A hybrid heuristic algorithm for solving the maximally diverse grouping problem

by

Xiao Yang

A dissertation

submitted to the Graduate School of Science and Engineering for Education

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Engineering

University of Toyama

Gofuku 3190, Toyama-shi, Toyama 930-8555 Japan

2022

(Submitted December 19, 2022)

Abstract

Due to combinatorial optimization (CO) problems play an important role in scientific and industrial world, a number of heuristic algorithms used to manage with CO problems have been rapidly developed and improved. This paper introduces a novel hybrid algorithm, called a three-phase search approach with dynamic population size (TPSDP), for solving a CO problem, named the maximally diverse grouping problem (MDGP). MDGP aims to assign a given set of elements into a number of groups with size restrictions for the sake of maximizing the sum of diversity in these groups. MDGP is an NP-hard CO problem, possessing widespread application and practical importance. The proposed TPSDP devises the search process into three phases with distinct functions which are iterated: (1) an undirected perturbation phase to improve the population diversity, (2) a restructure phase using a distinctive crossover operator to increase the information interaction among solutions, and (3) a directed perturbation phase to discover the adjacent local optima around current solutions. TPSDP also combines a dynamic population size strategy to reserve limited computing resources for potential solutions. The results of experiments and the Friedman test show that the overall performance of the proposed TPSDP is highly competitive with even better than previous state-of-the-art MDGP algorithms on 500 instances from five popular benchmark sets. Furthermore, an additional experiment of parameter analysis and a discussion of critical ingredients are presented.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The improvement of human social productivity comes from two aspects, one is the discovery of new technologies, and the other is the rational allocation of resources based on existing technologies to improve resource utilization. With the continuous development of the world economy, information science and technology has been widely developed and applied. At present, information technology has been widely used in all walks of life, especially in the direction of intelligence and large-scale resource integration. The emergence of various new economic forms and the rise of intelligent applications are driven by the strong demand of economic development for rational resource allocation after the maturity of basic technologies.

The problem of resource allocation is usually reduced to an optimization problem after modeling. Optimization problems can be divided into two categories: one is continuous variable problems, the other is discrete variable problems. A problem with discrete variables is called combinatorial. In a problem of continuous variables, it is generally to find a set of real numbers, or a function; In a combinatorial problem, it is to obtain an object from an infinite set or a countable infinite set, it may be an integer, a set, a permutation, or a graph. Generally, these two types of problems have quite different characteristics and the methods to solve them are also very different. For problems with discrete variables, the problem of finding the optimal solution from a finite number of solutions is the combinatorial optimization problem [1]. To put it simply, combinatorial optimization problems are a class of problems that seek extreme value in discrete state. As a coherent mathematical discipline, combinatorial optimization problems are relatively young [2]. However, along with the industrial and technological revolution and the development of modern management science, especially the rapid progress of computer technology and its wide application in various industries, combinatorial optimization has grown into an independent branch of operational research.

One reason for the diversity of the root causes of combinatorial optimization is that its problems directly originate from practice [2]. Typical combinatorial optimization problems include traveling salesman problem [3], scheduling problem [4], knapsack problem [5], packing problem [6], maximum clique problem [7], clustering problem [8], graph coloring problem [9], etc. The definition of the famous traveling salesman problem (TSP) can be summarized as that a traveling salesman wants to start from his hometown, visit each town only once, and finally return to his hometown. One of his problems was to find the shortest path of the trip. A direct application of the TSP is the drilling problem whose solution plays an important role in economical manufacturing of printed circuit boards [3]. Similarly, the scheduling problem often encountered in the manufacturing industry can be defined as allocating limited resources and time to several tasks under certain constraints to satisfy or optimize one or more performance metrics. In the transportation industry, ships, trains, aircraft, or trucks often carry loads for different clients. Transportation requests arrive stochastically over time, and prices are offered or negotiated for transporting loads. This kind of problem is usually modeled as a knapsack problem, that is, to select the items to be loaded into a fixed capacity knapsack from a set of given items with known size and reward, so as to obtain the maximum total return within the capacity limit [5].

The description of these problems is very simple and has strong engineering representation, but the optimization is very difficult to solve. The main reason is that the algorithms for solving these problems need extremely long running time and huge storage space, so that it is impossible to implement them on existing computers, namely the so-called ”combination explosion”. It is the representativeness and complexity of these problems that arouse people’s interest in the research of combinatorial optimization theory and algorithm. When faced with these combinatorial optimization problems, early researchers still hoped to calculate the optimal solution and proposed branch-and-bound method [10], cutting plane method [11], dynamic programming approach [12] and other exact algorithms.

When the scale of the problem is small, such exact algorithms can find the optimal solution to such problems in an acceptable time, but when the scale of the problem is large, the computational time spent increases exponentially with the size of the instances, and these exact algorithms cannot even give a feasible solution. Such problems have been proved to have NP-hard complexity [13], and it is now generally considered that there are no algorithms whose upper bound is polynomial in time complexity.

Because this kind of problems exist widely in practice, it is unnecessary to find the optimal solution in practice. Later researchers abandoned the goal of the strict optimal solution, and then studied the algorithms available in reality for such problems. In this process, approximate algorithms [14] were generated. Approximate algorithm refers to an algorithm that uses approximate methods to solve combinatorial optimization problems, considering obtain an approximate solution that is close to the optimal solution in polynomial time. The approximate algorithm can ensure that the difference between the computational result and the optimal result does not exceed a certain constant, but the algorithm is complex and difficult to program on a computer. Greedy algorithm [15], semi-definite programming approaches [16], linear programming [17], etc. can be used to construct approximate algorithms. With the development of computer technology, the study of heuristic algorithms [18] have gradually flourished. A heuristic algorithm can be defined as an algorithm based on intuition or experience, which gives a feasible solution to the combination optimization problem to be solved within an acceptable computing cost [19]. Heuristic algorithms are usually simple and easy to be implemented on computers, but the deviation between the feasible solution and the optimal solution may not be predictable in advance, and these heuristic algorithms usually depend on specific problems and are not universal. At this time, many scientists sought new inspiration for artificial systems from biology. Some scientists independently developed simulated evolutionary algorithms suitable for optimization of complex problems in the real world from the mechanism of biological evolution, and a large number of meta-heuristic algorithms [20] appeared one after another. A meta-heuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions [21]. Meta-heuristic algorithms include but are not restricted to, ant colony optimization (ACO) [22,23], particle swarm optimization (PSO) [24,25], artificial bee colony algorithm (ABC) [26], genetic algorithm (GA) [27], simulated annealing algorithm (SA) [28], tabu search algorithm (TS) [29], iterative local search (ILS) [30], a series of universal heuristic algorithms inspired by natural phenomena. As a general optimization mechanism, the above algorithms can not only solve large-scale problems in a relatively short time, but also more importantly, the optimization mechanism of such algorithms does not rely too much on the organizational structure information of the algorithm. Although meta-heuristic algorithms can not guarantee the optimal solution, they can be applied to a wide range of problems due to their good practicality.

The maximally diverse grouping problem (MDGP) is a combination optimization problem derived from practice, which requires a grouping scheme that satisfies the constraints of groups so that the sum of the diversity of all groups is maximized, given that the diversity matrix of a set of elements is known. The maximally diverse grouping problem has been shown to be theoretically NP-hard complex, and the scale of instances of MDGP is usually large, so there is no algorithm that can obtain exact optimal solution in an acceptable time. At present, researchers of MDGP focus on heuristic algorithms.

