Multiple Trajectory Search for Unconstrained/Constrained Multi-Objective Optimization

Lin-Yu Tseng and Chun Chen

Abstract—**Many real-world optimization problems involve multiple conflicting objectives. Therefore, multi-objective optimization has attracted much attention of researchers and many algorithms have been developed for solving multi-objective optimization problems in the last decade. In this paper the multiple trajectory search (MTS) is presented and successfully applied to thirteen unconstrained and ten constrained multi-objective optimization problems. These problems constitute a test suite provided for competition in the Special Session & Competition on Performance Assessment of Constrained/Bound Constrained Multi-Objective Optimization Algorithms in CEC 2009. In the multiple trajectory search, a set of uniformly distributed solutions is first generated. These solutions will be separated into foreground solutions and background solutions. The search is focuses mainly on foreground solutions and partly on background solutions. The MTS chooses and applies one of the three local search methods on solutions iteratively. The three local search methods begin their search in a very large "neighborhood". Then the neighborhood contracts step by step until it reaches a pre-defined tiny size, after then, it is reset to its original size. By utilizing such size-varied neighborhood searches, the MTS effectively solves the multi-objective optimization problems.**

Index Terms—**Multi-objective optimization, multiple trajectory search, simulated orthogonal array, local search.**

I. Introduction

In recent years, many algorithms had been proposed for solving the multi-objective (MO) optimization problems. For example, the Pareto archived evolution strategy (PAES) [4], the Pareto envelope-based selection algorithm (PESA) [1], the strength Pareto evolutionary algorithm 2 (SPEA2) [11], the nondominated sorting genetic algorithm II (NSGA II) [2], the incrementing multi-objective evolutionary algorithm (IMOEA) [7], the cooperative coevolutionary algorithm (CCEA) [8] and the dominance-based multi-objective simulated annealing [6] are some of them. Also some studies on the design of multi-objective test problems and the performance evaluation of multi-objective optimization algorithms were also suggested (e.g. [3]). In this paper, we presented the multiple trajectory search (MTS) and applied it to solve the thirteen unconstrained and ten constrained multi-objective optimization benchmark problems [10] provided for competition on Performance Assessment of Constrained/Bound Constrained Multi-Objective Optimization Algorithms in CEC 2009. We had attended the Special Session & Competition on Performance Assessment of Multi-Objective Optimization Algorithms in CEC 2007 and proposed the MTS [9] then. The MTS presented in this paper is an improved version of the original version. The local search methods and the procedure to adjust the approximation set to the maximum allowable size have been greatly improved. The remainder of this paper is organized as follows. Section II gives some definitions. The proposed MTS is described in Section III. The experimental results are given in Section IV and the conclusion is drawn in Section V.

Lin-Yu Tseng is with the Institute of Networking and Multimedia, also with the Department of Computer Science and Engineering, National Chung Hsing University, 250 Kuo Kuang Road, Taichung, Taiwan 402, ROC (corresponding author; phone: 886-4-22874020; e-mail: lytseng@cs.nchu.edu.tw).

Chun Chen is a PhD student with the Department of Computer Science and Engineering, National Chung Hsing University, 250 Kuo Kuang Road, Taichung, Taiwan 402, ROC (e-mail:phd9401@cs.nchu.edu.tw).

II. Preliminaries

In this section, we give some definitions. Let x, y be two solutions. We say that x strictly dominates y (denoted by $x << y$) if x is better than y in all objectives. x dominates y (denoted by $x < y$) if x is not worst than y in all objectives and better in at least one objective. x weakly dominates y (denoted by $x \leqq y$) if x is not worse than y in all objectives. x is incomparable with y (denoted by $x \| y$) if neither x weakly dominates y nor y weakly dominates x. We define the Approximate Set to be a set of solutions such that any element in this set does not weakly dominate any other element in this set.

An orthogonal array OA (n, k, q, t) is an array with n rows and k columns which is a representative sample of n testing experiments that satisfies the following three conditions : (1) For the factor in any column, every level occurs the same number of times. (2) For the t factors in any t columns, every combination of q levels occurs the same number of times. (3) The selection combinations are uniformly distributed over whole space of all possible combinations. For more details, the reader may refer to [5]. The orthogonal arrays exist for only some specific n's and k's. So it is not appropriate to use the OA in some applications. We applied the simulated OA (SOA) in this study. The SOA satisfies only the first of the above mentioned three conditions, but it is easy to construct an SOA of any size.

