Pareto Tree Searching Genetic Algorithm

Approaching Pareto Optimal Front by Searching Pareto Optimal Tree

Technical Report NK-CS-2001-002

Xianming Chen
Department of Computer Science, Nankai University, Tianjin, China
xchen@nankai.edu.cn

Abstract: A new multi-objective evolutionary algorithm, Pareto Tree Searching GA (PTSGA), is introduced in this paper. The algorithm is based on a recursive binary division of the objective space, which is represented by a hyper binary tree (Pareto Tree) structure. Only non-dominated population is kept in the tree, and Pareto comparison is carried out directly on the binary location codes of tree nodes, with comparison granularity ranging from the coarsest of the root level to the finest of the leaf level; thus the algorithm is efficient. On the other hand, population diversity is achieved readily because the tree depth will become deeper, and correspondingly the comparison precision finer, only in the later stage of the evolution. Finally, tree size (non-dominated population size) reducing, niching and fitness assigning are all performed based on the resident count of sub-trees; this ensures a good distribution and spread of the resulting non-dominated front. Preliminary experimental comparisons indicate that the new algorithm is the best or close to the best among all the successful MOEAs today.

Organization of this paper: Section one introduces some basic concepts related to multi-objective optimization problems and multi-objective evolutionary algorithms. Section two gives a brief survey of most successful multi-objective evolutionary algorithms in recent years. Section three presents the new algorithm in detail, and finally section four gives the experiment result.

1. Background of Multi-Objective Evolutionary Algorithm

(1) MOOP (Multi-Objective Optimization Problems)

Unlike single objective optimization problem (SOOP), in MOOP, we are optimizing simultaneously more than one objectives, which are usually contradictory, for example, less cost, better performance, and lower energy consumption as the objectives of software system design.

(2) Pareto Dominance Relation, Pareto Optimal set, and Pareto Optimal Front.

Solution *a* is said to weekly dominates solution *b* if all objective components of *a* is bigger or equal to (for maximization problem) those of *b*, dominates *b* if at least one of the above comparison is not equal, and equal to *b* if all the above comparisons are equal. If none of these relations hold, *a* is indifferent to *b*. To simplify discussion, in this context we give each pair of individuals, as well as each pair of objective vectors, the same Pareto dominance relation as that of their corresponding pair of solutions in the decision space.

A solution is optimal when it is non-dominated by all other feasible solutions. Pareto-optimal set consists of all Pareto-optimal solutions. The image of Pareto-optimal set under evaluation (vector) function is Pareto-optimal front or simply Pareto front. Actually, it is generally impossible to know the actual optimal set and the corresponding Pareto optimal front.

(3) Objectives of MOEA (Multi-Objective Evolutionary Algorithm)

Zitzler (1999) indicates three objectives of MOEA:

a. The distance of the resulting nondominated front to the Pareto-optimal front should be minimized.
b. A good (in most cases uniform) distribution of the solutions found is desirable.
c. The spread of the obtained nondominated front should be maximized, i.e., for each objective a wide range of values should be covered by the nondominated solutions.

(4) Key Issues of MOEA

a. Pareto Ranking and Fitness Assignment. Since Pareto dominance is only a partial order relation, in MOEA either a total order is constructed or any pair are made comparable temporarily (as in tournament selection). The methods used may differ from one algorithm to another, but typically are all based on Pareto dominance relation and niching information.
b. Niching and population diversity. They are actually closely related with the above problem. Niching, no mater how it is done, is aimed simultaneously at two goals: maintaining population diversity and making any two individuals comparable when they are indifferent.
c. Size reduction of current non-dominated set. This problem comes with the conclusion (Zitzler and Thiele 1999; Zitzler 1999) that elitism always pays in MOEA (unlike SOOP GA). For most MOOP, especially of continuous ones, the external population that contains nondominated individuals found so far, will increase quickly. As a result, some mechanism of size reduction is required. To achieve a good spread and distribution of the non-dominated front, niching is again the most important technique used here.

2. Most successful Multi-Objective Evolutionary Algorithms

Many new algorithms have been designed in recent years. Here, for comparison with our new algorithm later, we only give a brief introduction to the most successful ones; all of them are evolutionary algorithms using Pareto comparison. More detailed surveys can be found in Coello (1999), Coello (1999a), Fonseca and Fleming (1995), Veldhuizen and Lamont (2000).

MOGA: Multi-Objective Genetic Algorithm by Fonseca and Fleming (1993).
Each individual is assigned a Pareto rank, which is the total number of individuals in current population it dominates. Niching is performed to differentiate between individuals of the same rank.

NPGA: Niched Pareto Genetic Algorithm by Horn and Nafpliotis (1993)
Here tournament selection with the help of comparison set is used. If one candidate is nondominated regarding the set, and the other is not, the former will win the tournament. In the event of a tie, niche count is used to determine the winner.

NSGA: Nondominated Sorting Genetic Algorithm by Srinivas and Deb (1994).
Pareto ranking is done as following: First, the same raw fitness value is assigned to all the non-dominated individuals in current individual set. Fitness sharing is then performed among them to generate actual fitness value. The process will be repeated recursively for the remained set with the above non-dominated individuals deleted, but with a lower raw fitness.

