A Simple yet Efficient Multiobjective Combinatorial Optimization Method Using Decomposition and Pareto Local Search

Liangjun Ke, and Qingfu Zhang, *Senior Member, IEEE*, and Roberto Battiti, *Fellow, IEEE*

***Abstract*—Combining ideas from evolutionary algorithms, decomposition approaches and Pareto local search, this paper suggests a simple yet efficient memetic algorithm for combinatorial multiobjective optimization problems: MoMad. It decomposes a combinatorial multiobjective problem into a number of single objective optimization problems using an aggregation method. MoMad evolves three populations: population P_L for recording the current solution to each subproblem, population P_P for storing starting solutions for Pareto local search, and an external population P_E for maintaining all the nondominated solutions found so far during the search. A problem-specific single objective heuristic can be applied to these subproblems to initialize the three populations. At each generation, a Pareto local search method is first applied to search a neighborhood of each solution in P_P to update P_L and P_E. Then a single objective local search is applied to each perturbed solution in P_L for improving P_L and P_E, and re-initializing P_P. The procedure is repeated until a stopping condition is met. MoMad provides a generic hybrid algorithmic framework to use problem specific knowledge and employ well developed single objective local search and heuristics, and Pareto local search methods for dealing with combinatorial multiobjective problems. It is a population based iteration method and thus an anytime algorithm. Extensive experiments have been conducted in this paper to study MoMad and compare it with some other state of the art algorithms on the multiobjective traveling salesman problem and the multiobjective knapsack problem. The experimental results show that our proposed algorithm outperforms or performs similarly to the best so far heuristics on these two problems.**

***Index Terms*—Multiobjective optimization, multiobjective traveling salesman problem, multiobjective 0-1 knapsack problem, decomposition, Pareto local search.**

I. Introduction

A generic *multiobjective optimization problem* (MOP) can be stated as follows:

$$\begin{aligned} &\text{maximize} \quad F(x) = (f_1(x), \ldots, f_m(x)) \\ &\text{subject to} \quad x \in \Omega \end{aligned} \tag{1}$$

where Ω is the *decision space*, $F : \Omega \rightarrow R^m$ consists of m real-valued objective functions. The *attainable objective set* is $\{F(x)|x \in \Omega\}$.

L. Ke is with the State Key Laboratory for Manufacturing Systems Engineering, Xian Jiaotong University, Xi'an, 710049, China. Email: keljxjtu@xjtu.edu.cn.

Q. Zhang is with the School of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, U.K. E-mail: qzhang@essex.ac.uk.

R. Battiti is with the Department of Information Engineering and Computer Science, University of Trento, I-38123 Povo (Trento), Italy. Email: battiti@disi.unitn.it.

Let $u, v \in R^m$, u is said to *dominate* v, denoted by $u \succ v$, if and only if $u_i \geq v_i$ for every $i \in \{1, \ldots, m\}$ and $u_j > v_j$ for at least one index $j \in \{1, \ldots, m\}$[1]. Given a set S in R^m, a point in it is called nondominated in S if no other point in S can dominate it. A point $x^* \in \Omega$ is *Pareto-optimal* if it is nondominated in the attainable objective set. $F(x^*)$ is then called a *Pareto-optimal (objective) vector*. In other words, any improvement in one objective of a Pareto optimal point must lead to deterioration to at least another objective. The set of all the Pareto-optimal points is called the *Pareto set* (*PS*) and the set of all the Pareto-optimal objective vectors is the *Pareto front* (*PF*) [1]. In many real life applications, the PF is of great interest to decision makers for understanding the tradeoff nature of different objectives and selecting their final solutions.

If Ω is a finite set, then (1) is called a combinatorial MOP. Finding the exact PF of a real world combinatorial MOP is often NP-hard by nature even though its single objective counterpart is not [2]. For this reason, heuristics, particularly, hybrid heuristics, have been widely used and studied for approximating the PFs of combinatorial MOPs. Elements and ideas from many different single objective heuristics such as variable neighborhood search [3], tabu search [4], iterative local search [5], guided local search [6], simulated annealing [7] and ant colony optimization [8] have been generalized to combinatorial multiobjective optimization. In most hybrid heuristics developed over the last two decades, the following three highly related basic techniques are frequently employed:

- *Pareto Local Search* (*PLS*): It is a natural extension of single objective local search methods [9]–[11]. PLS works with a set of mutually nondominated solutions. It explores some or all of the neighbors of these solutions to find new solutions for updating this set at each iteration. Several variants of PLS have been proposed and investigated on various problems (e.g., [12], [13]).
- *Aggregation*: An MOP can be transformed into a single objective optimization problem by linearly or nonlinearly aggregating all the individual objectives $f_1, \ldots, f_M$ into one single objective. Under mild conditions, an optimal solution to this single objective problem is Pareto-optimal to (1). Thus, a single objective optimization method can be used for finding a set of Pareto optimal solutions by solving a set of such single objective problems with different weights [2].

[1]In the case of minimization, the inequality signs should be reversed.

- *Multiobjective Evolutionary Algorithms* (*MOEA*): A typical MOEA evolves a working population (i.e., on-line population) of a prefixed size by using selection and reproduction operators. An MOEA selection operator can be based on Pareto dominance [14], aggregation [15] or some performance indicators [16]. A second population (i.e., archive), is often employed in MOEAs to store nondominated solutions generated during the evolutionary process.

One of the most successful and simplest hybrid multiobjective heuristic frameworks is the two phase Pareto local search (2PPLS) [11]. It combines PLS and aggregation. As its name suggests, 2PPLS consists of two phases. Phase 1 generates an approximation to the set of all the efficient supported solutions (their definition will be given in Section II) by solving a number of linear aggregation problems. Phase 2 applies a PLS to every solution generated in Phase 1 to find non-supported Pareto optimal solutions. Since it has only two phases, it is necessary that the initial solutions provided by Phase 1 are of high quality and as many as possible. Therefore, as pointed out in [11], Phase 1 often incurs very heavy computational overheads. For this reason, 2PPLS is not very suitable for dealing with more than three objectives. Another disadvantage is that it has no mechanism to escape from local optimal solutions and it can stop even when there is still computational time available. Arguably, a very natural way for overcoming these disadvantages is to implement the 2PPLS idea in an iterative framework.

Multiobjective optimization evolutionary (MOEA/D) [17] is an MOEA using aggregation ideas. It decomposes an MOP into a number of single objective subproblems by aggregation. Solutions to these different subproblems are evolved and improved in a collaborative way. Single objective optimization methods can be easily employed in the MOEA/D framework. Since MOEA/D is an iterative algorithm, any amount of computational time can be, at least in principle, used to improve the quality of its solutions. The MOEA/D algorithmic framework has been adopted in a number of MOEAs (e.g. [18], [19]).