III. Multiple Trajectory Search

In this section, we described the proposed multiple trajectory search (MTS) for the multi-objective optimization problem. The MTS utilizes three local search methods and aims to effectively search the solution space. After the algorithm terminates, the solutions found by the MTS will be

978-1-4244-2959-2/09/$25.00 ©

collected in the Approximation Set. Each time a new solution *s* has been found, the algorithm will decide whether to put *s* into the Approximation Set or not. And this decision is done by the following function AddToAppSet

Function AddToAppSet(*s*)
  If there is a *y* in Approximation Set and
    y weakly dominates *s*
  Then stop and **Return** FALSE
  Else delete every *y* in Approximation Set
      which is dominated by *s*
  put *s* into Approximation Set
  Return TRUE

In the MTS algorithm, initially *M* solutions uniformly distributed over the solution space are generated using the SOA. The local search will be applied to all *M* solutions at the first time. Afterwards, only the best ones will be put in the foreground and the local search will be applied to them. The other solutions are put in the background. It is noted that before applying the local search, a test is done in order to select the one of the three local search methods that is best fit in searching the landscape of the solution's neighborhood. When a new solution is generated by one of the local search method, its objective function values will be evaluated and the function AddToAppSet will be called to check if this solution should be added to the Approximation Set. As the search goes on, the size of the Approximation Set increases. Therefore, the MTS checks the size of the Approximation Set after the number of evaluation reaches the pre-specified value. If the size is greater than the allowable maximum size, the procedure AdjustAppSet will be called to reduce the size.

In our experiments, *M* is set to 40 and *#ofForeground* is set to 5. Hence, five best solutions are put in the foreground and the other 35 solutions in the background. From time to time, a solution in the foreground may be put in the background and vice versa.

The pseudocode of the MTS for the multi-objective optimization is listed in the following. In this pseudocode, *U*[*j*] and *L*[*j*] are the upper bound and the lower bound of the *j*th dimension of the search space.

Multiple Trajectory Search
/*Generate *M* initial solutions */
Build simulated orthogonal array $SOA_{M\times N}$
Use $SOA_{M\times N}$ to generate *M* initial solutions *X*[*1*], *X*[*2*]…, *X*[*M*]
Evaluate objective function values of *X*[*i*]*'s*
For *i=1* to *M*
  Enable[*i*]←TRUE
  Improve[*i*] ←TRUE
  SearchRangeXi[*j*] = (*U*[*j*] – *L*[*j*])/*2* for *j = 1,2, ..., N*
End For
While (*#ofEvaluation* *predefined_max_evaluation*)
  For *i=1* to *M*
    If *Enable*[*i*]=TRUE
    Then *GradeXi*←0
        LS1_TestGrade←0
        LS2_TestGrade←0
        LS3_TestGrade←0
        For *j* =1 to *#ofLocalSearchTest*
          LS1_TestGrade←*LS1_TestGrade*+
              LocalSearch1(*X*, *i*, *SearchRangeXi*)
          LS2_TestGrade←*LS2_TestGrade*+
              LocalSearch2(*X*, *i*, *SearchRangeXi*)
          LS3_TestGrade←*LS3_TestGrade*+
              LocalSearch3(*X*, *i*, *SearchRangeXi*)
        End For
        Choose the one with the best *TestGrade* and
        let it be LocalSearchK /* K may be 1, 2, or 3 */
        For *j =1* to *#ofLocalSearch*
          GradeXi←*GradeXi*+
              LocalSearchK(*X*, *i*, *SearchRangeXi*)
        End For
  End For
  For *i =1* to *M*
      Enable[*i*] ← FALSE
  End For
  Choose *#ofForeground* X_i*'* s whose *GradeXi* are best
    among the *M* solutions and set their corresponding
    Enable[*i*] to TRUE
End While
If size of Approximation Set > max_size_of_AppSet
  Then AdjustAppSet()

In the MTS, three local search methods are used for searching different neighborhood structures. Local Search 1 is used to search along each dimension. Local Search 2 is similar to Local Search 1 except that it is used to search along about one-fourth of the dimensions. Local Search 3 is quite different from Local Search 1 and 2, although it also searches along each dimension, it checks a set of solutions evenly spaced on this dimension within a predetermined upper bound and lower bound. In general, a local search method is used to search a small neighborhood, but this is not the case for the proposed local search methods. In fact, the three local search methods begin their search in a very large "neighborhood". Then the neighborhood contracts step by step until it reaches a pre-specified tiny size, after then, it is reset to its original size. So the proposed local search methods search neighborhoods of variable sizes, cyclically from large ones to small ones and then back to large ones.

