science, 3005*, pages 525–536, 2004.

[9] D. Pratihar, K. Deb, and A. Ghosh. Fuzzy-genetic algorithms and time-optimal obstacle-free path generation for mobile robots. *Engineering Optimization*, 32:117–142, 1999.

[10] D. K. Pratihar, K. Deb, and A. Ghosh. Optimal path and gait generations simultaneously of a six-legged robot using a ga-fuzzy approach. *Robotics and Autonomous Systems*, 41:1–21, 2002.

[11] A. J. Wood and B. F. Woolenberg. *Power Generation, Operation and Control*. John-Wiley & Sons, 1986.