This paper combines the ideas from 2PPLS and MOEA/D and proposes a **M**ulti**o**bjective **M**emetic **a**lgorithm based on **d**ecomposition (MoMad). **Liangjun ask: Do we need some words on: Memetic algorithm is ...** It is simple yet efficient and makes use of all the three basic techniques mentioned earlier in this section. MoMad maintains three populations: Each solution in the first population (denoted by P_L in this paper) is associated with a weighted sum aggregation subproblem. The second population (denoted by P_P) stores some very promising solutions selected from P_L. The third population P_E is an achieve for recording all the nondominated solutions found so far. A problem-specific single objective heuristic can be applied to these subproblems to initialize the three populations. At each generation, a Pareto local search method is first applied to search a neighborhood of each solution in P_P to update P_L and P_E. Then a single objective local search applied to each perturbed solution in P_L for improving P_L and P_E, and re- initializing P_P. The procedure is repeated until a stopping condition is met. Extensive experiments have been conducted in this paper to study MoMad and compares it with some other state of the art algorithms on the multiobjective traveling salesman problem and the multiobjective knapsack problem. The experimental results show that our proposed algorithm outperforms or performs similarly to the best so far heuristics on these two problems.

The rest of the paper is organized as follows. Section II presents all the details of our proposed MoMad. Section III gives an implementation of MoMad and compare it with the two phase Pareto local search method suggested in [9]–[11] on the biobjective travelling salesman problems. Experimental studies on the effects of some control parameters on the performance of MoMad are also conducted in this section. Section IV implements MoMad for the multiobjective knapsack problem and compare it with three other algorithms. Section V concludes this paper and outlines some avenues for future research.

II. Main Techniques

This section introduces the major ideas and techniques used in our proposed algorithm.

A. Decomposition and algorithmic framework

A widely used multiobjective optimization methodology in traditional mathematical programming community is to cast an MOP into a single objective optimization problem by (linearly or nonlinearly) aggregating all the individual objective $f_1, \ldots, f_m$ with a weight coefficient vector. To obtain an approximation to the PF, one can solve a set of such single objective problems with carefully selected weight coefficient vectors. This idea has been successfully used in some MOEAs such as MOGLS [20] and MOEA/D [17].

There are a number of aggregation approaches for decomposing an MOP into a number of single objective optimization subproblems. For simplicity, this paper uses the weighted sum approach. It considers the following single objective optimization problem:

$$\begin{array}{ll} \text{maximize} & g^{ws}(x|\lambda) = \sum_{i=1}^{m} \lambda_i f_i(x) \\ \text{subject to} & x \in \Omega \end{array} \tag{2}$$

where $\lambda = (\lambda_1, \ldots, \lambda_m)$ is a weight vector with $\sum_{i=1}^{m} \lambda_i = 1$ and $\lambda_i \geq 0$ for all $i = 1, \ldots, m$.

For any nonnegative weight vector, (2) has an optimal solution which is Pareto optimal to (1). In the case of positive weight vectors, any optimal solutions to (2) are Pareto optimal to (1) [2]. An Pareto optimal solution is called supported efficient if it is an optimal solution to (2) with a particular weight vector. Let X_{se} and PF_{se} be the set of all the supported efficient solutions and the set of their corresponding Pareto optimal objective vectors, respectively, it is well-known that $PF_{se} = PF$ when PF is concave. In the case of nonconcave *PF*s, although $PF_{se} \neq PF$ as illustrated in Fig. 1, a basic assumption in this paper is that, in an appropriately defined neighborhood of a locally or globally optimal solution to (2), more Pareto optimal solutions can be found with a reasonable amount of computational effort. This assumption, illustrated

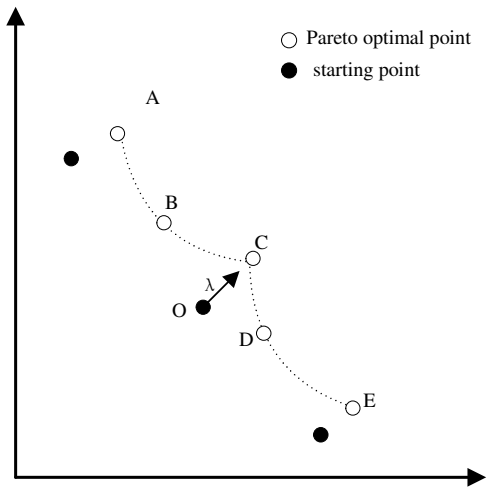

Fig. 1. The demo of the assumption. In this figure, PF is $\{A, B, C, D, E\}$, while PF_{se} is $\{A, C, E\}$. Although the starting point O is better than D with respect to the objective of subproblem with weight λ, it is more likely to yield D from O.

is Fig. 2, has been used in several successful metaheuristics [9] including MOGLS and 2PPLS [11].

Mutiobjective Memetic Algorithm based on Decomposition (MoMad), the algorithm proposed in this paper, adopts the MOEA/D framework. More specifically, MoMad first selects N weight vectors $\lambda^1, \ldots, \lambda^N$ and forms N single objective optimization subproblems. Subproblem k is defined by (2) with $\lambda = \lambda^k$. At each generation, MoMad maintains three populations:

- $P_L = \{x^1, \ldots, x^N\}$: x^k is the current candidate solution to subproblem k. The solutions in this population will be undergone perturbation and single objective local search operators.
- P_P: the solutions in this section will be undergone PLS.
- P_E: it is an external population for storing nondominated solutions found so far.

The MoMad works as follows:

Step 0: **Initialization:**

0.1 **Initialization of P_L:** For each $k = 1, \ldots, N$, apply a single objective heuristic on subproblem k to generate solution x^k. All these solutions form P_L.

0.2 **Initialization of P_P and P_E:** Find all the nondominated solutions in P_L and let them constitute P_P. Initialize $P_E := P_P$.

Step 1 : If a preset stopping condition is met, stop and return P_E. Otherwise, go to Step 2.

Step 2 :**Pareto Local Search:** Conduct Pareto local search on P_P to update P_L and P_E. Then set $P_P =: \emptyset$ (The details of PLS can be found in Subsection II-B).

Step 3 : For each $k = 1, \ldots, N$

3.1 **Perturbation:** apply a perturbation operator on x^k in P_L to generate $\tilde{x}^k$.

3.2 **Local Search:** apply a single objective local search for subproblem k on $\tilde{x}^k$ to generate a new solution y^k.

3.3 **Updating P_L:** $P_L = Update1(x', P_L)$.

3.4 **Updating P_E:** $P_E = Update2(x', P_E)$.

3.5 **Updating P_P:** If x' has been added to P_E in 3.4, $P_P = Update2(x', P_P)$.

End of for

Go to Step 1.

Algorithm 1: $P_L = Update1(x', P_L)$

input : $P_L = \{x^1, \ldots, x^N\}$, x'.
output: $P_L = \{x^1, \ldots, x^N\}$.

```
Replacement = false,
P_α := P_L.
while (P_α ≠ ∅)&(Replacement = false) do
    Randomly select a solution x^k from P_α.
    P_α := P_α \ {x^k}.
    if g^ws(x^k|λ^k) < g^ws(x'|λ^k) then
        x^k := x', Replacement := true.
    end
end
```

Algorithm 2: $P = Update2(x', P)$

input : P, x'.
output: P.

```
if There is no solution in P which can dominate x' then
    Remove all the solutions in P which are dominated
    by x'
    Add x' to P.
end
```

In Step 0, the three populations are initialized. Since solution x^k in P_L is for single objective subproblem k, it can use well-developed single objective heuristics to obtain initial x^k. Then, based on Pareto dominance, it compares the solutions

in P_L to find all the nondominated solutions in P_L which are then used for initializing P_E and P_P.