In local search methods, each time the objective function values of a solution in the neighborhood of *X* are evaluated, the procedure Grading will be called to check if this solution can be added to the Approximation Set, also the corresponding bonus will be added to the grade of *X*. In the following, the procedure Grading is given. In this procedure, we say that a solution *x* number-dominates another solution *y* if and only if *x* and *y* are incomparable, and the number of the objective

function values with which x is better than y is larger than the number of objective function values with which y is better than x.

```
Procedure Grading(X, Old_X, grade, Improve)
If AddToAppSet(X)
   Then grade←grade+BONUS1
If X dominates Old_X  or  X number-dominates Old_X
   Then grade←grade+BONUS2
        Improve←TRUE
```

Next, the pseudocodes of the three local search methods are described in the following. In these pseudocodes, $X[k]$ is a solution that is passed from the main program. $SR[i]$ is the search range along the dimension i, $Improve[k]$ is used to indicate if an improvement was made when the last Local Search 1 or Local Search 2 was applied. Random{-1, 1} will randomly choose one of -1 and 1, and Random{0, 1, 2, 3} will randomly choose one of 0, 1, 2 and 3.

```
Function LocalSearch1(X, k, SR)
If Improve[k]=FALSE
   Then  SR[i]←SR[i]/2 for all i
         If all SR[i] < 1e-8
            Then SR[i]←(U[i] – L[i]) *0.4 for all i
Improve[k] ←FALSE
For i = 1 to N
   D[i] ←Random{-1,1}
End For
For  i from 1 to N in a randomly permuted order
   Old_X_k ← X[k]          /* Save X[k] in Old_X_k */
   X[k][i] ← X[k][i] + SR[i] *D[i] /*Search in one direction*/
   Grading (X[k], Old_X_k, grade, Improve[k])
   If Old_X_k weakly dominates X[k]
      Then X[k] ← Old_X_k        /* Restore X[k]*/
           X[k][i] ← X[k][i] - 0.5*SR[i]*D[i]
                    /*Search in the opposite direction*/
           Grading (X[k], Old_X_k, grade, Improve[k])
           If Old_X_k weakly dominates X[k]
              Then X[k] ← Old_X_k
End For
Return grade
```

```
Function LocalSearch2(X, k, SR)
If Improve[k]=FALSE
   Then  SR[i]←SR[i]/2 for all i
         If all SR[i] < 1e-8
            Then SR[i]←(U[i] – L[i]) *0.4 for all i
Improve[k] ←FALSE
For  l = 1 to N
   Old_X_k ← X[k]       /* Save X[k] in Old_X_k */
   For i = 1 to N
      r[i] ←Random{0,1,2,3}
      D[i] ←Random{-1,1}
   End For
    For =1 to N
       If r[i]=0      /*Check if i is in the chosen N/4 dimensions*/
          Then X[k][i] ← X[k][i] + SR[i]*D[i]
    End For
    Grading (X[k], Old_X_k, grade, Improve[k])
    If Old_X_k weakly dominates X[k]
    Then X[k] ← Old_X_k      /* Restore X[k] */
         For i = 1 to N
           If r[i] = 0
              Then X[k][i] ← X[k][i] - 0.5*SR[i]*D[i]
         End For
         Grading (X[k], Old_X_k, grade, Improve[k])
        If Old_X_k weakly dominates X[k]
           Then X[k] ← Old_X_k
End For
Return grade
```

```
Function LocalSearch3(X, k, dummy)
  For i = 1 to N
    SearchU[i] ← U[i]
    SearchL[i] ← L[i]
    Disp[i] ← (SearchU[i] - SearchL[i])/10
  End For
  Best ← X[k]
  While (at least one of Disp[i] > 1e-3)
    For i from 1 to N in a randomly permuted order
      Find all solution points Y_1, Y_2, …,Y_l(i) that have distance
        values of integral multiple of Disp[i] with Best along
        the dimension i and within the range [SearchL[i],
        SearchU[i]].
      For j=1 to l(i)
        Grading(Y_j, Best, grade, dummy)
        If Y_j dominates Best then Best ←Y_j
      End For
      SearchU[i] ← min(Best[i] + 2*Disp[i], U[i])
      SearchL[i] ← max(Best[i] - 2*Disp[i], L[i])
      Disp[i] ← (SearchU[i] - SearchL[i])/10
    End For
End While
X[k] ← Best
Return grade
```