This paper presents a novel and effective hybrid algorithm, called a three-phase search approach with dynamic population size (TPSDP), for solving MDGP. Inspired by the three-phase local search [31] framework, the TPSDP algorithm iteratively uses the three-phase with distinct functions to achieve a balance between diversification and intensification in the search process. In the first phase, TPSDP draws on an undirected perturbation operator, which is adopted from [32], followed by a local search process. This phase improves the diversification of solutions. In the second phase, an extended crossover operator based on [33] is proposed for both equal group size instances and different group size instances. The second phase improves the information interaction among solutions, and it can be considered as a transition phase. In the third phase, a novel directed perturbation operator is proposed to fine-turn current solutions. This operator shifts the search to limited regions around the current solutions and finds their adjacent promising solutions. This phase intensifies the quality of solutions. In addition, a dynamic population size strategy is utilized to improve the efficiency of the algorithm. Experimental results on five widely used MDGP benchmark sets show that TPSDP has significantly better or at least competitive performance compared to the state-of-the-art algorithms, especially on instances with different group sizes.

The main contributions of this study can be summarized as follows:

(1) TPSDP adopts a time-varying population size strategy to avoid redundant examination of no-promising solutions and enable promising solutions to be allocated with more computational resources in a given time budget. Thereby, the dynamic population size strategy improves the efficiency of the algorithm.

(2) I present an extended crossover operator applied not only to exceptional cases like the instances with equal group sizes (EGS) but also to the instances with different group sizes (DGS) in MDGP. The crossover operator maintains the integrity of the part inherited from the parent solutions as much as possible, thus guaranteeing the quality of the offspring solutions to some extent. At the same time, the crossover operator also enhances the information interaction among solutions.

(3) A novel directed perturbation operator is proposed to exploit the neighborhood of current solutions and find potential neighbor solutions. Additionally, the idea of the directed perturbation operator is general and can be applied to other algorithms for MDGP or other related combinatorial optimization problems.

The remaining parts of this paper are structured as follows: Chapter 2 lists several metaheuristic techniques applied to CO problems. The introduction of MDGP is summarized in Chapter 3. Chapter 4 presents the components of the proposed TPSDP algorithm. Chapter 5 shows the computational results and assessments based on the benchmark instances used in the recent literature. Analysis of the parameter settings and discussion of the proposed algorithm are given in Chapter 6. This paper ends with a conclusion in Chapter 7.

Chapter 2

Related work

Meta-heuristics can be said to originate from the Artificial Intelligence and Operations Research communities [20,53,54]. The term meta-heuristics usually refers to the approximate algorithm for optimization, not specifically expressed for a specific problem. During the past two decades, due to advances in mathematical programming theory and algorithmic design, the rapid improvement of computer performance, and the development of complex software, meta-heuristic technology as one of the optimization tools has been greatly improved. This chapter will introduce several meta-heuristic techniques applied to combinatorial optimization (CO) problems in brief.

Variable neighborhood search (VNS) is a meta-heuristic proposed by authors more than a decade ago [55]. The motivation of this approach can be found in some earlier work, like [56–59]. The basic idea of VNS is a systematic change of neighborhood, finding a local optimum in a descent phase and getting out of the corresponding basin in a perturbation phase. Before generating an initial solution, a set of neighborhood structure should be defined (see Algorithm 1). The main cycle of VNS contains three phases: shaking, local search, and move. In the shaking phase, which can also be called perturbation phase, a solution s' is randomly selected from the kth neighborhood of the incumbent solution s, and becomes a new start point of the next iteration. At the end of a round of local search, the new solution s'' is compared with s, and if s'' is accepted, the algorithm starts again with $k = 1$. Or else, k is incremented and a new round of shaking phase starts with a distinct neighborhood.

VNS was originally designed for the approximate solution of CO problems, and later extended to tackle mixed integer programming, nonlinear programming and the latest mixed integer nonlinear programming. Moreover, VNS has been also used as a tool for automatic graph theory or computer-assisted graph theory. VNS has increasing applications and pertain to many fields: cluster analysis, vehicle routing, lot-sizing, engineering, biology, phylogeny, telecommunication design, location theory, scheduling, network design, artificial intelligence, pooling problems, reliability, geometry, etc. (see, e.g., [60–67]). The VNS framework has been also adopted in the proposed TPSDP.

Algorithm 1: Main framework of VNS

```
begin
    Choose a set of neighborhood structures N_k, k = 1, 2, ..., k_max
      s ← GenerateInitialSolution();
    while terminate condition not met do
        k ← 1;
        while k<k_max do
            s' ← PickAtRandom(N_k(s));
            s'' ← LocalSearch(s') if (f(s'')<f(s') then
                s ← s'';
                k ← 1
            else
                k ← k + 1
            end
        end
    end
end
```

Simulated annealing (SA) algorithm [68] was first invented in 1983, using a method similar to hill-climbing, but accepts some deteriorating solutions with an acceptance probability which is decreasing with time, in order to escape from local minima. The SA algorithm (Algorithm 2) starts from an initial solution and an initial so-called temperature parameter T. Then it repeats numbers of iteration until a terminate condition is met. At each iteration, a solution randomly picked from a neighborhood is sampled and accepted as a incumbent solution depending on its fitness value and temperature parameter T. The temperature T is decreased with the search process, therefore, the probability of accepting a worse solution is high at the beginning of the search and it gradually decreases. This process is similar to the annealing process of glasses and metals, which exhibit a low-energy configuration when cooled using an appropriate cooling schedule. SA can apply to several CO problems, like the quadratic assignment problem (QAP) [69] and the job shop scheduling (JSS) problem [70], etc. In addition, among the clique partitioning problem I am studying, the state-of-the-art algorithm in the literature also utilize SA as the framework of local search.

Algorithm 2: Main framework of SA

```
begin
    s ← GenerateInitialSolution();
    T ← T_0;
    while terminate condition not met do
        s' ← ChooseRandom(N(s));
        if (f(s') ≤ f(s) then
            s ← s'
        else
            Accept s' as a new solution with probability p(T, s, s')
        end
        Update(T);
    end
end
```

Tabu search (TS) [71] is a global neighborhood search algorithm, which is one of the most cited and used meta-heuristic in CO problems. The basic idea of TS is to simulate the optimization features of human memory function. It avoids circuitous search by local neighborhood search mechanism and using a short term memory, and releases some forbidden high-quality solutions by breaking the aspiration criteria, thus ensuring diverse and effective exploration to finally achieve the global optimum. The simple TS framework is shown in Algorithm 3.

The short memory implemented as a tabu list. The main purpose of the tabu list is to prevent endless cycles in the search process and avoid getting trapped in local optimum. It is usually used to record the movements of the previous several times and prohibit these movements from returning in the near future. The tabu list is the core of TS algorithm. The length of tabu list ((i.e., the tabu tenure) affects the search speed and the quality of solutions. If the length of the tabu list is too small, the search will focus on small regions of the search space, and the search process may enter an endless loop. On the contrary, too large tabu list forces the search process to explore larger areas, but high-qulity solutions may be skipped and solutions that cannot be improved will increase the running time of the algorithm. Therefore, the size of the tabu tenure can make the search divergent or convergent. In a search process, a dynamic tabu list can lead to a more robust algorithm [72, 73]. When forbidden solutions contains high-quality unvisited solution, the algorithm should accept the solution without being restricted by the tabu list. In order to overcome this problem, the aspiration criterion is defined. The measurement standard is to define a aspiration level function, which usually selects the fitness value of the optimal solution obtained so far as the aspiration level function.

Overall, tabu search field is a rich source of ideas, some of which have been and are being adopted by other meta-heuristics. Furthermore, TS has been successfully applied to many CO problems, such as the reactive tabu search to the MAXSAT problem [73], the reactive tabu search to assignment problem [74] and the vehicle routing area [75], etc.