Step 2 conducts PLS to explore the neighborhoods of the solutions in P_P to update P_L and P_E. After this step, P_P will be set to be an empty set. The purpose of this step is to find those Pareto optimal solutions that may not be found by Perturbation and Local Search in Step 3. A major computational overhead of MoMad comes from this step.

Step 3.1 is to drive the search into a new region. The perturbation should be big enough such that the generated solution $\tilde{x}^k$ is out of the attractive basin of current x^k. Too big perturbation, however, may take $\tilde{x}^k$ to an unpromising search area. y^k, generated in Step 3.2 by a single objective local search on $\tilde{x}^k$, could be a better solution to subproblem k or other subproblems. Steps 3.3−3.5 use y^k to update P_L, P_E and P_P. P_P contains all the new nondominated solutions generated in Step 3, and it is a subset of P_L.

In $Update1(x', P_L)$ (Algorithm 1), its inputs are x' and P_L. As mentioned earlier, each solution x^k in P_L is the current solution to subproblem k. $Update1(x', P_L)$ compares x' and solutions in P_L in a random order based on their aggregated objective function values. Due to the use of flag variable $Replacement$, x' is allowed to replace only one solution in P_L, which is useful for maintaining the diversity of P_L.

In $Update2(x', P)$ (Algorithm 2), x' is compared with every solution in P, it will be added to P if there is no solution in P which can dominate it. All the solutions in P which are dominated by x' will be removed from P.

We would like to make the following comments on the above algorithmic framework:

- Since each solution x^i in P_L is for single objective subproblem i, single objective optimization techniques can be used in Step 0.1 and Step 3.2. A lot of effective approaches have been developed and studied in single objective combinatorial optimization for utilizing problem-special knowledge [21]. Thus, this framework provides a natural way for using these working experiences in combinatorial multiobjective optimization.
- In the existing two phase local search methods such as 2PPLS [11] and PD-PLS [9], each subproblem undergoes a single objective local search only once for generating one supported efficient solution. In contract, MoMad adopts an iterative fashion and applies a local search precedure to every subproblem many times with different starting solutions during the search. Therefore, with Pareto local search, MoMad can explore the neighborhoods of different locally optimal solutions to each subproblem. The motivation behind the use of iteration in MoMad is supported by the assumption illustrated in Fig. 1.
- In some other MOEA/D variants (e.g., [22]–[24]), a neighborhood relationship among all the subproblems is defined based on the distances among their weight vectors. Correspondingly, a neighborhood relationship among all the current solutions can be established. These MOEA/D variants update a current solution mainly by offspring of its neighboring solutions. This is based on the assumption that two neighboring subproblems have similar optimal solutions. Although this assumption is often true in combinatorial multiobjective optimization, some exceptions can be observed (e.g. [18]). For this reason, MoMAD does not employ the neighborhood concept, a new solution has a chance to replace any other current solutions in P_L.
- A major computational overhead of MoMad is incurred by PLS. To avoid unbearable computational costs, P_P is set to be empty at the end of Step 2. A solution is added to P_P in Step 3 only if it has entered into P_E. As a result, P_P contains all the new nondominated solutions generated in Step 3 and it is a subset of P_L. Therefore, the size of P_P cannot exceed N and neither can the number of PLS **Liangjun: calls ???** at each generation.

B. Pareto Local Search

MoMad conducts PLS in Step 2. Several different versions of PLS have been developed and studied during the last several years. The PLS algorithm used in our experimental studies is a modified version of Pareto local search proposed in [11]. Its detail is given in Algorithm 3.

Algorithm 3: Pareto Local Search

```
   input : P_P, P_L, P_E, Max.
   output: P_L, P_E.
1  j := 0.
2  while (P_P ≠ ∅)&(j++ < Max) do
3      P_α := ∅.
4      for each x ∈ P_P do
5          for each x' ∈ N(x) do
6              update1(x', P_L)
7              if F(x) ≺ F(x') then
8                  update2(x', P_E)
9                  if x' has been added to P_E in Line 8 then
10                     update2(x', P_α)
11                 end
12             end
13         end
14     end
15     P_P := P_α
16 end
```

In this PLS algorithm, Max is the maximal number of passes in the while loop (Line 2 to 16). It is used to bound the computational overheads. In line 1, j, the index of the current loop pass, is initialized to be 0. P_α, which is initialized to be empty in Line 3, is to reset P_P at the end of each while pass in Line 15. For each x' in the neighborhood of every x in P_L, $Update1(x', P_L)$ is executed in Line 6 to update P_L. If x' dominates x, $update2(x', P_E)$ is executed to update P_E in Line 8. If x' has entered P_E, then Line 10 executes $update1(x', P_\alpha)$ to update P_α. The two if conditions in Lines 7 and 8 only allow very promising solutions to update P_E and P_L. In such a way, the computational overheads can be significantly reduced.

III. Comparison of MoMad with 2PPLS on Multiobjective Traveling Salesman Problem

A. Problem

Given n cities with edges connecting any two cities. Suppose that edge e has m distance metrics $d_{e,1}, \ldots, d_{e,m}$. Each feasible solution is an edge subset which can form a Hamiltonian cycle. The i-th objective function to minimize is:

$$f_i(x) = \sum_{e \in x} d_{e,i} \tag{3}$$

In our experiments, we use biobjective test instances collected by T. Lust[2], which have been used to test 2PPLS.

2PPLS [11] is the best-so-far heuristic algorithm for the biobjective TSP instances used in our experimental studies. It consists of two phases. Its first phase generates a good approximation of the supported efficient solutions. The second phase uses the Pareto local search for generating non-supported solutions. The experimental results of 2PPLS used in our comparison are from the author's web.

B. Implementation of MoMad

1) Initialization of P_L (Step 0.1): each subproblem k in the mTSP is a single objective TSP with the following objective:

$$\sum_{e \in x} (\sum_{i=1}^{m} \lambda_i^k d_{e,i}) \tag{4}$$

We use the chained Lin-Kernighan heuristic [25] on subproblem k to generate x^k in our experiments.

2) Candidate List: Our implementation of PLS (Step 2) and LS (Step 3.2) requires a candidate edge (CE) list. The CE list consists of all the edges of the solutions in the current P_P. It will be updated once P_P has been updated.

3) $N(x)$ in Pareto local search (Step 2) and local search (Step 3.2) : Given a feasible solution x, y is a neighbor of x if y is feasible and can be obtained from x in the following way:

1) remove two nonadjacent edges from x, and
2) add two new edges to x to form a cycle, where at least one added edge is from the candidate list.

$N(x)$ is then the set of all the neighbors of x.

4) Perturbation (Step 3.1): Let x^k in P_L be the current solution to subproblem k, we apply a random double bridge move (i.e., 4-interchange) on x^k to generate $\tilde{x}^k$. A double bridge move is illustrated in Fig. 2.

5) Local Search (Step 3.2): To do local search on $\tilde{x}^k$ for subproblem k, we find the solution z from $N(\tilde{x}^k)$ (the neighbourhood definition is the same as in PLS) with the highest aggregated objective function $f^{ws}(x|\lambda^k)$ value. If z is better than $\tilde{x}^k$ in terms of $f^{ws}(x|\lambda^k)$, then we replace $\tilde{x}^k$ by z and repeat this procedure, otherwise, we stop and output $\tilde{x}^k$ as y^k.