Now, let us describe the procedure AdjustAppSet in the following. In this procedure, for each objective function f_i, the solutions in the Approximation Set that have the smallest f_i's values will first be removed from the Approximation Set and put into the New Approximation Set. Afterwards, solutions in the Approximation Set will be chosen, removed from the Approximation Set, and put into the New Approximation Set one by one until the size of the New Approximation Set reaches the allowable maximum size. Each time the solution in the Approximation Set that has the largest distance with its nearest neighbor in the New Approximation Set will be chosen. This choosing scheme aims to make the solutions in the New Approximation Set as diverse as possible.

```
Procedure AdjustAppSet()
```

New Approximation Set ←Ø
For i =1 to number_of_objective_functions
Sort all solutions in the Approximation Set with respect to objective function f_i's values
Mark solutions that have the smallest f_i's values
End For
Remove all marked solutions from Approximation Set and put them in New Approximation Set
While (|New Approximation Set| < max_size_AppSet)
For each solution s_i in Approximation Set, find the nearest neighbor t_j in New Approximation Set
Let d_i = distance(s_i, t_j)
End For
Find $d_k = \max_i d_i$
Remove s_k from Approximation Set and put it into New Approximation Set
End While
Approximation Set ← New Approximation Set

The above mentioned MTS is for the unconstrained multi-objective optimization problems. As for the constrained multi-objective optimization problems, we take each constraint as an additional objective function with function values 0 and 1. If the constraint is satisfied, the function value is 0; Otherwise, the function value is 1. Therefore, the above mentioned MTS can be applied to solve the constrained multi-objective optimization problems except that a small modification must be made for the procedure AdjustAppSet. In AdjustAppSet, only the original objective functions are considered, and those solutions in the Approximation Set that satisfy all constraints are first chosen, removed from the approximation Set, and put into the New Approximation Set. After that, everything goes the same as the original AdjustAppSet.

IV. Experimental Results

The PC configuration, the parameters setting and the performance of the MTS are stated in the following.