SPEA: Strength Pareto Evolutionary Algorithm by Zitzler and Thiele (1999).
SPEA has two populations, a traditional one, and a non-dominated external one. Both of them participate in genetic recombination. Each individual in the external population is assigned a stronger "strength" value if it dominates more individuals in the traditional population. On the other hand, each individual in the traditional population is assigned a greater fitness value if it is dominated by more individuals in the external population. Here, the greater fitness value an individual has, the worse it is. Unlike all the above algorithms, niching in SPEA is achieved implicitly also by the dominance comparison.
Finally, external population size is reduced simply by clustering.

PAES: Pareto Archived Evolutionary Strategy by Knowles, and Corne (1999).
In its essential form, PAES is 1+1 Evolutionary Strategy. like SPEA, it uses an external non-dominated population (archive), but only to determine whether or not the mutant should replace the current individual and be inserted in the archive in case of a tie. Another novel feature of this approach is that a

binary division of objective space is used to implement crowding strategy.

3. Pareto Tree Searching Genetic Algorithm.

Our algorithm differs from all the current algorithms mentioned in section 2 in nearly all the essential features of MOEA. Its most significant characteristic is that, Pareto relation is compared on binary encoded integers (instead of float objective vectors) and with comparison granularities ranging from the coarse to the fine. Also noteworthy is that a tree structure is used to represent current **nondominated** population, and all the key issues of MOEA, such as niching, fitness assignment and reduction, are all performed (as one action) efficiently on the tree structure. Below we will give the basic concepts and the core procedures used in this algorithm.

(1) Recursive binary division of objective space. For the set of all the objective vectors in certain generation, there exists a least upper bound objective vector, which weekly dominates the whole set, and is dominated by all other dominating objective vectors regarding the whole set. Similarly there exist a most lower bound objective vector. From these two objective vectors, a hyper rectangular can be constructed. We recursively divide this hyper rectangular to get grids of different size corresponding to different division levels. See figure one for an example with objective dimension 2, and division depth 3.

(2) Hyper binary tree. The recursively-generated grids can be best represented by a tree structure. The original undivided hyper rectangular is the node of level 0, i.e., root node. The first level of binary division generates nodes in the first level of the tree, and so on. We call such a tree hyper binary tree. See figure 2 for a detailed illustration.

(3) Binary encoding of node (grid) locations. Each node in the tree (each grid in the objective space) is given a location code, which is simply a concatenation of binary strings. There is one such binary string for each objective dimension. Each string consists of 1 + DEPTH (division depth) bits, with one bit per division level and an extra Pareto Comparison Bit ('x' in the figure) inserted for efficient Pareto comparison. Initially at level 0, all these bits are set to 0. Later in any division level j, the j'th most significant bit will be set to 1, if the node (grid) is in the upper binary division of the objective space in the considered dimension

(4) Pareto comparison based on location codes and comparison granularity. Unlike Pareto comparison in all the MOEA algorithms so far, we compare individuals based on their encoded node locations. As each individual occupies simultaneously all the nodes in a path from the root to the appropriate leaf node in the tree, there are actually a series of such node comparisons from level 0 down to the leaf level, with the comparison granularity ranging from the coarsest to the finest. The finest comparison granularity in the leaf level actually determines the comparison precision of current generation.

As the location code of each node is actually a concatenation of binary strings, each of which represents the relative objective value in the respective dimension, a simple subtraction of the location codes can indicate the Pareto relation of any pair of nodes. The simple algorithm is listed below. (PCBs : the reserved Pareto Comparison Bits mentioned above)

Node_compare*(loc1, loc2)* { // supposing maximization.
If ***loc1*** equal to ***loc2***, return ***"same".***
Set **PCBs** of the smaller one of ***loc1*** and ***loc2,*** and subtract it by the greater one.
If the resulting Pareto comparison bits are all ***0***, return ***the smaller one is dominated***.
Else return "***indifferent***";
}

Fig 1. Recursive Binary Division of Objective Space

An example with objective dimensions 2, and division depth 3. Under the Pareto comparison granularity of level 1, only individual 12 is dominated by [4,5,6,7,8,11] (here '[ ]' means all these individuals occupy the same grid of this level, and consequently are "equal" to each other). In level 2, one more individual, 13, is dominated both by [1,2,3] and [4,5]. In level 3, another individual, 11, is dominated by [5] and [6]. All the other individuals are non-dominated to each other under current comparison precision, and their (shadowed) grids represent the current non-dominated front.