[2]http://sites.google.com/site/thibautlust/research/multiobjective-tsp

C. Performance Metrics

- **Hypervolume indicator** (I_H) [26]: Let $y^* = (y_1^*, \ldots, y_m^*)$ be a point in the objective space which is dominated by any Pareto optimal objective vectors. Let S be the obtained approximation to the PF in the objective space. Then the I_H value of S (with regard to y^*) is the volume of the region which is dominated by S and dominates y^*. The higher the hypervolume, the better the approximation.
- **Set Coverage (C-metric) [26]:** Let A and B be two approximations to the PF of an MOP, $C(A, B)$ is defined as the percentage of the solutions in B that are dominated by at least one solution in A, i.e.,

$$C(A, B) = \frac{|\{u \in B | \exists v \in A : v \text{ dominates } u\}|}{|B|}$$

$C(A, B)$ is not necessarily equal to $1 - C(B, A)$. $C(A, B) = 1$ means that all solutions in B are dominated by some solutions in A, while $C(A, B) = 0$ implies that no solution in B is dominated by a solution in A.

D. Experimental Parameter Setup

1) The Size of P_L (N) and Weight Vectors (λ) in MoMad: Too large N will result in very high computational costs at each generation, while too small N may make the search miss some parts of the PF. Based on these considerations, N is set as follows:

- $N = n$ when $n = 200, 300, 400, 500$, where n is the number of cities.
- $N = 600$ when $n = 750, 1000$.

λ^k ($k = 1, \ldots, N$) is set as:

$$\lambda^k = \left(\frac{k-1}{N-1}, \frac{N-k}{N-1} \right) \tag{5}$$

2) Max in PLS: Max is set to be 10 in our experiments.

3) The Stopping Condition: The algorithm stops after 500 generations.

4) The number of Independent Runs: The algorithm has been run 20 times on each test instance.

E. Results

The final solution sets of 20 independent runs of 2PPLS reported in its author's web are used for comparison. On each test instance, each final nondominated solution set obtained by our algorithm is compared with every solution set by 2PPLS. To compute the hypervolumes of two algorithms on each instance , y_i^* ($i = 1, \ldots, m$) is set to be the largest value of the i-th objective among all the final solutions generated by both 2PPLS and MoMad. The hypervolume values of the final solution sets produced by the two algorithms are plotted in Fig. 3. We compute the mean C-metric values of all the 20×20 comparisons on each instance and report them in Table I. The average running times consumed by the two algorithms are given in Table II.

From these results, we can make the following remarks:

- In terms of the coverage metric, MoMad outperforms 2P-PLS on all the test instances. For example, on Kroab1000,

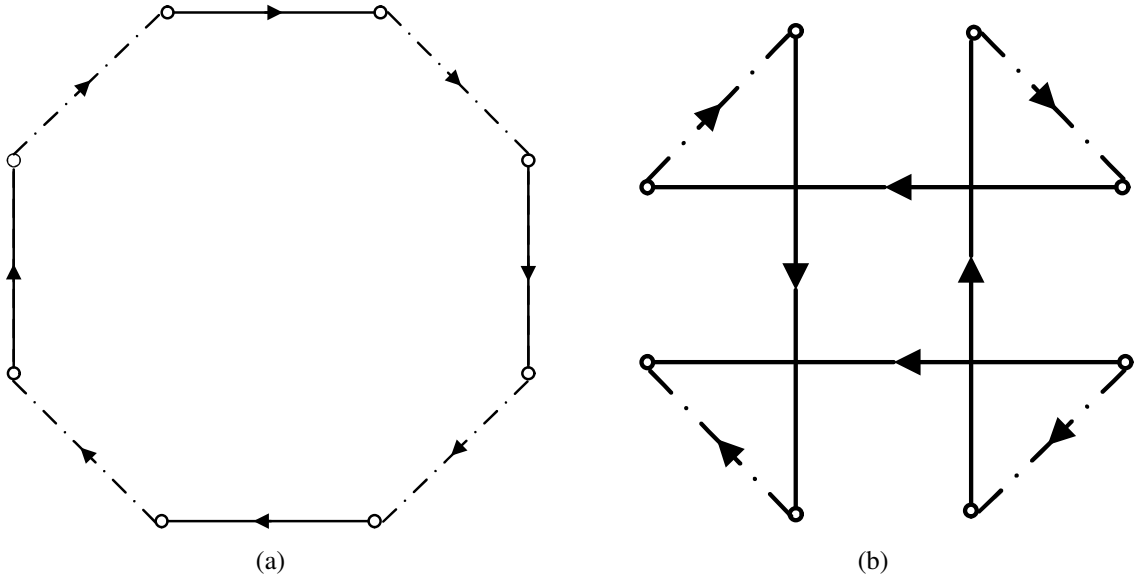

Fig. 2. (a) the solution to be perturbed. (b) the resultant solution. In this figure, a random double bridge move first assigns a direction to the circle formed by x^k, randomly remove four nonadjacent edges from the circle, and then add four new edges to form a new circle without changing the direction.

the instance with 1,000 cities, a solution set produced by MoMad can on average dominate 60.4% of a solution set by 2PPLS, and 38.2% vice versa.

- On 16 out of 18 test instances, the hypervolume metric values shown in Fig. 3 are consistent with the coverage metric values, which further confirms the competitiveness of MoMad. On rdAB300 and rdAB500, the two exception instances, we have tested MoMad with larger population sizes and found that MOEAD can perform better with regards to the hypervolume metric as shown in Fig. 4.
- Regarding the running time, two algorithms were tested on different but similar computing environments. The statistics in Table II indicate that two algorithms have consumed about the same amount of running time on each test instance. We also want to emphasize that unlike 2PPLS, MoMad is an anytime algorithm. As shown in the next section, MoMad can find better solutions if more computational resources are allocated to it.

F. More Discussions

MoMad has two control parameters: N and Max. We have experimentally investigate the effects of the two control parameters on the algorithm performance. We have taken Kroab200 as a test instance in our experimental studies and each parameter configuration has been tested on 20 independent runs.

1) The Influence of The Number of Subproblems N: N is also the size of P_L, which thus decides how many times perturbation and single objective local search operators are conducted at Steps 3.1 and 3.2 in each generation. We have tested MoMad with $N = 100, 200, 300$, and $1,700$, the running time is set to be 600 seconds and all the other settings are the same as in the previous section. The hypervolume of the initial P_E and the running time used for different values of N are reported in Table III and the hypervolume of the final P_E obtained with the different running time are given in Table IV. From the experimental results, it is evident that:

- As expected, N is a crucial factor in determining the computational overhead and performance of the initialization step in MoMad. As shown in Table III, when $N = 50$, the running time consumed by the initialization step is 38 seconds, and the hypervolume of initial P_E is 8.85177×10^{10}. In the case of $N = 1700$, the running time increases to 582 seconds and the hypervolume to 8.87253×10^{10}. These results are not supervising. The larger N value is, the more times the chain Lin-Kernigham heuristic will be conducted and the more solutions will be generated in Initialization. Therefore, more running time and better P_E can be expected.
- For all the different settings of N, the iteration steps after initialization can improve the quality of P_E effectively. For example, in the case of $N = 200$, the hypervolume of P_E has been increased from 8.86107×10^{10} to 8.88140×10^{10}. These results should be due to two reasons. The first one is that the initial P_E are generated by the chain Lin-Kernigham heuristic, thus some of its solutions may not be globally Pareto optimal and the size of the initial P_E is also limited. As a result, there is room for improving the the initial P_E. The second reason is that perturbation and local search in Step 3.1 and Step 3.2 can explore new search areas and produce better solutions.
- The best value of N depends on the available running time. For example, if the available running time is 100 seconds, the best value of N is 200. However, if the running time is 600, the best value is $N = 300$.