A. PC Configuration

System: Linux
RAM: 2GB
CPU: Intel Xeon E5310 1.6 GHz
Computer Language: C++

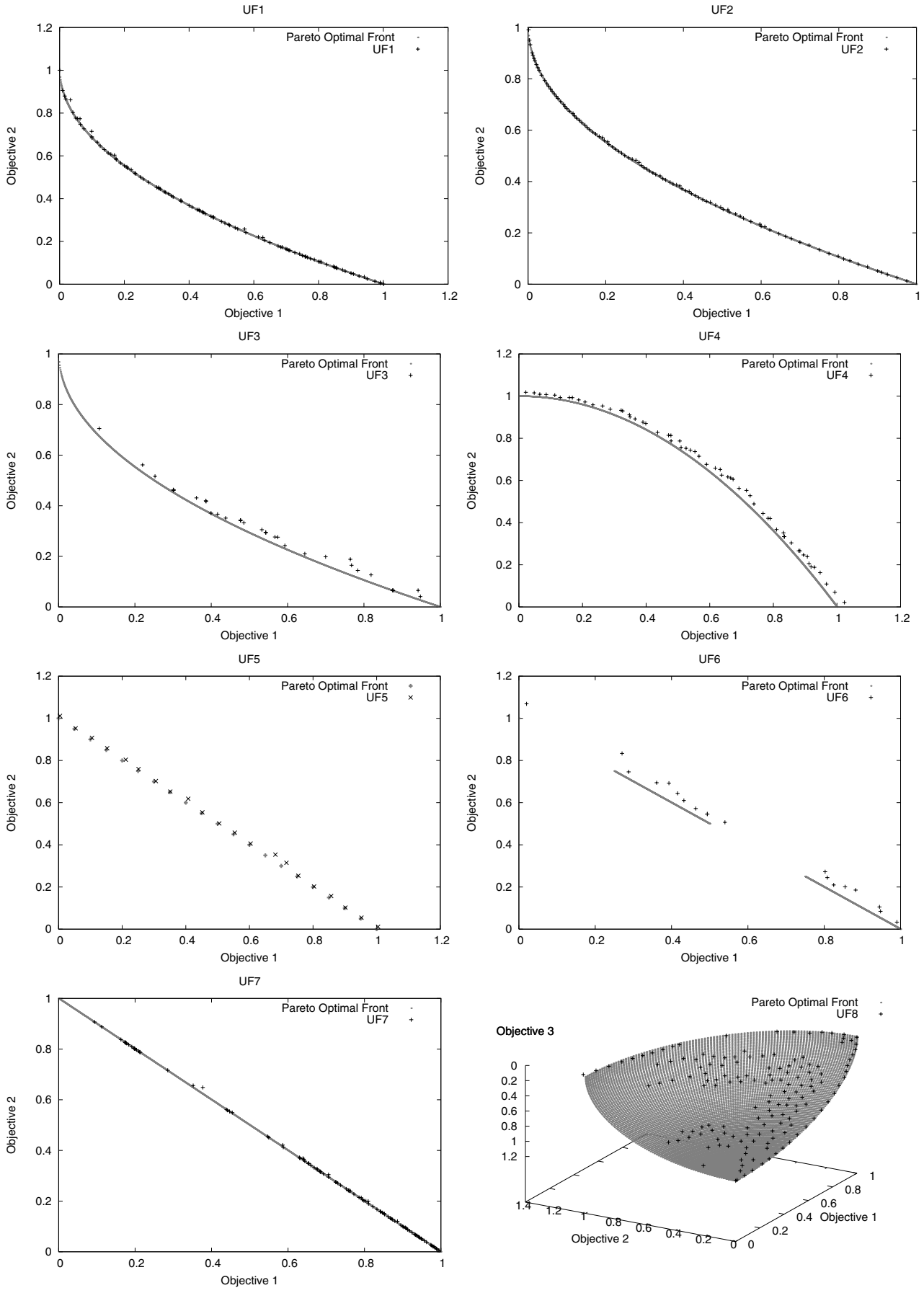
B. Algorithmic Parameter Setting

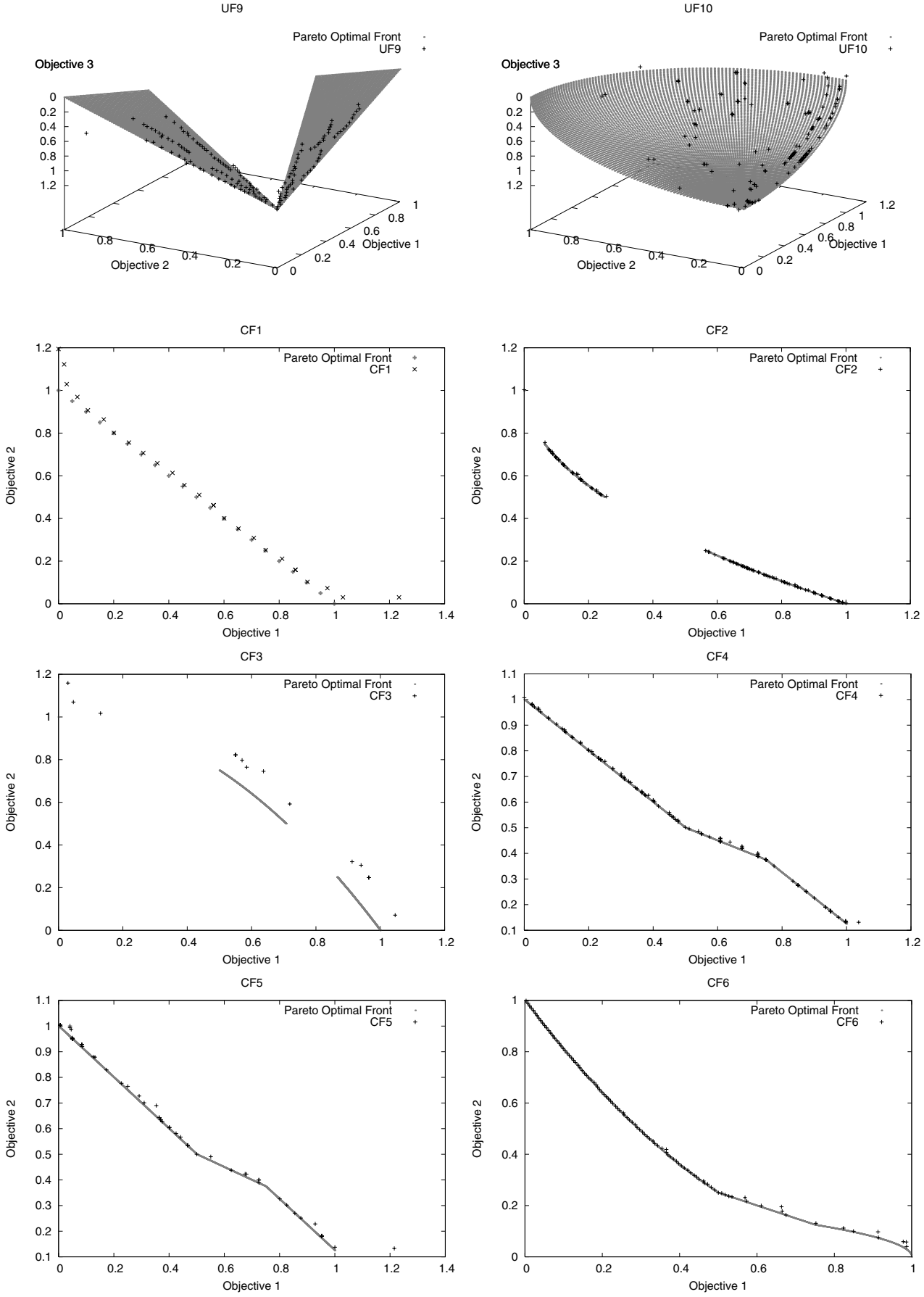
(a) M = 40
(b) *BONUS1* = 9
(c) *BOUND2* = 2
(d) *#ofLocalSearchTest* = 5
(e) *#ofLocalSearch* = 45
(f) *#ofForeground* = 5

C. Experimental Results

TABLE I
Average IGD values of the 30 final approximation sets obtained for each test problem

Function	Average	Std. deviation	Max	Min	CPU Time (s)
UF1	0.00646722	0.000348504	0.00722128	0.00578230	0.657000
UF2	0.00615756	0.000508008	0.00749917	0.00518893	1.156330
UF3	0.05310720	0.011736600	0.07786330	0.03739470	1.427000
UF4	0.02356120	0.000664177	0.02495010	0.02248650	1.102000
UF5	0.01489430	0.003277170	0.02203690	0.00974315	0.913667
UF6	0.05917810	0.010622400	0.09026840	0.04163100	1.146670
UF7	0.04079490	0.014445600	0.08110390	0.01595160	0.617667