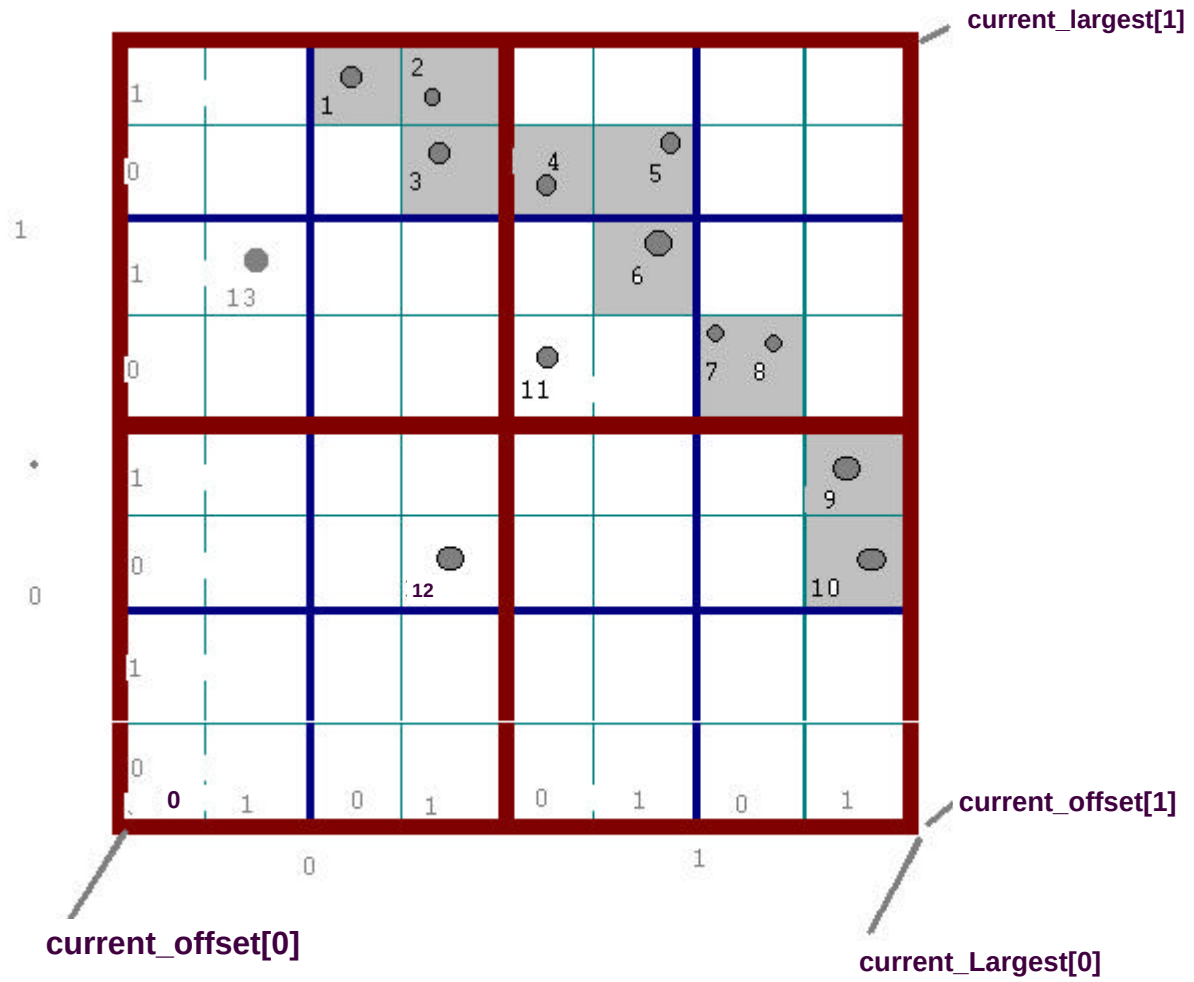
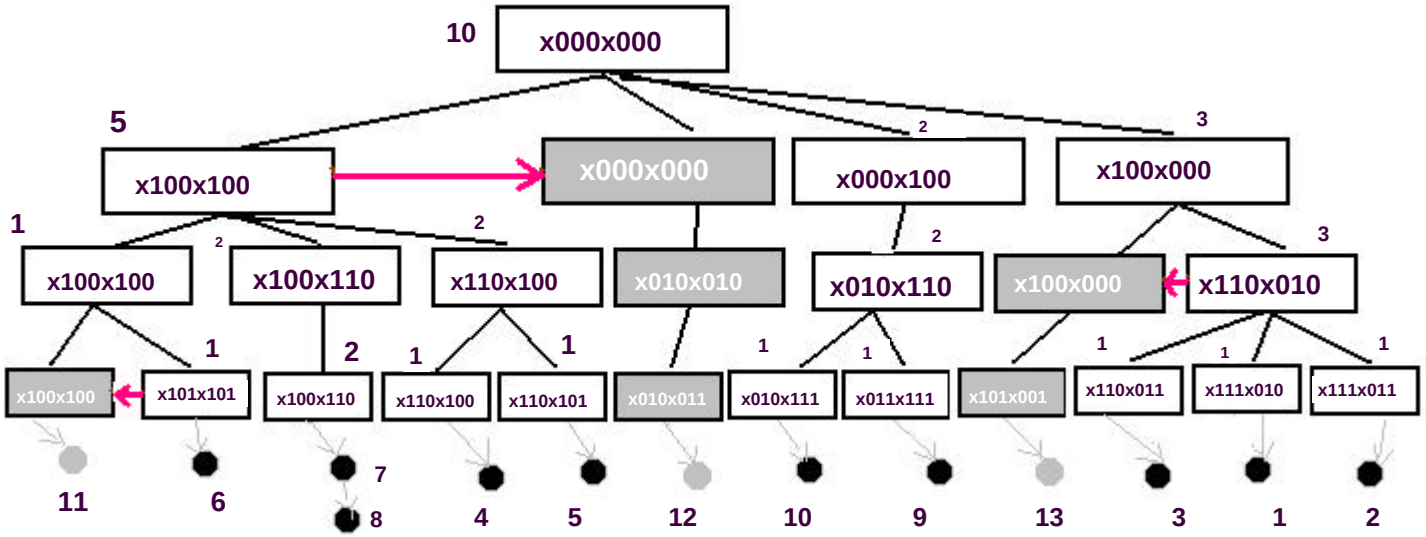


Fig2. Hyper Binary Tree and Pareto Optimal Tree corresponding Fig 1.