2) The Influence of Parameter Max: This parameter is a key parameter determining the overall computational overhead of the PLS procedure. To study its effect on the algorithm performance, we have conducted experiments with its four values $Max = 5, 10, 20$, and 40, all the other settings are the

TABLE I
THE COVERAGE VALUES BETWEEN MOMAD AND 2PPLS FOR MTSP.

Instance		C(A,B)	C(B,A)
$name$	n		
kroab200	200	30.4	14.6
kroab300	300	42.4	24.9
kroab400	400	53.4	28.0
kroab500	500	53.7	38.1
kroab750	750	63.7	35.2
kroab1000	1000	60.4	38.2
euclAB100	100	37.6	4.0
euclAB300	300	46.2	22.5
euclAB500	500	56.4	34.0
ClusterAB100	100	43.2	2.6
ClusterAB300	300	70.5	15.2
ClusterAB500	500	61.1	25.1
rdAB100	100	32.9	23.8
rdAB300	300	49.5	43.4
rdAB500	500	57.5	40.4
mixedAB100	100	36.3	11.9
mixedAB300	300	48.2	28.5
mixedAB500	500	53.0	40.7

Algorithm A corresponds to MoMad
Algorithm B corresponds to the compared algorithm

TABLE II
THE RUNNING TIME (IN SECONDS) OF 2PPLS AND MOMAD.

Instance		2PPLS	MoMad
$name$	n		
kroab200	200	115	94
kroab300	300	232	213
kroab400	400	434	475
kroab500	500	731	823
kroab750	750	2301	1849
kroab1000	1000	7206	3303
euclAB100	100	26	22
euclAB300	300	259	275
euclAB500	500	693	711
ClusterAB100	100	50	46
ClusterAB300	300	366	347
ClusterAB500	500	1518	849
rdAB100	100	25	20
rdAB300	300	305	274
rdAB500	500	816	1211
mixedAB100	100	26	21
mixedAB300	300	238	309
mixedAB500	500	866	1104

2PPLS is tested on a PC with 3.0GHz CPU
MoMad is tested on a PC with 2.4GHz CPU

TABLE III
THE HYPERVOLUME ($\times 10^{10}$)OF THE INITIAL P_E AND THE RUNNING TIME USED WITH DIFFERENT VALUES OF N

N	time	hyperpervolume
100	38	8.85177
200	73	8.86107
300	105	8.86444
1700	582	8.87253

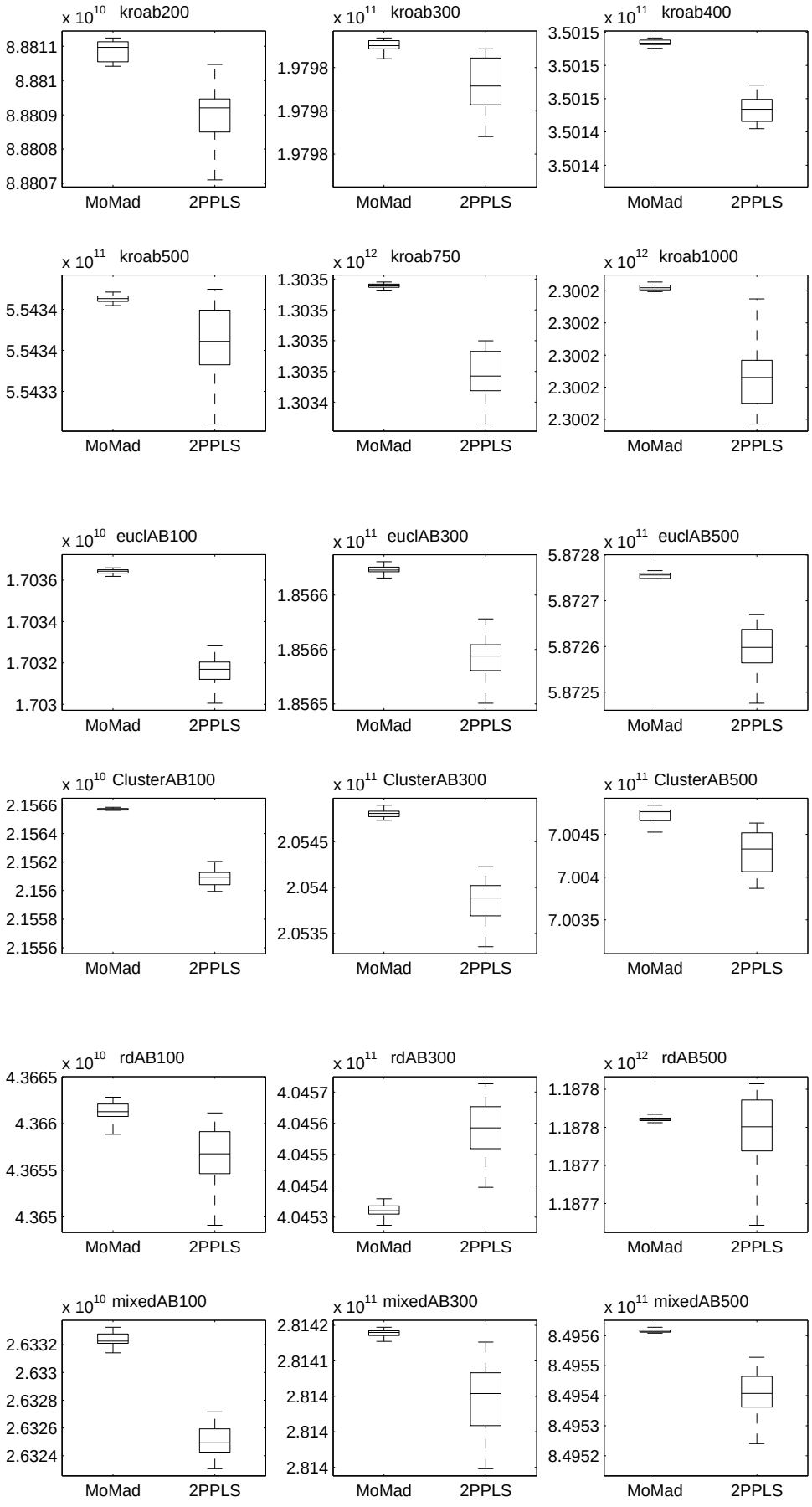

Fig. 3. The hypervolume values obtained by MoMad and 2PPLS on the instances. In each subplot, the left corresponds to the results of MoMad.