UF8	0.11251700	0.012933500	0.13865200	0.09092770	2.527670
UF9	0.11442300	0.025495500	0.18269400	0.06246370	1.609330
UF10	0.15306500	0.015833100	0.19801400	0.12450400	0.996667
UF11	0.45505200	0.037163000	0.52717900	0.39306000	3.603670
UF12	305.20600000	45.118600000	407.36800000	222.52100000	2.502000
UF13	1.90798000	0.025373100	1.96630000	1.87281000	837.824000
CF1	0.01918730	0.002568030	0.02359740	0.01385480	0.325667
CF2	0.02677900	0.014715200	0.05181580	0.00414239	0.321000
CF3	0.10446000	0.015595300	0.14282800	0.07530050	0.365000
CF4	0.01109600	0.001368680	0.01427600	0.00893742	0.210333
CF5	0.02077970	0.002421980	0.02783230	0.01756540	0.256000
CF6	0.01616860	0.005985450	0.03811290	0.00956871	0.321333
CF7	0.02469530	0.004654400	0.03714410	0.01869120	0.393000
CF8	1.08544000	0.219108000	1.42867000	0.62196800	0.398000
CF9	0.08513920	0.008191600	0.09628290	0.07207180	12.704300
CF10	0.13764800	0.009215920	0.16347300	0.11731700	1.061670