Each node in the tree (each grid in the objective space) is given a binary code, which is simply a concatenation of binary strings. There is one such binary string for each objective dimension. The string consists of 1 + DEPTH (division depth) bits, with one bit per division level and an extra Pareto Comparison Bit ('x' in the figure) inserted for efficient Pareto comparison. Initially at level 0, all these bits are set to 0. Later in any division level j, the j'th most significant bit will be set to 1, if the node (grid) is in the upper binary division of the objective space in the considered dimension

The colored arrow in the figure indicates that one node dominates (with granularity of that level) another node in the same level, and therefor also dominates (with some finer granularity) all nodes in the sub-tree of that dominated node.

A Pareto Optimal tree is a hyper binary tree, with all the dominated sub-trees deleted, and represents the current Pareto optimal front regarding the set of all individuals sampled so far.

Also displayed in the figure is resident count for each node of the Pareto tree.

As indicated in the above algorithm, there is one subtle difference between our definition of dominance relation and that of the traditional one: dominating here only holds when all the components are better than those of the dominated. Such a change is necessary because an "equal to" relation in our algorithm is really inconclusive: very likely it will be "better than" or "worse than" as the comparison goes to more deeper level in the tree or as the tree becomes higher as the evolving goes on.

(5) Pareto Binary Tree. If any node in the hyper binary tree is dominated by some node in the same level, all the nodes in the subtree of the dominated node are also dominated by the same dominating node (with some finer granularity). This sub-tree represents an inferior region in the objective space and should be deleted. A Pareto Optimal Tree, or simply Pareto Tree, is a hyper binary tree, with all the dominated subtrees deleted. It represents the current non-dominated (optimal) front regarding all individuals sampled in all generations so far. Be aware that the front is non-dominated or optimal only in the sense of the comparison precision determined by current leaf level.

(6) The Pareto Binary Tree Searching Genetic Algorithm.

```
ParetoTreeSearchingGA {
    Initialize PT (Pareto Tree) to empty.
    While (not termination) {
        Generate n new individuals from population (size n) in PT.
        If n < N, re-initialize N-n new individuals. .
        Updating PT from these N new individuals.
        If hyper rectangular of objective space or PT depth changed, rebuild PT.
        Assign fitness to individuals in PT and reduce population in PT to N if necessary.
    }
}
```

We use only one population, and it is nondominated population. This appears to be too risky of premature convergence due to the lack of population diversity. Two components of this algorithm, however, can deal with this problem effectively. First, when the current nondominated population is smaller than population size, re-initialization is performed. Second, we discard dominated individuals and keep non-dominated ones only with the comparison precision determined by current depth of Pareto tree. The depth of Pareto tree increase adaptively as the evolving goes on, and consequently the comparison granularities will only become much finer at later stage of the algorithm. This way, we can achieve population diversity and computation efficiency at the same time.

(7) Updating Pareto Tree from new individuals. Updating is carried out step by step, from the root level to the leaf level. In each step, each (non-discarded) new individual's node (grid) location is computed incrementally from its previous value in last step (level), and is then compared to nodes already in the current level of Pareto tree. If the new individual is dominated, it is discarded and no more location calculation and node comparison are needed in deeper levels (finer granularities). If dominating, the dominated subtree is deleted. And if it equals to some node already in the tree, the remained comparison in this level is unnecessary and is skipped. Finally, if neither discarded nor equal to some node, a new node is created and inserted into current level of the tree.

Updating_Pareto_Tree_from_new_individuals {
Initialize grid locations of all individuals ***loc[ind]*** to ***0.***
For each level in the PT
For each non-discarded individual ***ind***
For each objective dimension ***i***
If(***objective[ind][i] > half-range[i] + offset[ind][i]***)
Set the relative bit of ***loc[ind]*** to ***1***,
Offset[ind][i] = half-range[i] + offset[ind][i];
Compare to nodes in this level of PT & discard, create node, or delete subtree as appropriate
For each objective dimension ***i*** do
range[i] = half-range[i], half-range[i] /= 2;
}

(8) Fitness assigning, niching and reducing. The hyper tree represents recursively the grid sub-populations of different grid sizes; therefore an accumulated niching strategy is readily implemented. Each node in Pareto tree keeps record of the total number of individuals occupying its sub-tree (sub-region). Fitness assignment, niching, and reducing of Pareto tree size (non-dominated population size) are all related to this resident counts. Two iterations on the Pareto Tree are all the computations needed. One upward iteration to gather global resident information from the local ones, and another downward recursive iteration to distribute global information to local nodes and reduce Pareto tree if necessary. Here, as in SPEA, the greater fitness value an individual has, the worse it is.