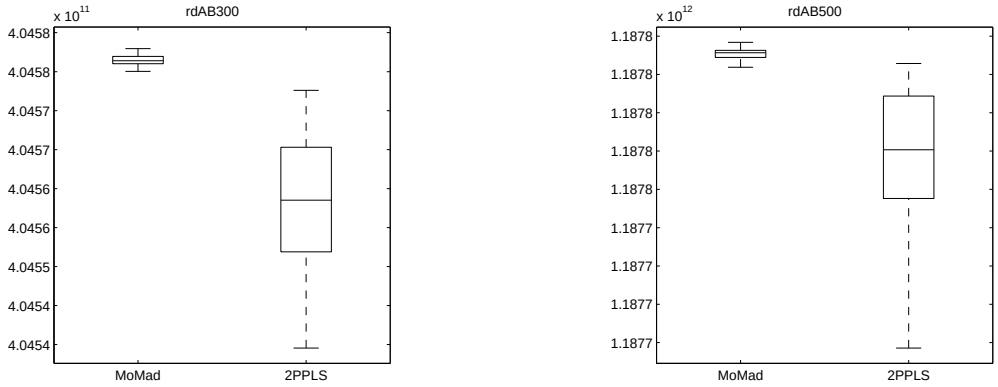

Fig. 4. Larger N is tested for rdAB300 and rdAB500 where $N = 400$ and 600 is used for solving them respectively.

TABLE IV
THE HYPERVOLUME (10^{10}) OF THE FINAL E OBTAINED WITH THE DIFFERENT RUNNING TIME (t).

N	$t = 50$	$t = 100$	$t = 150$	$t = 300$	$t = 600$
100	8.86657	8.88108	8.88115	8.88120	8.88124
200	-	8.88129	8.88135	8.88138	8.88140
300	-	-	8.87294	8.88142	8.88151
1700	-	-	-	-	8.88139

'-' means that the initialization procedure is still uncomplete

TABLE V
THE INFLUENCE OF Max ON HYPERVOLUME (10^{10}) WHEN DIFFERENT RUNNING TIME (t) IS USED.

N	$t = 100$	$t = 150$	$t = 300$	$t = 600$
5	8.88113	8.88130	8.88133	8.88138
10	8.88129	8.88135	8.88138	8.88140
20	8.88129	8.88132	8.88135	8.88140
40	8.88129	8.88132	8.88135	8.88140

same as in Section III.D. The experimental results are reported in Table V, we can make the following comments:

- As for the hypervolumes of the final solution sets obtained after 600 seconds, $Max = 5$ performs a bit poorer than the three other settings, and there is no big difference among $Max = 10, 20, 40$. Thus, we can claim that MoMad is not very sensitive to the setting of Max and it is better to use a large Max value if the available running time allows one to do so.
- In terms of the performance over the whole search period, $Max = 20, 40$ have about the same performance, $Max = 10$ shows a small edge over other values at time points $t = 150$ and 300. It may be worthwhile investigating a dynamic way for adjusting Max in an online manner.

IV. EXPERIMENTAL STUDIES ON THE MULTIOBJECTIVE 0-1 KNAPSACK PROBLEM

In the above section, the test instances are the MTSP with two objectives. Taking the Multiobjective 0-1 Knapsack Problem (MOKP) as a test bed, this section compares MoMad with some other state-of-the-art algorithms, particularly with respect to their ability for dealing with more than two objectives.

A. Multiobjective 0-1 Knapsack Problem (MOKP)

Given a set of n items and a knapsack. Each item j has m weight indexes $w_{i,j} \geq 0$ and m profit indexes $p_{i,j} \geq 0$ $(i = 1, \ldots, m)$. The capacity of the knapsack for weight index i is c_i. A feasible solution x is an item subset which meets the following m constraints:

$$\sum_{j \in x} w_{i,j} \leq c_i, \ i = 1, \ldots, m \tag{6}$$

The i-th objective to maximize is:

$$f_i(x) = \sum_{j \in x} p_{i,j}, \ i = 1, \ldots, m \tag{7}$$

This problem is also known as the multidimensional multiobjective knapsack problem [27]. The MOKP is NP-hard [2] and can model a variety of applications in resource allocation. It has also been widely used in testing multiobjective heuristics [28]. A set of nine test instances proposed in [26] are used in our experimental studies.

B. Implementation of MoMad

1) Initialization of P_L (Step 0.1): x^k to subproblem k is constructed as follows. A utility value of each item j is first defined and computed:

$$\sigma_j^k = \frac{\sum_{i=1}^m \lambda_i^k p_{i,j}}{\sum_{i=1}^m w_{i,j}} \tag{8}$$

where $\sum_{i=1}^m w_{i,j}$ is the overall weight of item k, and $\sum_{i=1}^m \lambda_i^k p_{i,j}$ is its weighted profit, i.e., the coefficient of x^i in the objective function of subproblem k defined by (2) in the MOKP case.

We first initialize $x^k = \emptyset$, and then add items one by one to x^k until there is no item can be added to x^k without invaliding any constraints. Each item added to x^k should have the largest σ_j^k among all the items which can be added to x^k without invaliding any constraints.

2) $N(x)$ in Pareto Local Search (Step 2): If the neighborhood in PLS (Algorithm 3) is too large, the computational overheads of PLS can be very high. Our neighborhood definition is inspired by [11]. To define $N(x)$, we first define a domination relationship among all the items. An item u is said to dominate another item v if and only if

$$\frac{p_{i,u}}{\sum_{k=1}^m w_{k,u}} \geq \frac{p_{i,v}}{\sum_{i=k}^m w_{k,v}}$$

for all $i = 1, \ldots, m$, and

$$\frac{p_{i,u}}{\sum_{k=1}^m w_{k,u}} > \frac{p_{i,v}}{\sum_{i=k}^m w_{k,v}}$$

for at least one index $j \in \{1, \ldots, m\}$. Then we divide all the items into different subsets such that F_1 contains all the nondominated items in $\{1, \ldots, n\}$, F_2 contains all the nondominated items in $\{1, \ldots, n\} \setminus F_1$, and so on. The rank of the items in F_r is set to be r. The lower the rank of an item is, the better it is.

Let $U = \{1, \ldots, n\} \setminus x$ and $\theta \in (0, 1]$ We select the $\theta \times |x|$ best ranked items from set x. If there is any tie in selection, we randomly select one. Let $\bar{x}$ be the set of all these selected items. We also select $\theta \times |U|$ worst items from U and let them constitute $\bar{U}$. Roughly speaking, $\bar{x}$ are the worst items in x and $\bar{U}$ are the best items in U.

Then we define $N(x)$ as the set of all the feasible solutions which can be obtained from x by removing one item in $\bar{x}$ and add one item from $\bar{U}$ to x. Obviously, θ determines the size of $N(x)$.

3) Perturbation (Step 3.1): Let x^k in P_L be the current solution to subproblem k. The output of perturbation $\tilde{x}^k$ is generated as follows: Let S be the number of items in x^k. To perturb x^k, we first remove $\lfloor \alpha \times |x^k| \rfloor$ randomly-selected items from x^k to form initial $\tilde{x}^k$, where $\alpha \in (0, 1)$ is a control parameter. Then as in Initialization of x^k in Step 0.1, add items to $\tilde{x}^k$ in a greedy way until any item addition will violate some constraints.