UF1
Pareto Optimal Front
UF1
Objective 2
Objective 1
UF2
Pareto Optimal Front
UF2
Objective 2
Objective 1
UF3
Pareto Optimal Front
UF3
Objective 2
Objective 1
UF4
Pareto Optimal Front
UF4
Objective 2
Objective 1
UF5
Pareto Optimal Front
UF5
Objective 2
Objective 1
UF6
Pareto Optimal Front
UF6
Objective 2
Objective 1
UF7
Pareto Optimal Front
UF7
Objective 2
Objective 1
Pareto Optimal Front
UF8
Objective 3
Objective 2
Objective 1


UF9
Pareto Optimal Front
UF9
Objective 3
Objective 2
Objective 1
UF10
Pareto Optimal Front
UF10
Objective 3
Objective 2
Objective 1
CF1
Pareto Optimal Front
CF1
Objective 2
Objective 1
CF2
Pareto Optimal Front
CF2
Objective 2
Objective 1
CF3
Pareto Optimal Front
CF3
Objective 2
Objective 1
CF4
Pareto Optimal Front
CF4
Objective 2
Objective 1
CF5
Pareto Optimal Front
CF5
Objective 2
Objective 1
CF6
Pareto Optimal Front
CF6
Objective 2
Objective 1

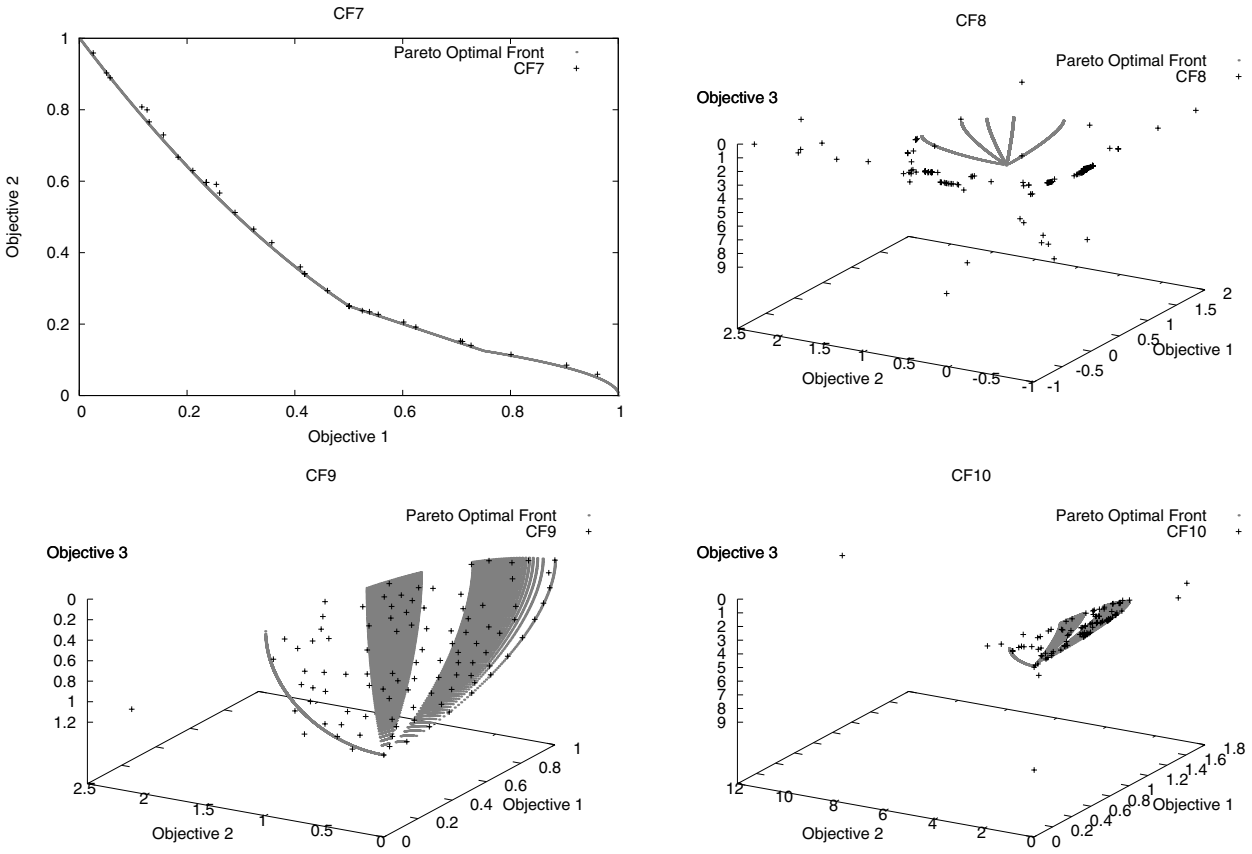


Fig. 1. The Pareto front and the final approximation set with the smallest IGD value.

There are six algorithmic parameters used in the MTS. The parameter setting is based on the results of the preliminary computational studies. The mean, standard deviation, the smallest and the largest values of the IGD values of the 30 final approximations obtained for each test instance are shown in Table I. In this table, the CPU time for UF13 is much larger than those of other problems because we obtain a very large approximation set in the process of solving UF13 and we have to find the nearest neighbor of every solution in the approximation set when using procedure AdjustAppSet to reduce the size of the approximation set to the maximum allowable size. Observing Table I, it is noted that the average IGD values are good for all problems except UF12. We are still trying to find out the reason why the IGD value is so large for UF12. The Pareto front and the final approximation set with the smallest IGD value for each test instance with two or three objectives are plotted in Figure 1. The approximation sets fit the Pareto front quite well for most instances.

V. Conclusion