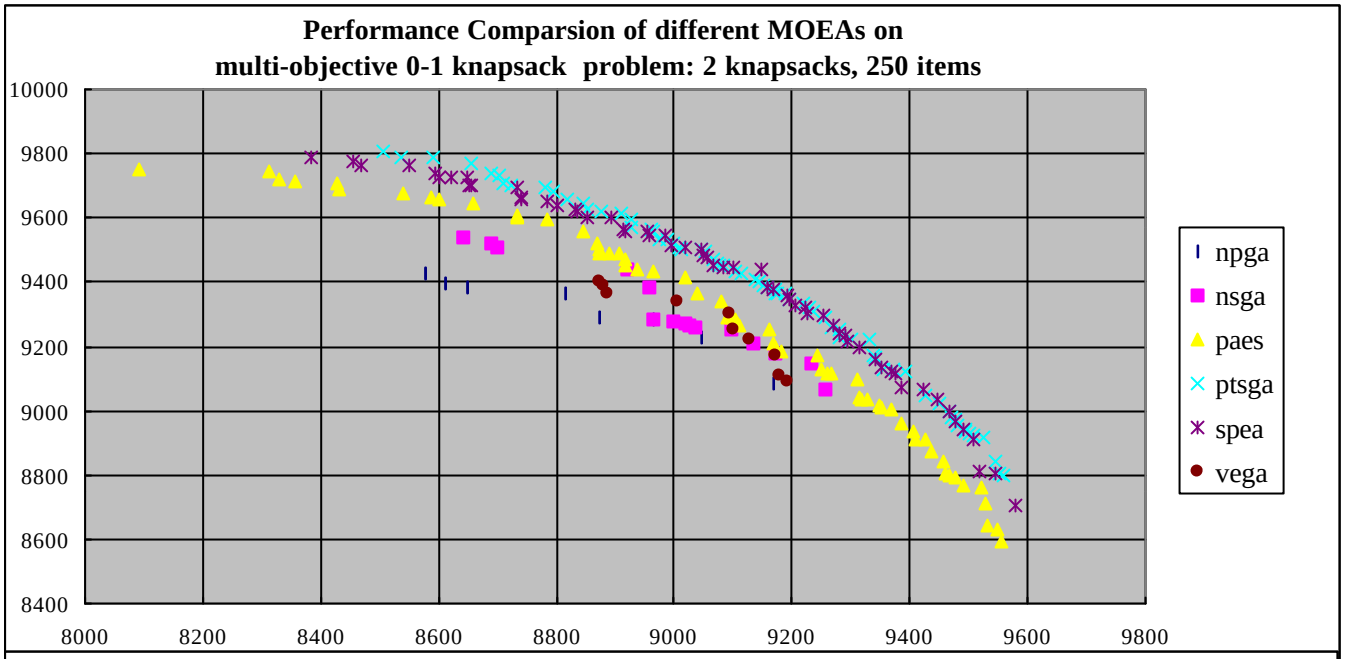
FitnessAssign&Reduction {
Reset all residence ***count*** to ***0*** for all nodes.
Add # of individuals in each leaf node to ***count*** of all its ancestor nodes.
# of individuals should be reduced = count of root node ***– N;*** // reset to 0 if < 0.
Iterate_downward_PT (***# of individuals should be reduced, 0***);
Assign fitness to each individual the ***count*** value of the leaf node it is in.
}

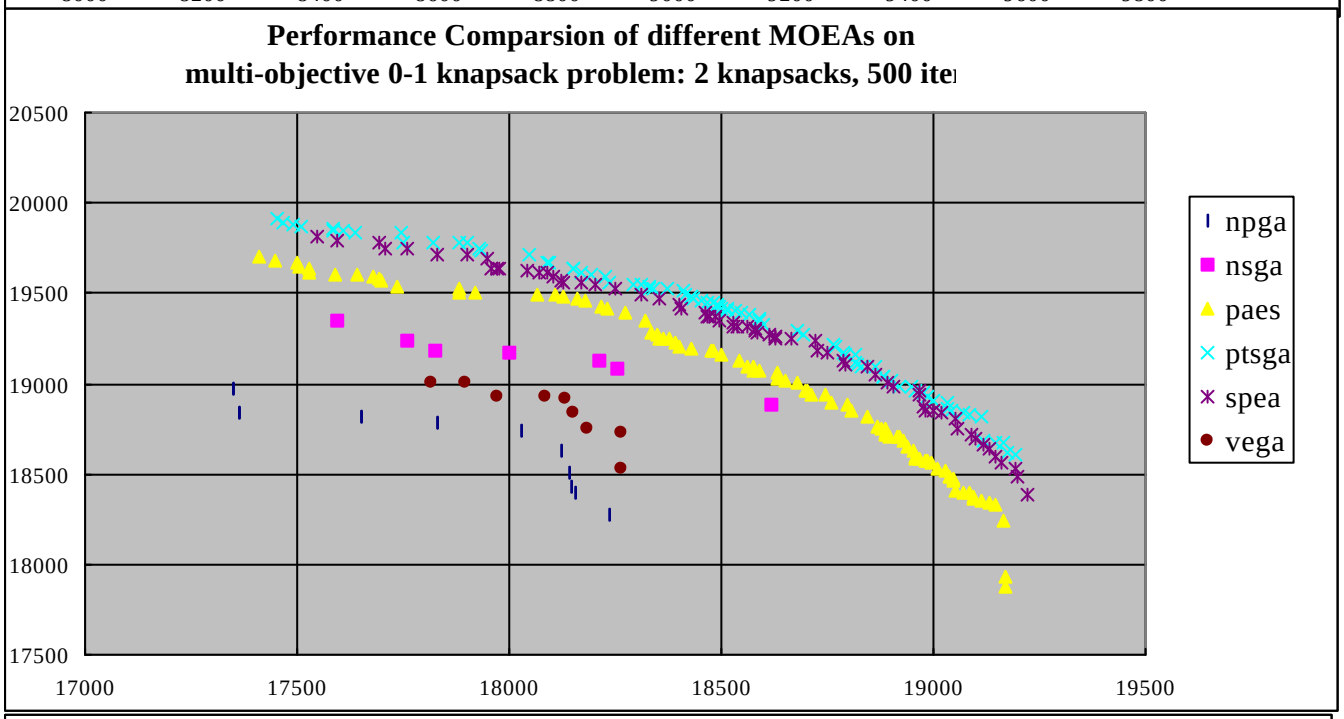
Iterate_downward_PT(*# of individuals should be reduced, residence count of parent*) {
Decrease its residence ***count*** by ***# of individuals should be reduced***
If ***count <= 0***
Delete this subtree, and ***return;***
Else
Add ***resident count of parent*** to its own ***count.***
If leaf node, delete ***# of individuals should be reduced*** individuals, and ***return.***
For each child node do
calculate ***del,*** which is the # of deletion occurred in its subtree.
Iterate_downward_PT(*del, count*)
}