4) Local Search (Step 3.2): y is a neighbor of x if y is feasible and can be obtained from x by removing one item from it and adding a new item to it. To do local search on $\tilde{x}^k$ for subproblem k, we first find its neighboring solution z with the highest aggregated objective function $f^{ws}(x|\lambda^k)$ value. If z is better than $\tilde{x}^k$ in terms of $f^{ws}(x|\lambda^k)$, then we replace $\tilde{x}^k$ by z and repeat the neighborhood search, otherwise, we stop and deliver $\tilde{x}^k$ as y^k.

C. Experimental Parameter Setup

1) The Size of P_L (N) and Weight Vectors (λ) in MoMad: Their settings are determined by a parameter H. More precisely, $\lambda^1, \ldots, \lambda^N$ constitute all the weight vectors in which each individual weight component takes a value from

$$\left\{ \frac{0}{H}, \frac{1}{H}, \ldots, \frac{H}{H} \right\}.$$

Therefore, the size of P_L (i.e., the number of weight vectors) is:

$$N = C_{H+m-1}^{m-1},$$

where m is the number of objectives. Following the practice in [17] N is set as shown in Table VI.

TABLE VI
THE SETTINGS OF THE NUMBER OF N IN MOMAD FOR THE MOKP.

Instance		N
n	m	
250	2	150
500	2	200
750	2	250
250	3	351
500	3	351
750	3	351
250	4	455
500	4	455
750	4	455

where n is the number of items and m is the number of objectives.

2) Max in PLS: Max is set to be 1 in our experiments.

3) θ in $N(x)$: it is set to be 0.02.

4) α in Perturbation: it is set to be 0.05.

5) The Stopping Condition: The algorithm stops after 1,500 generations in the case of two objectives, and 100 generations in other test instances.

6) The number of Independent Runs: The algorithm has been run 20 times on each test instance.

D. Algorithm in Comparison

- MOTGA [29]: : This algorithm divides the whole PF into a few small parts, each of them is approximated by an independent stage. Each stage consists of a number of consecutive genetic processes, each of which is guided by a different Tchebycheff aggregation function. The experimental results of MOTGA used in our comparison are from its authors' web.
- MEMOTS [30]: It is a memetic algorithm. A dynamic hypergrid in the objective space is used to guide selection. Tabu search is applied to improve offspring. The experimental results of MEMOTS used in our comparison are from its authors' web.
- 2PPLS [27]: The experimental results of 2PPLS used in our comparison are from the author's web. As pointed out

in [27], 2PPLS could involve very heavy computational overhead if the number of objectives is larger than two, there is no experimental results available on instances with three or more objectives.

E. Comparison Results

The hypervolume values of the final solution sets produced by all the algorithms are plotted in Fig.5 7. In computing the hypervolme, $y_i^*(i = 1, \ldots, m)$ is set to be 0. The mean C-metric values between MoMad and the other algorithms on each instance are given in Table VII. The average running times used by these algorithms are given in Table VIII. We have the following observations:

- In terms of both the hypervolume and C-metric values, MoMad is much better than MOTGA and MEMOTS. For example, on the 500-3 instance, a solution set obtained by MoMad can on average dominate 73.2% of a solution set by MOTGA, and 0.9% vice versa; MOMEAD dominates MEMOTS by 72.9%, and 2.6% vice versa. On the same instance, as shown in Fig. 5, MoMad has produced much larger hypervolume values than MOTGA and MEMOTS, which confirms the advantages of MoMad.
- On the three biobjective instances with experimental results of 2PPLS , MoMad performs slightly better than or about the same as 2PPLS with respect to both the hypervolume and C-metric values .
- The running time consumed by MoMad is at the same magnitude as that by the other algorithms with experimental results.

V. Conclusion and future research issues

Evolutionary algorithms, decomposition approaches and Pareto local search methods are often used as basic elements in designing hybrid heuristics for combinatorial MOPs. However, very little, if any, effort has made to study how to combine these three elements together before. This paper has suggested a simple yet efficient multiobjective memtic algorithm, called MoMad. It hybridizes all these three elements in one algorithmic framework. MoMad decomposes an MOP into a number of single objective subproblems by using an aggregation method. It works with three populations: population P_L for recording the current solution to each subproblem, population P_P for storing starting solutions for Pareto local search, and an external population P_E for maintaining all the nondominated solutions found so far during the search. A problem-specific single objective heuristic can be applied to these subproblems to initialize the three populations. At each generation, a Pareto local search method is first applied to search a neighborhood of each solution in P_P to update P_L and P_E. Then a single objective local search applied to each perturbed solution in P_L for improving P_L and P_E, and re-initializing P_P. The procedure is repeated until a stopping condition is met. MoMad provides a generic hybrid algorithmic framework to use problem specific knowledge and employ well developed single objective local search and heuristics, and Pareto local search methods for dealing with combinatorial multiobjective problems. It is a population based iteration method and thus an anytime algorithm. Extensive experiments have been conducted in this paper to study MoMad and compare it with some other state of the art algorithms on the multiobjective knapsack problem and the biobjective traveling salesman problem and the multiobjective knapsack problem. The experimental results show that our proposed algorithm outperforms or performs similarly to the best so far heuristics on these two problems.

The future research avenues include:

- studies of MoMad on other hard or real-world multiobjective problems. Such studies will be useful and necessary for understanding MoMad and establishing some working principles for designing hybrid multiobjective algorithms.
- studies of combination of MoMad with the DM's preference information. It is very interesting how to use such information to guide problem decompostion and Pareto local search in the MoMad framework.
- investigation of on-line dynamic allocation of computational resources to different PF parts in the MoMad framework. It should be an effective way for improving the performance of MoMad.
- combinations of MoMad with machine learning techniques. It is worthwhile study how to using machine learning techniques to model the MOP landscape and use it in the MoMad framework.

The C++ source code of MoMad can be downloaded from Qingfu Zhang's homepage:
http://dces.essex.ac.uk/staff/zhang/.

References

[1] K. Miettinen, *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999.

[2] M. Ehrgott, *Multicriteria Optimization*. Springer, 2005.

[3] Y.-C. Liang and M.-H. Lo, "Multi-objective redundancy allocation optimization using a variable neighborhood search algorithm," *Journal of Heuristics*, vol. 16, no. 3, pp. 511–535, 2010.

[4] E. Ulungu, J. Teghem, P. Fortemps, and D. Tuyttens, "MOSA method: a tool for solving multiobjective combinatorial optimization problems," *Journal of Multi-Criteria Decision Analysis*, vol. 8, no. 4, pp. 221–236, 1999.

[5] L. Paquete and T. Sttzle, "Design and analysis of stochastic local search for the multiobjective traveling salesman problem," *Computers and Operations Research*, vol. 36, no. 9, p. 2619C2631, 2009.

[6] A. Alsheddy and E. Tsang, "Guided pareto local search based frameworks for biobjective optimization," in *IEEE Congress on Evolutionary Computation (CEC)*, 2010, pp. 1–8.

[7] S. Saha, U. Maulik, and K. Deb, "A simulated annealing-based multiobjective optimization algorithm: AMOSA," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 3, pp. 269–283, 2008.

[8] C. Garcia-Martinez, O. Cordon, and F. Herrera, "A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP," *European Journal of Operational Research*, vol. 127, no. 1, pp. 116–148, 2007.