Algorithm 3: Main framework of TS

```
begin
    s ← GenerateInitialSolution();
    TabuList ← ∅;
    while terminate condition not met do
        s ← SelectBestOf(N(s)\TabuList);
        Update(TabuList);
    end
end
```

Swarm intelligence (SI) algorithm [76] was proposed in 1989. It is inspired by the collective behavior of schools of fish, flocks of birds, swarm of insects and other biological aggregation. In the past decades, a number of swarm intelligence algorithm have been developed. The most successful SI algorithms are ant colony optimization (ACO) and particle swarm optimization (PSO). ACO is inspired by the behavior of ants in the process of food search or risk avoidance, while PSO is based on a simplified model of bird flocking behavior. Here, I will choose PSO for a brief introduction.

As a well-known swarm intelligence algorithms, PSO has attracted great interest with regard to theoretical value and real-world applications. In PSO (see Algorithm 4), a group of particles performs the search process in a given problem space. Each particle maintains its velocity and position, adjusts its velocity and position with some random perturbation, and share its current best position with other one or more particles in the swarm to determine its next position throughout the search space. Once all particle update their respective positions, the next iteration starts, approaching the region near the optimum. At the end, the whole swarm probably presses on towards the global optimal at a convergence speed. Since its simplicity and effectiveness, PSO is widely applied to various CO problems in the fields of wireless-sensor networks [77], electric power systems [78], and data mining [79].

Algorithm 4: Main framework of standard PSO

```
Create and initial an N D-dimension swarm;
repeat
    for particle i = 1, 2, ..., N do
        if f(x_i) < f(p_best_i) then
            p_best_i = x_i
        end
        if f(p_best_i) < f(g_best) then
            g_best = p_best_i
        end
    end
    for particle i = 1, 2, ..., N do
        Update particle's velocity;
        Update particle's position;
    end
until stopping condition is true;
```

Evolutionary algorithms (EAs) are methods that modeling the process of natural evolution. They apply the principle of survival on individuals in a population, each of which represents a potential solution of search space, to purse regions near the perceived optimum. A new set of approximations is arbitrarily initialized, and it evolves towards better regions of search space by using operators called recombination or crossover, modification or mutation, and selection.

The most famous among EAs are genetic algorithm (GA) which have been invented in 1975. GA is a population based optimization problem, which exploits the concept of survival of the fittest. The basic elements of GA are a population of chromosome, fitness function computation, and genetic operators (i.e., selection, crossover, and mutation). The first step of any GA is to create an initial population. In the canonical genetic algorithm, each gene or chromosome will be represented as a binary string of length l. It provides faster and easier implementation of genetic operators. After generating an initial population, each string (solution) is evaluated by the evaluation function and given a fitness value. Then, selection procedure is carried out.

Selection is an significant step in a genetic algorithm that decides which string can participate in the crossover process. The way to do selection is various, such as roulette wheel, tournament, rank, boltzmann, and stochastic universal sampling. For example, the roulette wheel might regard all the string as mapping onto a wheel, where each individual is represented by a portion of the wheel that proportionally according to its fitness value. Then spinning the roulette wheel repeatedly, individuals are chosen by stochastic sampling to select specific solutions that will get involved in formation of the next generation [80].

After selection, the construction of the intermediate population is done and crossover procedure can be executed. The aim of this process is to produce the next population from the intermediate population and provide diversity for the population. As a tool for creating the offspring, the crossover operator combines the genetic information of two or more parents. Picking a random crossover point in the two parents solution, the genetic information beyond that point will be swapped with each other. Two newly formed strings will be inserted into the new population.

To maintain the genetic diversity from one population to the next populaton, the genetic algorithm applies a mutation operator. The frequently-used mutation operators are displacement, simple inversion, and scramble mutation. Put it briefly, the mutation operator performs mutation on some bit in the population, with some low probability. After finishing the above three process, the next population will be evaluated. The process of evaluation, selection, crossover and mutation constitutes one generation in a genetic algorithm execution process.

As an enfficient meta-heuristic, genetic algorithms have been successfully applied in various CO problems. A few application areas are listed: facility layout problem (FLP) [81], scheduling [82], inventory control [83], forecasting [84, 85], and supply network design [86, 87]. In our TPSDP, the crossover operator also plays an important role in solving MDGP.

Algorithm 5: Main framework of EA

```
begin
    P ← GenerateInitialPopulation();
    Evaluate(P);
    while terminate condition not met do
        P′ ← Recombine(P);
        P″ ← Mutate(P′);
        Evaluate(P″);
        P ← Select(P″ ∪ P)
    end
end
```

Artificial neural networks (ANNs) [88] is also called neural networks (NNs) or connection model for short. It is an algorithmic mathematical model that mimic the behavior characteristics of animal neural network for distributed parallel information processing. This kind of networks relies on the complexity of the system, and achieves the purpose of processing information by adjusting the interconnection between a large number of internal nodes.

The method of using neural networks to solve CO problems can be traced back to Hopfield network proposed by Hopfield et al. in 1985 [89]. This neural network is used to solve TSP and other CO problems. However, the neural network can only learn and solve the single small-scale TSP instance at a time. For a newly given TSP instance, it needs to be trained again from the beginning, which has no advantage over traditional algorithms.

The neural networks can really effectively solve the CO problem in 2015. Vinyals et al. [90] analogized the CO problems to the machine translation process (i.e., sequence-to-sequence mapping). The input of the neural network is the feature sequence of the problem (such as the coordinate sequence of the city), and the output of the neural network is the solution sequence (e.g., the order of visits to cities). According to this idea, Vinyal et al. improved the classical sequence-to-sequence (Seq2Seq) mapping model in the field of machine translation, and proposed a pointer network model that can solve CO problems. The author trained the network in a supervised learning way and achieved high-quality results on TSP. The traditional CO algorithms are all solved by ”iterative search”, but Vinyals et al.’s model can directly output solutions by using neural networks, opening a new research field of combinatorial optimization.

Chapter 3

The maximally diverse grouping problem

The maximally diverse grouping problem (MDGP) is devoted to partitioning a set of elements (or nodes) into a set of mutually independent groups (or subsets), maximizing the sum of the diversity between each pair of elements assigned to the same group. Consider an undirected complete and edge-weight graph $G = (V, E, D)$, where $V = \{1, 2,..., n\}$ is the set of n vertices, $E = \{\{i, j\} : i, j \in V, i \neq j\}$ is the set of $n \times (n-1)/2$ edges, and $D = \{d_{ij} \geq 0 : \{i, j\} \in E\}$ is the set of non-negative edge weights. Let g denotes the g^{th} group, where $g \in \{1, 2, ..., m\}$. The size c_g of each group is in a given interval $[L_g, U_g]$, where L_g and U_g denote the lower and upper bounds, respectively. In MDGP, a vertex $v \in V$ can also be called an element, and an edge weight $d_{ij} \in D$ represents the diversity between element i and j. The objective of the MDGP is to maximize the overall edge weights of the m groups. Mathematically, the MDGP can be formulated as follows:

$$\text{maximize} \sum_{g=1}^{m} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_{ig} x_{jg} \tag{3.1}$$

$$\text{s.t.} \sum_{g=1}^{m} x_{ig} = 1, i = 1, 2, \ldots, n \tag{3.2}$$

$$L_g \leq \sum_{i=1}^{n} x_{ig} \leq U_g, g = 1, 2, \ldots, m \tag{3.3}$$

$$x_{ig} \in \{0, 1\}, i = 1, 2, \ldots, n; g = 1, 2, \ldots, m \tag{3.4}$$

where x_{ig} is a binary variable and takes the value of 1 if element i belongs to the group g, and the opposite is 0. The Eqs. (2) and (3) are constraints where (2) enforces that each element will be put into one specific group while constraint (3) guarantees the size of each group will lie between the lower bound L_g and upper bound U_g.