4. Preliminary Experiment Result

For comparison to main current MOEAs, we have carried out experiments on multi-objective 0-1 knapsack problems described in Zitzler (1999). The parameters of all the random runs are the same as that of Zitzler (1999) except PAES and PTSGA. In all the runs of PAES, the total iterations is set to the

Fig 3. Resulting non-dominated fronts for 2 knapsacks problems





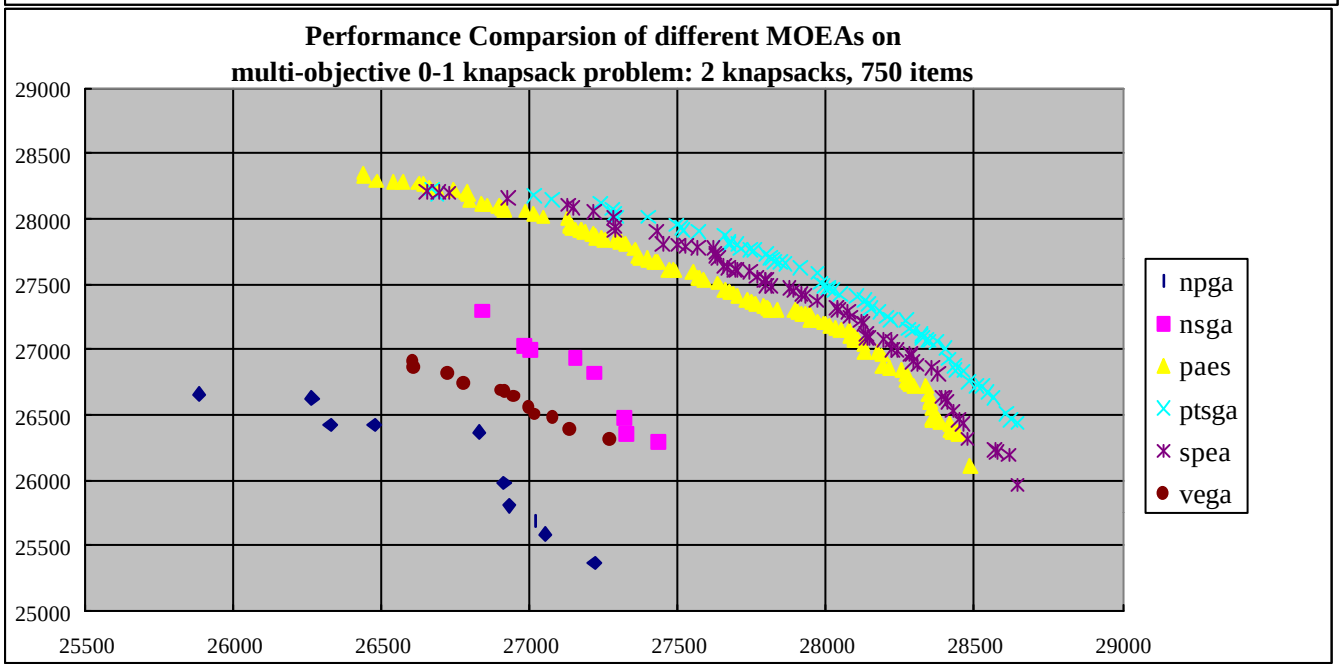
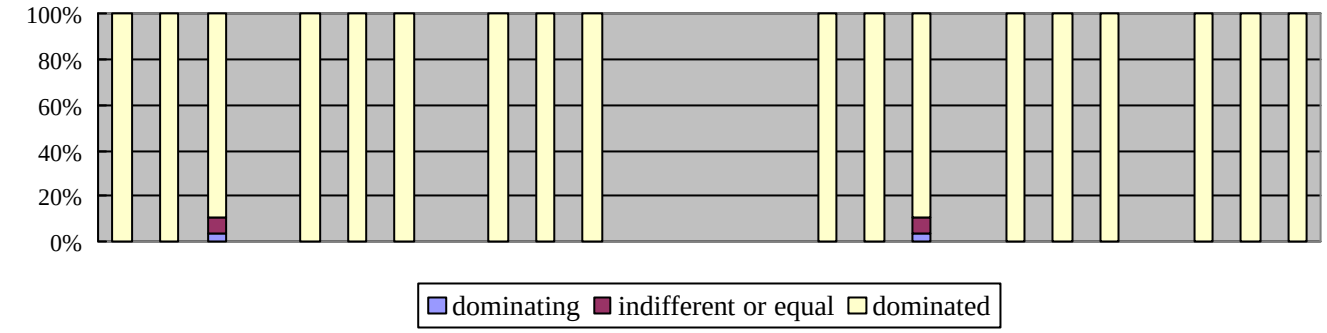
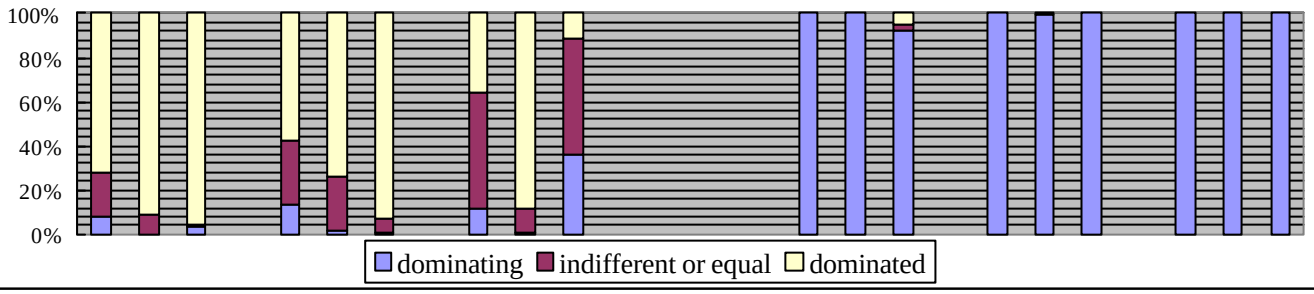