[9] L. PAQUETE and T. STÜTZLE, "A two-phase local search for the biobjective traveling salesman problem," *Lecture notes in computer science*, pp. 479–493, 2003.

[10] T. Lust and J. Teghem, "Two-phase pareto local search for the biobjective traveling salesman problem," *Journal of Heuristics*, vol. 16, no. 3, pp. 475–510, 2010.

[11] T. Lust and A. Jaszkiewicz, "Speed-up techniques for solving large-scale biobjective TSP." *Computers & Operations Research*, vol. 37, pp. 521–533, 2010.

[12] L. G. Eric Angel, Evripidis Bampis, "A dynasearch neighborhood for the bicriteria traveling salesman problem," *Lecture Notes in Economics and Mathematical Systems*, vol. 535, p. 153C176, 2004.

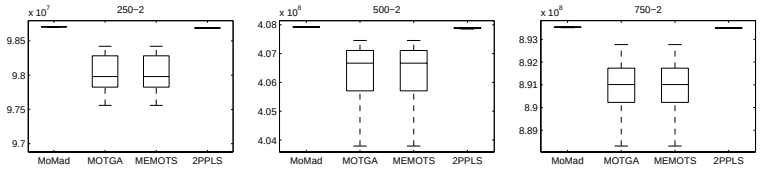

Fig. 5. The hypervolume values obtained by MoMad, MOTGA, MEMOTS, and 2PPLS on three instances with two objectives (from left to right).

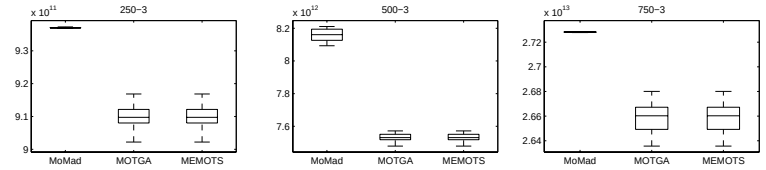

Fig. 6. The hypervolume values obtained by MoMad, MOTGA,and MEMOTS on three instances with three objectives (from left to right).

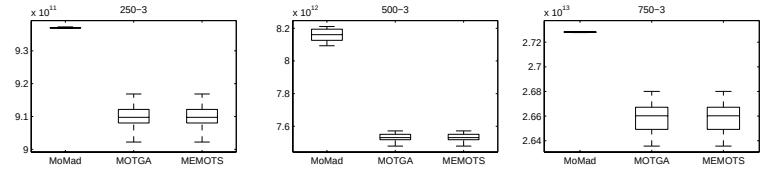

Fig. 7. The hypervolume values obtained by MoMad and MOTGA on three instances with four objectives (from left to right).

TABLE VII
THE COVERAGE VALUES BETWEEN MOMAD AND MOTGA, MEMOTS, 2PPLS FOR THE MOKP.

Instance	MOTGA		MEMOTS		2PPLS	
n	$C(A,B)$	$C(B,A)$	$C(A,B)$	$C(B,A)$	$C(A,B)$	$C(B,A)$
250-2	92.1	0.1	57.7	4.8	19.2	10.5
500-2	95.2	0.2	82.5	3.6	26.9	25.2
750-2	91.4	1.6	65.7	23.0	47.5	32.4
250-3	86.7	0.4	77.5	3.0	N.A.	N.A.
250-4	78.6	0.9	76.4	3.9	N.A.	N.A.
500-3	73.2	0.9	72.9	2.6	N.A.	N.A.
500-4	66.6	0.3	N.A.	N.A.	N.A.	N.A.
750-3	48.6	0.5	N.A.	N.A.	N.A.	N.A.
750-4	41.7	0.6	N.A.	N.A.	N.A.	N.A.

A corresponds to MoMad
B corresponds to the compared algorithm
N.A. means unavailable

[13] L. Paquete, M. Chiarandini, and T. Stützle, "Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study," *Metaheuristics for Multiobjective Optimisation*, vol. 535, pp. 177–200, 2004.

[14] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II." *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[15] Y. Jin, T. Okabe, and B. Sendhoff, "Adapting weighted aggregation for multiobjective evolution strategies," in *First International Conference on Evolutionary Multi-criterion Optimization (EMO 2001)*, ser. LNCS, vol. 1993. Zurich, Switzerland: Springer, 2001, pp. 96–110.

[16] E. Zitzler and S. Knzli, "indicator-based selection in multiobjective search," *Lecture Notes in Computer Science*, vol. 3242, pp. 832–842, 2004.

[17] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[18] Y. Mei, K. Tang, and X. Yao, "Decomposition-based memetic algorithm for multi-objective capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 151–165, 2011.

[19] R. Saldanha, B. Gomes, A. Lisboa, and A. Martins, "A multi-objective evolutionary algorithm based on decomposition for optimal design of yagi-uda antennas," *IEEE Transactions on Magnetics*, vol. 48, no. 2, pp. 803–806, 2012.

[20] H. Ishibuchi and T. Murata, "Multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, no. 3, pp. 392–403, 1998.

[21] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization*. Springer, 2009.

[22] H. Li and Q. Zhang, "Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II," *IEEE Transactions Evolutionary Computation*, vol. 13, no. 2, pp. 284–302, 2009.

[23] Q. Zhang, W. Liu, E. Tsang, and B. Virginas, "Expensive multiobjective optimization by MOEA/D with gaussian process model," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 5, pp. 456–474, 20010.

[24] Q. Zhang, A. Zhou, and Y. Jin, "RM-MEDA: A regularity model based multiobjective estimation of distribution algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 41–63, 2008.

[25] A. D., "Chained linckernighan for large traveling salesman problems," *INFORMS Journal on Computing*, vol. 15, p. 82C92, 2003.

[26] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach." *IEEE Trans. Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[27] T. Lust and J. Teghem, "The multi-objective multidimensional knap-

TABLE VIII
THE RUNNING TIME (IN SECONDS) CONSUMED BY MOMAD, MOTGA, MEMOTS AND 2PPLS

Instance	MoMad	MOTGA	MEMOTS	2PPLS
250-2	7.7	1.6	5.0	5.0
500-2	24.1	7.7	23.0	26.0
750-2	44.4	20.8	59.0	51.0
250-3	1.2	2.9	11.0	N.A.
500-3	13.3	13.7	38.0	N.A.
750-3	19.6	35.6	95.0	N.A.
250-4	6.7	4.5	N.A.	N.A.
500-4	15.6	19.4	N.A.	N.A.
750-4	34.7	55.4	N.A.	N.A.

MOTGA, MEMOTS and 2PPLS were tested on a PC with Pentium 3.2GHz, 3.0GHz, and 3.0GHz CPU respectively. N.A. means that the result is not available.

sack problem: a survey and a new approach," in *Computing Research Repository*, 2010.

[28] A. Jaszkiewicz, "On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment." *IEEE Trans. Evolutionary Computation*, vol. 6, no. 4, pp. 402–412, 2002.

[29] M. J. Alves and M. Almeida, "MOTGA: A multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem." *Computers & Operations Research*, vol. 34, pp. 3458–3470, 2007.

[30] T. Lust and J. Teghem, "MEMOTS: a memetic algorithm integrating tabu search for combinatorial multiobjective optimization," *RAIRO Operations Research*, vol. 42, pp. 3–33, 2008.