MDGP is a combinatorial optimization problem that originated in practice and has many practical applications. One of the most intuitive and the earliest applications in practice is the human resource grouping problem, such as the assignment of students to groups [34–38] and peer review [39]. For instance, in MBA programs [35], it is important to assign students to diverse study groups. The general purpose of these tasks is to create diversity-rich groups that allow the group members to be in a rich interpersonal and learning environment, which helps to motivate the members and bring out their strengths, resulting in solid and efficient groups. Other applications include VLSI design [40] and the storage allocation of large programs onto page memory [41].

MDGP has been widely studied as a topic based on its theoretically proven NP-hard complexity, and its value for life applications [42]. Several algorithms have been proposed in the literature that can find an approximate optimal solution within an acceptable time. These algorithms can be divided into two categories: (1) single point-based local search algorithms, and (2) hybrid evolutionary algorithms. The first category includes a multistart algorithm [43], a Weitz-Jelassi (WJ) algorithm [44] which implements a heuristic used to construct the initial solution, a Lotfi-Cerveny-Weitz (LCW) algorithm [34] which focuses on constructing the initial solution and simple local search, a tabu search with strategic oscillation (TS-SO) [45], multi-start simulated annealing (MAS) [46], a variable neighborhood search (VNS) [46], a general variable neighborhood search (GVNS) [47] which is a variant of VNS, a skewed general variable neighborhood search (SGVNS) [48] which is an extension of the GVNS, an iterated tabu search (ITS) [49], an iterated maxima search (IMS) [32], and the latest published neighborhood decomposition-based variable neighborhood search and tabu search (NDHA) [50]. The second category includes a hybrid genetic algorithm [51], a hybrid steady-state genetic algorithm (HGA) [46], an artificial bee colony algorithm (ABC) [52], a new hybrid genetic algorithm (NSGGA) [33] that solves only equal-sized instances. According to the experimental results reported on benchmark instances, ITS, IMS, NSGGA, and NDHA can be considered as state-of-the-art algorithms for MDGP.

Chapter 4

A three-phase search approach with dynamic population size for MDGP

4.1 General framework

The implementation framework of TPSDP for MDGP is summarized in Algorithm 6. Three distinct phases control the search behavior of the TPSDP. The first phase, called the undirected perturbation phase, consists of an undirected perturbation operator, which aims to strongly modify the current solutions to jump out of the current search regions and move to new regions far away. This process increases the diversity of the population and, at the same time, prevents premature convergence. Therefore, this phase can also be seen as a diversification phase. The second phase uses a crossover operator to produce high-quality offspring solutions, and to retain the parents' strengths. This phase maintains the diversity of the population and improves the information interaction among solutions, resulting in high-quality solutions. The second phase is the transition phase of the algorithm. In the third phase, a newly proposed directed perturbation operator is used to discover solutions with better quality in the regions adjacent to the current solutions. The third phase can be considered as an intensification phase. It should be noticed that each phase follows a local search in order to find local optimum solutions with higher quality. These processes are iterated until a termination condition is encountered. In addition, a dynamic population size strategy is used to improve the efficiency of the TPSDP algorithm.

Algorithm 6: The main procedure of TPSDP

```
Function TPSDP(β_max, β_min, θ_max, θ_min, η_d, α, t_max)
    /* Population Initialization */
    for i = 1 to β_max do
        S_i = InitialSolution();
        S_i = LocalSearch(S_i);
    end
    β = β_max;
    θ = θ_max;
    while Time() < t_max do
        /* Undirected Perturbation Phase */
        Update η_s;
        for i = 1 to β do
            S_i = UndirectedPerturbation(S_i, η_s);
            S_i = LocalSearch(S_i);
        end
        if β > 1 then
            /* Reconstruction Phase */
            for i = 1 to β do
                Randomly select an individual j, j ≠ i;
                O_i = Crossover(S_i, S_j);
                O_i = LocalSearch(O_i);
                if f(O_i) > f(S_i) then
                    S_i = O_i;
                else if f(O_i)/f(S_i) + α × Dis(S_i, O_i) > 1 then
                    S_i = O_i;
                else
                    S_i = S_i;
                end
            end
        end
        /* Directed Perturbation Phase */
        for i = 1 to β do
            S_i = DirectedPerturbation(S_i, η_d);
            S_i = LocalSearch(S_i);
        end
        β = (β_min − β) * Time()/t_max + β;
        θ = θ_max − (θ_max − θ_min) × Time()/t_max;
        Sort all S according to f and record the best solution S^b;
    end
    return S^b;
end
```

4.2 Population initialization

Algorithm 7: Initial population

```
Function InitialSolution()
    V = {1, 2, ..., n}; G = Ø;
    for g = 1 to m do
        c_g = 0;
    end
    for g = 1 to m do
        while c_g < L_g do
            v = RandomEle(V);
            s[v] = g;
            c_g = c_g + 1;
            V = V \ {v};
        end
    end
    for g = 1 to m do
        if c_g < U_g then
            G = G ∪ {g};
        end
    end
    while V ≠ Ø do
        g = RandomEle(G);
        v = RandomEle(V);
        s[v] = g;
        c_g = c_g + 1;
        V = V \ {v};
        if c_g = U_g then
            G = G \ {g};
        end
    end
    return s;
end
```

TPSDP algorithm uses random initialization to build the initial population due to its simplicity and generality. The initial population of TPSDP consists of β_{max} solutions. Each solution is generated by three steps as shown in Algorithm 7. First, fill each group g ($g \in \{1, 2, ..., m\}$) until reaching their lower bound L_g with randomly selected elements. Then, select an unassigned element at random to fill a group whose current size c_g is not reached its upper bound U_g. Repeat this process until all elements are assigned. Finally, each solution is enhanced by a local search. The best one among the population is recorded as the current best solution S^b.

4.3 Double-neighborhood local search method

4.3.1 Solution space of MDGP

For a given MDGP instance with n elements, m groups, and a diversity matrix $D = [d_{ij}]_{n\times n}$, the solution space searched by the TPSDP algorithm covers solutions formed by assigning n elements to the m groups, satisfying the restrictions that each group g contains at least L_g and at most U_g elements. A solution in search space is expressed by an n-dimensional vector s, where $s[i]$ $(i = 1, 2, ..., n)$ refers to a particular group containing the element i. Additionally, to improve computational efficiency of local search, I introduce an $n \times m$ matrix M such that $M[i][g]$ is used to represent the sum of the diversity between element i and all elements in the group g:

$$M[i][g] = \sum_{i,j=1,2,...,n; s[j]=g} d_{ij} \tag{4.1}$$

where the calculation of matrix M has complexity $O(N^2)$.

4.3.2 Double-neighborhood local search

As the basis of the overall algorithm process, the local search introduced in this paper is the double neighborhood local search used in [32], as detailed in Algorithm 8. As the name implies, the double neighborhood local search method uses two underlying neighborhoods: the insertion neighborhood and the swap neighborhood.