Fig 4. Coverage and Attainment Comparisons for knapsack problems

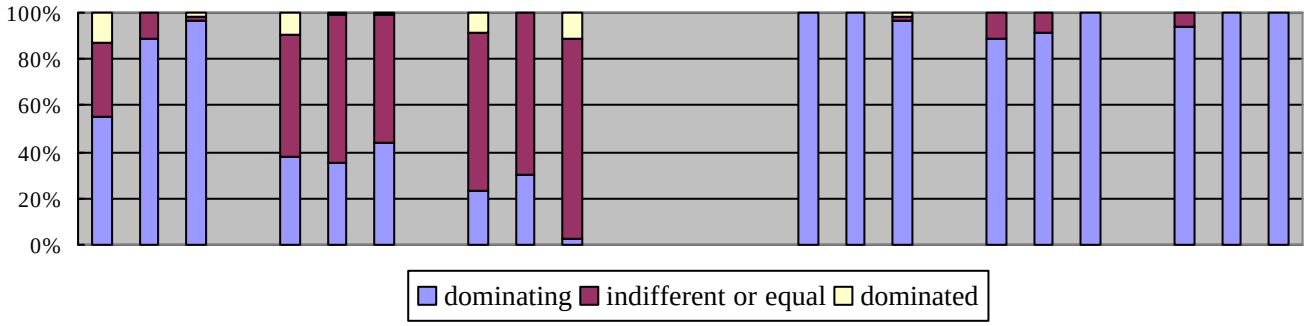
coverage: PAES to PTSGA (left half) & SPEA (right half). Each half consists of knapsack problems with objectives and items of (2, [250,500,750]), (3, [250,500,750]), (4, [250,500,750]) from left to right.



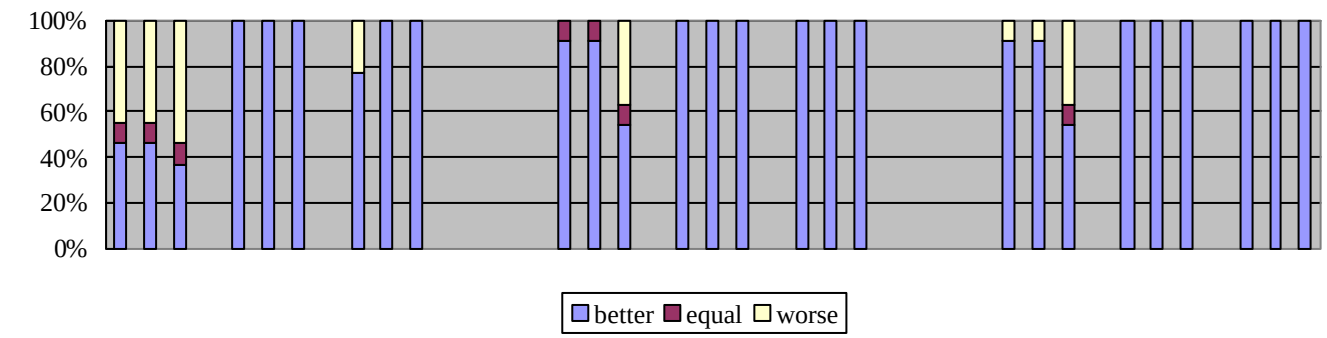
coverage: SPEA to PTSGA (left half) & PAES (right half). Each half consists of knapsack problems with objectives and items of (2, [250,500,750]), (3, [250,500,750]), (4, [250,500,750]) left to right.



coverage: PTSGA to SPEA (left half) & PAES (right half) Each half consists of knapsack problems with objectives and items of (2, [250,500,750]), (3, [250,500,750]), (4, [250,500,750]) from left to right.



attainment comparsions between pairs of algorithms:(from left to righ)
PTSGA vs.SPEA, PTSGA vs.PAES, SPEA vs.PAES on knapsack problems with knapsacks and items of (2, [250,500,750]), (3, [250,500,750]), (4, [250,500,750])



product of population size and maximum generations of other MOEAs, archive size is set to the 2/5 of the population size, and probability of mutation is set to 0.002. Parameters specific to Pareto Tree Searching GA are as follows:

- Population size is set to 2/3 of the value given to other algorithms.
- Generations are set to 3/2 of the value given to other algorithms.
- Probability of mutation is set to 0.004.
- Tree depth is increased from initial value of 3 to maximal 15, after every step-size generations.
- Step-size is set to 35 initially and will be multiplied by 1.1 each time tree depth increased.