In this paper the multiple trajectory search (MTS) is presented and applied to the test suite that contains thirteen unconstrained and ten constrained multi-objective optimization problems. This test suite is designed for the competition on Performance Assessment of Constrained/Bound Constrained Multi-Objective Optimization Algorithms in CEC 2009. The MTS utilizes three local search methods that search in different neighborhood structures. And the sizes of neighborhoods in which local search methods search vary cyclically from large to small and then back to large. Hence, the MTS may make effective search in the search space. As indicated by the average IGD values, the MTS performs well for every problem except UF12.

Acknowledgment

The authors gratefully acknowledge the support of the National Science Council of ROC under the contract NSC 96-2628-E-005-074-MY3.

References

[1] D. W. Corne, J. D. Knowles, and M. J. Oates, "The Pareto envelope-based selection algorithm for multiobjective optimization," in *Proc. Parallel Problem Solving Nature VI Conf.*, 2000, pp. 839–848.

[2] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 182–197, 2002.

[3] S. Huband, P. Hingston, L. While, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Trans. Evol Comput.*, Vol. 10, no.5, pp. 477-506, 2006.

[4] J. D. Knowles and D.W. Corne, "Approximating the nondominated front using the Pareto archived evolution strategy," *Evol. Comput.*, vol. 8, no.2, pp. 149–172, 2000.

[5] D. C. Montgomery, *Design and Analysis of Experiments*, 3rd ed. New York:Wiley. 1991
[6] K. I. Smith, R. M. Everson, J. E. Fieldsend, C. Murphy, R. Misra. "Dominance-base multiobjective simulated annealing," *IEEE Trans. Evol Comput.*, Vol. 12, no.3, pp. 323 -342, 2008.
[7] K. C. Tan, T. H. Lee, and E. F. Khor, "Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization," *IEEE Trans. Evol. Comput*., vol. 5, pp. 565–588, 2001.
[8] K. C. Tan, Y. J. Yang, and C. K. Goh "A distributed coorperative coevolutionary algorithm for multiobjective optimization," *IEEE Trans. Evol Comput*. Vol 10, No. 5, pp. 527-549, 2006.
[9] L. Y. Tseng and C. Chen, "Multiple trajectory search for multiobjective optimization," *Proceedings of 2007 IEEE Congress on Evol. Comput*., .pp. 3609-3616, 2007.
[10] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari, "Multiobject optimization test Instances for the CEC 2009 special session and competition," Technical Report CES-487, University of Essex and Nanyang Technological University, September. Available: http://www.ntu.edu.sg/home/EPNSugan/
[11] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," Computer Engineering and Networks Laboratory (TIK), Swiss Federal Inst. Technology (ETH), Zurich, Switzerland, 103, 2001.