The insertion neighborhood is also called the constrained one-move neighborhood in [32], which is represented by N_1. Given a feasible solution S^f, the search is realized by moving an element v from its current group g_1 to another group g_2 while following the constraints on the size of each group in S^f. All solutions created in this way are called

Algorithm 8: Double-neighborhood local search method

```
Function LocalSearch(s, f(s))
    Initialize M[n][m];
    imp = true;
    while imp do
        imp = flase;
        for v = 1 to n do
            for g = 1 to m do
                if (s[v] ≠ g) ∧ (c_{s[v]} > L_{s[v]}) ∧ (c_g < U_g) then
                    Δf = M[v][g] − M[v][s[v]];
                    if Δf > 0 then
                        c_{s[v]} = c_{s[v]} − 1;
                        c_g = c_g + 1;
                        s[v] = g;
                        f(s) = f(s) + Δ;
                        Update M;
                        imp = true;
                    end
                end
            end
        end
        for v = 1 to n − 1 do
            for u = v + 1 to n do
                if s[v] ≠ s[u] then
                    Δf = M[v][s[u]] − M[v][s[v]] + M[u][s[v]] − M[u][s[u]] − 2d_{vu};
                    if Δf > 0 then
                        t = s[v];
                        s[v] = s[u];
                        s[u] = t;
                        f(s) = f(s) + Δ;
                        Update M;
                        imp = true;
                    end
                end
            end
        end
    end
    return s, f(s);
end
```

N_1 neighbor solutions of S^f, and the set of these neighbor solutions constitutes the N_1 neighborhood of S^f. It should be noted that since $L_g = U_g$ for each group in instances with equal group sizes (EGS), the solutions in these instances do not have N_1 neighborhood.

S^n is used to denote a newly generated solution of S^f after one move of N_1 neighborhood search. In order to calculate the objective value of the S^n efficiently, I apply the $n \times m$ matrix M mentioned above. After the N_1 neighborhood search, the diversity values of all groups are unchanged, except for the group g_1, which eliminates one element, and the group g_2, which accepts one element. The difference Δf between the objective values of S^n and S^f is only related to the two changed groups. The diversity value of group g_1 loses the sum of the diversity between element v and all other elements in group g_1, while the diversity value of group g_2 gains the sum of the diversity between element v and the other elements in the group g_2. Therefore, the difference Δf between the objective values of S^n and S^f can now be easily calculated as:

$$\Delta f = f(S^n) - f(S^f) = M[v][g_2] - M[v][g_1] \tag{4.2}$$

If $\Delta f > 0$, it means that the quality of the solution is improved after an insertion move. Then, the element v is moved from g_1 to g_2 and the matrix M will be immediately updated in the following way:

$$\begin{aligned} M[j][g_1] &= M[j][g_1] - d_{jv} \\ M[j][g_2] &= M[j][g_2] + d_{jv} \end{aligned} \tag{4.3}$$

where j is each element belonging to V ($V = \{1, 2, ..., n\}$). Hence, the complexity of updating the matrix M is $O(N)$.

N_2 represents another commonly used neighborhood, i.e., swap neighborhood. For a feasible solution S^f, suppose that elements v and u locate in groups g_v and g_u, respectively. N_2 neighborhood of S^f is composed of solutions obtained by swapping a single pair of elements belonging to different groups. That is to say, let element v be in group g_u and element u be in group g_v. It is clear that, unlike the N_1 neighborhood, the group size of the solution after the N_2 neighborhood search does not change. Therefore, the swap neighborhood applies to the EGS instances as well.

S^n is used to denote an N_2 neighbor solution of S^f. Like the N_1 neighborhood search, the difference Δf between the objective values resulting from the swap move is only relevant to the groups g_v and g_u. For the element v, it is removed from g_v, the diversity of g_v decreases by the sum of diversity between element v and other elements in g_v. At the same time, group g_u receives the element v, the diversity of group g_u increases by the sum of the diversity between it and the other elements in g_u. It is the same process for element u. Thus, we can find that Δf between $f(S^n)$ and $f(S^f)$ can be calculated as:

$$\Delta f = f(S^n) - f(S^f) = (M[v][g_u] - M[v][g_v]) + (M[u][g_v] - M[u][g_u]) - 2d_{vu} \qquad (4.4)$$

I use the above two neighborhood structures to perform a local search for improving the quality of solutions in the population. The double-neighborhood local search method explores both the N_1 and N_2 neighborhoods in a deterministic and token-ring way ($N1 \rightarrow N2 \rightarrow N1 \rightarrow N2 \rightarrow N1 \rightarrow N2, ...$). Moreover, the neighborhood solutions are accessed in a lexicographical order way when detecting the neighborhoods. Given a current solution S^f, the local search starts with exploration in the N_1 neighborhood of S^f. Once an improved neighbor solution S^n is found, S^n is taken as the current solution S^f, and the process continues to examine neighbor solutions within the N_1 neighborhood of the new current solution S^f. This process is repeated until $n \times m$ neighbor solutions are detected and then transferred to the N_2 neighborhood. The search behavior in the N_2 neighborhood is the same as that in the N_1 neighborhood, with the difference that the search shifts to the N_1 neighborhood after $n \times (n-1)/2$ neighbor solutions have been visited. The double-neighborhood local search is performed until no improved solution is found in both N_1 and N_2 neighborhoods.

4.4 Undirected perturbation phase

The proposed TPSDP algorithm employs an undirected perturbation operator derived from [32] for each input solution, as shown in Algorithm 9. The undirected perturbation proce-

Algorithm 9: Undirected Perturbation

```
Function UndirectedPerturbation(s, η_s)
    s_p = s;
    for η = 1 to η_s do
        s_p = RandomSolution(N_1(s_p) ∪ N_2(s_p));
    end
    return s_p;
end
```

dure randomly picks a neighbor solution from N_1 or N_2 neighborhood to replace the current solution η_s times without considering the objective value of the neighbor solution, where η_s represents the strength of undirected perturbation and $\eta_s = \theta \times \frac{n}{m}$. Different from the way that fixing the η_s during the whole search process, in this study, I set the value of θ adaptively, dropping from 2.0 to 1.0 for instances with $n > 400$ and 1.2 to 0.1 for the remaining instances. According to Eq. (4.5), θ_{max} and θ_{min} are maximum and minimum coefficient values for the strength of strong perturbation, respectively. $Time()$ and t_{max} are current time and maximum time, respectively. Due to the property of the undirected perturbation operator, this process increases the diversity of solutions in the population. It strongly modifies a solution to jump out of the current local optimal region and relocates it to a more distant region. Afterward, the double-neighborhood local search follows the undirected perturbation operator to find the local optimal solutions in new regions.

$$\theta = \theta_{max} - (\theta_{max} - \theta_{min}) * \frac{Time()}{t_{max}} \tag{4.5}$$

4.5 Population reconstruction phase

The population reconstruction phase consists of two ingredients: the offspring generation and the replacement strategy.

4.5.1 The offspring generation method

After the undirected perturbation phase, each solution S^i in the population is characterized by high quality. In order to use valuable parts of solutions to guide the next search, I propose

Algorithm 10: Crossover operator

Function `Crossover`(p_1, p_2)

- $S^o = \emptyset$;
- $G = \{1, 2, \ldots, m\}$;
- $H = \{1, 2, \ldots, n\}$;
- **for** $i = 1$ **to** m **do**
  - **if** $r < 0.5$ **then**
    - Select a group g' with maximum diversity from p_1;
  - **else**
    - Select a group g' with maximum diversity from p_2;
  - **end**
  - $AG = \{g | U_g \geq c_{g'}, g \in G\}$;
  - **if** $AG \neq \emptyset$ **then**
    - Randomly select a group g from AG;
    - put the elements of g' into group g of S^o;
  - **else**
    - Select a group g that U_g is cloest to $c_{g'}$;
    - randomly selecte U_g elements from g' and put them into group g of S^o;
  - **end**
  - $G = G \setminus g$;
  - Remove all elements of g from p_1,p_2, and H;
- **end**
- Assign each remained element from H randomly to a random group g of S^o whose $c_g < L_g$;
- Assign each remained element from H randomly to a random group g of S^o whose $c_g < U_g$;
- **return** S^o;

end

a new crossover operator, as shown in Algorithm 10, to generate offspring solutions, which plays a crucial role in TPSDP. The newly proposed crossover operator extends the former used in [33] to make it more general, which can be applied not only to exceptional cases like the EGS instances but also to the DGS instances in MDGP. To be specific, each solution S^i in the population is selected as a parent solution (say p_1), and the other different parent solution (say p_2) is chosen randomly from the current population. These two solutions are used to generate a child solution according to the proposed crossover operator. It should be noticed that the population reconstruction phase is performed only when the current population size β is greater than 1.