We use both the coverage statistics, similar to Zitzler (1999), and attainment statistics, similar to Fonseca and Flemming (1995a), to compare the experimental data of different algorithms. Five random runs are performed for each algorithm on each problem, and they are merged together and culled to be non-dominated before the coverage or attainment comparison is made. We use a total *10* (objective dimensions – 1)* lines to calculate the attainment comparison.

For the three 2-objectives knapsack problems, with respective items of 250, 500, and 750, the relative performances of all six MOEAs are displayed in Fig 3. The six algorithms are:

- npga Niched Pareto Genetic Algorithm.
- nsga Non-dominated Sorting GA.
- paes Pareto Archived ES.
- ptsga Pareto Tree Searching GA
- spea Strength Pareto Evolutionary Algorithm
- vega Vector Evaluated GA (Shaffer, 1984)

As the performances of the three best algorithms, paes, ptsga, and spea, are significantly better than all other algorithms, the comparison results of coverage and attainment statistics are displayed in Fig 4 only for these three algorithms. For each pair of algorithms A and B, we define the following:

dominating (A, B) = solutions in A which dominate any solutions in B / total solutions in A.
dominated (A, B) = solutions in A which are dominated by any solutions in B / total solutions in A.
Indif_or_equal (A, B) = 1 – dominated (A, B) – dominating (A, B).

better (A, B) = lines where A's attainment is better than that of B's / total lines.
worse (A, B) = lines where A's attainment is worse than that of B's / total lines.
equal (A, B) = lines where A's attainment is equal to that of B's / total lines.

We use the stacked bar to display the relative performance of any pair of algorithm on a certain problem.

Though there seems to be a performance hierarchy of ptsga, spea, and paes in descending performance order, it is really inconclusive, as the experiment is very preliminary; more accurate statistical methods, as well as more complete comparisons on other standard test functions, are required. These works will come later.

Bibiliography

Carlos A. Coello Coello(1999). *A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques*, Knowledge and Information Systems. An International Journal, 1(3):269-308, August 1999 .

Carlos A. Coello(1999a). *An Updated Survey of Evolutionary Multiobjective Optimization Techniques :State of the Art and Future Trends*, In 1999 Congress on Evolutionary Computation, pages 3-13, Washington, D.C., July 1999. IEEE Service Center.

Fonseca, C. M. and P. J. Fleming (1993). *Genetic algorithms for multiobjec-tiveoptimization:*

Formulation, discussion and generalization. In S. For-rest(Ed.), Proceedings of the Fifth International Conference on GeneticAlgorithms, San Mateo, California, pp. 416–423. Morgan Kaufmann.

Fonseca, C. M. and P. J. Fleming (1995). *An overview of evolutionary al-gorithmsin multiobjective optimization.* Evolutionary Computation 3(1), 1–16.

Fonseca, C.M. and P. J. Fleming (1995a). *On the performance Assessment and Comparison of Stochastic Multiobjective Optimizers*. In Voigt, H-M, Ebeling, W., Rechengerg, I. and Schwefel, H-P., editors, *Parallel Problem Solving From Nature- PPSN IV*, pages 584-593, Springer.

Horn, J. and N. Nafpliotis (1993). *Multiobjective optimization using theniched pareto genetic algorithm.* IlliGAL Report 93005, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, Champaign.

Horn, J., N. Nafpliotis, and D. E. Goldberg (1994). *A niched pareto genetic algorithm for multiobjective optimization.* In Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress

Knowles, J. and D. Corne (1999a). *Approximating the non-dominated front using the pareto archived evolution strategy*. Technical Report RUCS/1999/TR/005/A, Department of Computer Science, University of Reading, UK, March.

Schaffer, J.D.(1984). *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.* Ph. D. Thesis, Vanderbilt University. Unpublished.

Srinivas, N. and K. Deb (1994). *Multiobjective optimization using non-dominated sorting in genetic algorithms.* Evolutionary Computation 2(3),221–248.

David A. Van Veldhuizen and Gary B. Lamont(2000). *Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art*, Evolutionary Computation, 8(2):125-147, 2000.

Eckart Zitzler and Lothar Thiele(1999). *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach*, IEEE Transactions on Evolutionary Computation, Vol. 3, No. 4, pp. 257-271, November, 1999

Eckart Zitzler(1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications* PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.

Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele(1999). *Comparison of Multiobjective Evolutionary Algorithms on Test Functions of Different Difficulty,* In Annie S. Wu, editor, Proceedings of the 1999 Genetic and Evolutionary Computation Conference. Workshop Program, pages 121-122, Orlando, Florida, July 1999 .