At the beginning of the crossover process, I first prepare two duplicate solutions (p_1 and p_2) of the parent solutions, a set H containing the elements of V, and an empty offspring solution S^o. Then, I select p_1 and p_2 with equal probability and define the selected one as p'. The group with the largest diversity in p' is picked as a candidate group g', which is an important component of p'. To retain as many elements in g' as possible, priority is given to select those empty groups in S^o whose upper bound is greater than the number of elements in g'. An empty group with the above property is randomly selected among S^o, and the elements of g' are put in it. If such an empty group does not exist, find an empty group whose U_g is closest to the number of elements in g', and fill the group with randomly select U_g elements from g'. The elements that have been assigned are removed from the p_1, p_2, and H. Then, the diversity of each group in the parent solutions is recalculated. The above steps are performed m times. Note that there is a high probability that the constructed offspring solution S^o is still illegal, which means that the number of elements in some groups has not reached their lower bound L_g, or there are still elements in H that have not been assigned yet. In this situation, an adjustment process is implemented as in the following.

I first calculate whether the number of unassigned elements in H can satisfy L_g of m groups in S^o. In the first case, if the number of unassigned elements in H is sufficient, a random unassigned element is assigned to a random group whose group size has not yet reached L_g in S^o. Repeat this process until all m groups satisfy its L_g. The remaining elements are randomly assigned to those groups whose group sizes have not yet reached their upper bound U_g. Repeat this step until all n elements are allocated into S^o. In the second case, if the number of unassigned elements in H is insufficient, randomly selected an element from a group whose current group size is larger than L_g and put it in H. Repeat this process until the number of unassigned elements can satisfy L_g of m groups in S^o. The following element assignment process is the same as the first case.

Figure 4.1 illustrates the process of the proposed crossover operator on a given example. Suppose that there are 12 elements (i.e., $n = 12$) and 4 groups (i.e., $m = 4$) with given upper and lower bounds. At step 1, p_1 is selected as p', group {5, 6, 7, 8} with the largest diversity is chosen to become the candidate group g'. To retain as many elements in g' as possible, empty group g_4 in S^o whose U_g is greater than $c_{g'}$ is randomly selected. Put the elements 5, 6, 7, 8 in g_4 and remove them from p_1, p_2, and H. Similarly, I build g_2, g_1, and g_3 of S^o from parents p_2 and p_1, respectively. At the end of these four steps, g_3 of S^o is the only one group whose current size is smaller than its L_g. The unassigned element 3 in H is picked randomly and placed it in g_3. Removing element 3 from p_1, p_2, and H. The remaining element 4 in H is then put in g_2 where c_{g_2} is not reached U_{g_2}.

Every time S^o is constructed, the double-neighborhood local search will be performed to find the corresponding local optimal solution $S^{o'}$. The composition of the offspring solutions derived by using this crossover operator will not produce overlapping parts of elements in groups, which maintains the integrity of the elements in the groups of parent solutions as much as possible, thus ensuring the quality of the offspring solutions to a certain extent. On the other hand, this process not only maintains the diversity of the population but also enhances the intensification of the algorithm at the same time.

4.5.2 Replacement strategy

Whether $S^{o'}$ can replace the corresponding parent solution p_1 into the new population depends on the fitness value and the structure of $S^{o'}$. When $S^{o'}$ has a larger fitness value than the corresponding parent solution p_1, $S^{o'}$ directly replaces p_1 into the new population. Otherwise, in order to maintain the diversity of the population, I retain the deteriorating offspring $S^{o'}$ if the following condition is satisfied:

$$\frac{f(S^{o'})}{f(p_1)} + \alpha \times Dis(p_1, S^{o'}) > 1 \tag{4.6}$$

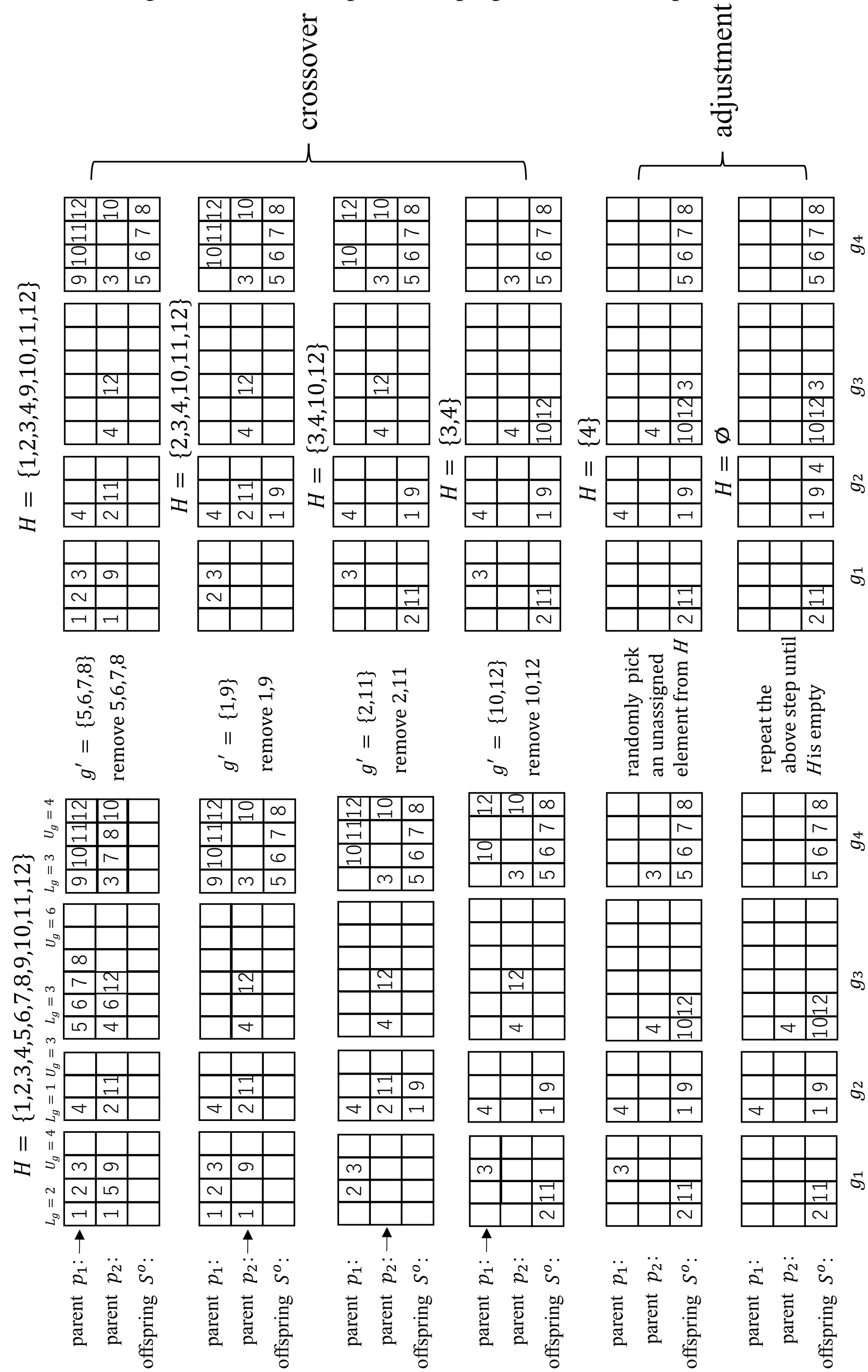
where $f(S^{o'})$ and $f(p_1)$ denote the objective values of the offspring solution $S^{o'}$ and the corresponding parent solution p_1, respectively. α is a parameter, taking a value of 0.05 after detailed testing (see Section 6.1.2). $Dis(p_1, S^{o'})$ indicates the distance between p_1 and $S^{o'}$. Generally, the distance between two solutions (S^1, S^2) is defined in the following way:

$$Dis(S^1, S^2) = \frac{\left|\left\{(i,j) : \left(\left(g^1[i] == g^1[j]\right) \wedge \left(g^2[i] \neq g^2[j]\right)\right) \vee \left(\left(g^1[i] \neq g^1[j]\right) \wedge \left(g^2[i] == g^2[j]\right)\right)\right\}\right|}{n^2/m} \tag{4.7}$$

which estimates the "fraction" of pairs locating in the same group in one solution, but not in the same group in the other solution, as described in [48].

Note that, the way I use the Eq. (4.6) is quite different from [33], which is applied to judge whether a local search for a child solution is necessary. I use the formula to discriminate whether to accept a slightly worse local optimal offspring solution.

Figure 4.1: An example of the proposed crossover process.



4.6 Directed perturbation phase

To further improve the quality of solutions after the reconstruction phase, a directed perturbation phase, which consists of a directed perturbation operator and the double-neighborhood local search, is adopted as shown in Algorithm 11. For each solution, I first initialize an $m \times m$ matrix Avg, a $1 \times m$ matrix R, and an empty set U to record the average diversity contribution of selected m elements to the m groups, the elements that should be reassigned, and the groups whose group size is less than L_g after removing an element, respectively. Then, for each group, an element who contributes the least diversity value is taken out and put into R. After that, if the group size c_g is less than L_g, the group g will be put into the set U. In order to eliminate the influence caused by the different number of elements in each group on DGS instances (e.g., the element generates more diversity contribution concerning those groups with a larger number of elements than those groups with a smaller number of elements), the average diversity contribution of the elements to the groups is calculated. Eq. (4.8) initializes the average diversity contribution of each element to each group, where M which is defined in Eq. (4.1) represents the sum of the diversity between each element and all elements in the group g, D is the matrix of the diversity between two elements, c_g denotes the current number of elements of group g.

$$\begin{aligned} M[R[k]][g] &= M[R[k]][g] - D[R[k]][R[g]] \\ Avg[k][g] &= \frac{M[R[k]][g]}{c_g}, k = 1, 2, \ldots, m; g = 1, 2, \ldots, m \end{aligned} \tag{4.8}$$

In the process of element reassignment, I prioritize the groups in set U to ensure the feasibility of the solution. Specifically, a group g is randomly selected in U first. Then, in terms of matrix Avg, an element in R possesses the greatest contribution to the group g is placed into this group. Once an element is assigned, the matrix Avg, R, M and U will be updated immediately. The update process starts by removing the assigned element from R and removing the group g from U. Then the average contribution of this element to all groups is automatically changed to 0, and the average contribution of the remaining

Algorithm 11: Directed Perturbation

```
Function DirectedPerturbation(s, η_d)
    for L = 1 to η_d do
        Avg = [m][m], R = [m], U = ∅, G = {1, 2, ..., m};
        for g = 1 to m do
            Find an element i with the lowest diversity contribution in group g;
            R[g] = i;
            c_g = c_g − 1;
            if c_g < L_g then
                U = U ∪ {g};
            end
        end
        Initialize Avg according to Eq. (4.8);
        while U ≠ ∅ do
            Randomly select a group g_r in U;
            In terms of Avg, find an element R[e] with the largest diversity
              contribution to g_r;
            c_{g_r} = c_{g_r} + 1;
            for k = 1 to m do
                if k ∈ G then
                    M[R[k]][g_r] = M[R[k]][g_r] + D[R[k]][R[e]];
                    Avg[k][g_r] = (M[R[k]][g_r] − D[R[k]][R[g_r]])/c_g;
                end
            end
            Set the e^th row of Avg to be 0;
            s[R[e]] = g_r;
            U = U \ {g_r};
            G = G \ {e};
        end
        while G ≠ ∅ do
            Randomly select a number e in G;
            Find a group g* whose size less than U_g, and R[e] can bring the largest
              diversity contribution to g*;
            c_{g*} = c_{g*} + 1;
            for k = 1 to m do
                if k ∈ G then
                    M[R[k]][g*] = M[R[k]][g*] + D[R[k]][R[e]];
                    Avg[k][g*] = (M[R[k]][g*] − D[R[k]][R[g*]])/c_{g*};
                end
            end
            Set the e^th row of Avg to be 0;
            s[R[e]] = g*;
            G = G \ {e};
        end
    end
    return s;
end
```

unallocated elements to each group is updated. This step is repeated until U is empty. If there are still elements in R, one element is selected at random, and the group with the highest average contribution is found according to the matrix Avg. The element is added to the group if the number of elements in that group does not reach U_g. The following update process of matrix Avg is the same as the above. This step is repeated until R has no more elements. The above procedure successive runs η_d times to obtain a perturbed solution. Finally, a local search process is performed for this perturbed solution to exploits the local optimum.

This directed perturbation operator constructs a slightly perturbed solution and preserves the quality of the current solution as much as possible. Instead of the random search of the undirected perturbation, the search process with directed perturbation is less destructive to the solution and locates the solution in a neighborhood closer to the current solution. This phase can be regarded as the intensification phase of the algorithm in the solution space.

4.7 Linear decline of population

The local search in the TPSDP algorithm is applied to every solution in the population, which is very time-consuming. In order to use the limited computing resources efficiently, I reserve more resources for those more promising solutions by decreasing the population size, shown as:

$$\beta = (\beta_{min} - \beta) * \frac{Time()}{t_{max}} + \beta \tag{4.9}$$

where β and β_{min} are current population size and minimum population size, respectively. $Time()$ and t_{max} are current time and maximum time, respectively.

Chapter 5

Experimental result and comparison

5.1 Experimental setup

To verify the performance of the proposed TPSDP, I test it on different scale benchmark instances and make comparisons with four state-of-the-art algorithms including ITS [49], IMS [32], NSGGA [33], and NDHA [50]. Among these reference algorithms, the source code of ITS, IMS, and NDHA can be obtained from `https://www.personalas.ktu.lt/~ginpalu/`, `http://www.info.univ-angers.fr/~hao/mdgp.html`, `http://www.info.univ-angers.fr/pub/hao/NDHA.html`, respectively. It is worth mentioning that the proposed TPSDP algorithm as well as ITS, IMS, NDHA have been implemented in the C++ language. Moreover, all of the experiments of the four algorithms were carried out under the same computing platform, a Windows PC with a configuration of Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz 8.00GB RAM. Each algorithm was run 20 times for an instance to obtain statistical results. Regrading NSGGA, I directly use the reported experimental results in [33] as the comparison data.

5.2 Benchmark instances

I have used the same benchmark instances for our algorithm that have been widely used in other algorithms for MDGP in the literature, including three small-scale sets and two large-scale sets. Three small-scale sets and one large-scale set can be found on the website: `https://grafo.etsii.urjc.es/optsicom/mdgp/`, and the rest can be downloaded from https://grafo.etsii.urjc.es/optsicom/mdp/. Next, the characteristics of each benchmark set are described in detail.

RanInt set: In this set, there are four kinds of instances with different vertex numbers, ranging from 120 to 960. Each kind has ten EGS instances and ten DGS instances, where the distances or diversities between the pairs of elements are generated with an integer uniform distribution in the interval (0, 100). For these instances, the number of groups m varies from 10 to 24, while the lower and upper bounds are between 2 and 48. Meanwhile, the EGS instances have the same lower and upper bounds $[n/m]$ for any group g.

RanReal set: This set has the same structure and size as RanInt. The only difference is the distances d_{ij} between points which are real numbers generated using a uniform distribution between 0 and 100.

Geo set: The main feature of this set is the distances between two elements, which are calculated as Euclidean distances between pairs of elements with coordinates randomly generated in [0,10]. The number of coordinates for each element is created randomly in the 2 to 21 range. The structure and size are similar to RanInt and RanReal.

MDG-a set: This set has some dissimilarities from the above sets, which is a large-scale set with $n = 2000$, and consists of 11 types of instances, including six types of DGS instances and five types of EGS instances. Each type comprises 20 instances, and the distances of pairs of points are the same as RanInt, except that the interval is 0 to 10. The specific characteristics are listed in Table 5.1 below.

Table 5.1: Main information of the instances in MDG-a set.

		DGS		**EGS**
n	m	L_g	U_g	$L_g = U_g$
2000	50	32	48	-
2000	10	173	227	200
2000	25	51	109	80
2000	50	26	54	40
2000	100	13	27	20
2000	200	6	14	10

MDG-c set: The set that adapted from the instances of maximum diversity problem is a new set, which is used in [32, 50] merely. Two types of instances belong to this set. One of them has 20 DGS instances with $n = 3000$ and $m = 50$, where the lower and upper bounds of group sizes are fixed to $[0.8n/m]$ and $[1.2n/m]$, respectively. The other possesses 20 EGS instances with the same numbers of vertices and groups as the DGS instance, except that the group sizes are set to $[n/m]$. The edge weights d_{ij} are integers randomly generated from 0 to 1000.

5.2.1 Parameter setting

Table 5.2: Setting of parameters.

Parameters	Section	Description	Value
β_{max}	4.2	maximum (initial) population size	15
β	4.7	current population size	time-variant
β_{min}	4.7	minimum (final) population size	{1, 2}
η_s	4.4	strength of undirected perturbation	$\theta \times \frac{n}{m}$
θ	4.4	coefficient for strength of undirected perturbation	time-variant
θ_{max}	4.4	the maximum value of θ	{1.2, 2}
θ_{min}	4.4	the minimum value of θ	{0.1, 1}
α	4.5.2	a parameter of replacement strategy	0.05
η_d	4.6	strength of directed perturbation	3

This section states some necessary parameter settings of the TPSDP algorithm, which is summarized in Table 5.2. β_{max}, β, and β_{min} are three parameters with respect to the population size. It is worth noting that β is a time-varying parameter because the population size in TPSDP decreases with time. Also, notice that the parameter θ, a crucial coefficient determining the strength of undirected perturbation η_s, declines from 1.2 to 0.1 over time for the instances with $n \leq 400$ and from 2.0 to 1.0 for the other instances. The parameter η_d used in the directed perturbation operator is set to 3 applicable to all scale instances. All the parameter values presented in Table 5.2 are employed in all the following experiments reported in this work as the default value, although some parameters can be fine-tuned to produce better results for some instances. Parameters of ITS and IMS follows the recommendation setting in the literature [32, 49].

In this paper, I choose the same termination condition as [32, 33, 50], that is, the termination condition for all the above algorithms is the cutoff time limit t_{max}, which is related to the size of the instances. The specific settings are: $t_{max} = 3$ seconds for $n = 120$, $t_{max} =$

20 seconds for $n = 240$, $t_{max} = 120$ seconds for $n = 480$, $t_{max} = 600$ seconds for $n = 960$, $t_{max} = 1200$ seconds for $n = 2000$, and $t_{max} = 3000$ seconds for $n = 3000$.

5.3 Experimental results and comparison on five benchmark sets

In this section, I present the experimental results obtained by TPSDP, ITS, IMS, NSGGA, and NDHA, and make comparisons to evaluate the performance of the proposed TPSDP. Tables 5.3-5.8 summarize the computational results of ITS, IMS, NDHA, and TPSDP on RanInt, RanReal, and Geo beachmark instances. The data comparison outcomes with NSGGA on these three benchmark sets are summarized in Tables 5.9-5.11. Moreover, I also evaluate the performance of TPSDP on MDG-a and MDG-c benchmark sets, which were tested in [32, 50]. Refer to Tables 7.1-7.13 for detailed experimental results of all MDG-a and MDG-c instances. The first column of the tables states the names and some information of the experimental instances. The data listed in columns f_{best} and columns f_{avg} record the best and average values of 20 independent runs on each instance, respectively. For each instance, the best result among all compared algorithms is shown in bold. The row 'Avg' reflects the average value of each column, and the row '#Best' presents the total number of the best values obtained by each algorithm in the comparison. In the last row of the table, p-value is obtained by the Wilcoxon signed-rank test to verify whether there is a significant difference between TPSDP and comparison algorithms in terms of f_{best} and f_{avg}.

Tables 5.3 and 5.4 report the experimental results of ITS, IMS, NDHA, and TPSDP on RanInt instances. On the DGS instances, TPSDP outperforms its peers in terms of f_{best} and f_{avg}, obtaining the best result on 26 and 36 out of 40 instances while ITS, IMS and NDHA obtain 6, 5, 3 and 1, 1, 2 best results, respectively. On the EGS instances, TPSDP is also ahead of ITS, IMS, and NDHA in both f_{best} and f_{avg}, and TPSDP produces 26 and 23 best results on 40 instances based on f_{best} and f_{avg}, respectively. Furthermore, the p-value smaller than 0.05 also shows that TPSDP is significantly better than reference algorithms on both DGS and EGS instances. Additionally, these results show that TPSDP works better on instances with $n = 960$ and performs more stable on instances with different group sizes than those with equal group sizes.

Tables 5.5 and 5.6 report the experimental results of ITS, IMS, NDHA, and TPSDP on RanReal instances. Results show that the performance of TPSDP on the RanReal instances is similar to that on the RanInt instances. On the DGS instances, ITS, IMS, NDHA, and TPSDP achieve 2, 7, 1, and 30 best results in terms of f_{best}, and 0, 1, 2, and 37 best values in terms of f_{avg}, respectively. On the equal group sizes instances, TPSDP is also superior to the comparison algorithms based on f_{best} and f_{avg}, and the small p-value obtained by the Wilcoxon signed-rank test confirms that TPSDP is significantly better than ITS, IMS, and NDHA on both DGS and EGS RanReal instances.

The experimental results of ITS, IMS, NDHA, and TPSDP on Geo instances are summarized in Tables 5.7 and 5.8. For the DGS instances, the p-value reveals that TPSDP performs comparably with ITS and NDHA in terms of f_{avg}. For the EGS instances, clearly, ITS shows an overwhelming advantage over TPSDP, IMS, and NDHA.

Tables 5.9-5.11 show the results of comparing NSGGA and TPSDP on the three types of instances with equal group sizes. From Tables 5.9 and 5.10, we can see that although the p-value based on f_{best} is greater than 0.05, TPSDP finds more best f_{best} than NSGGA on both EGS and DGS. Furthermore, the p-values based on f_{avg} show that TPSDP is significantly better than NSGGA with respect to stability. While on the Geo instances, NSGGA and TPSDP get the best values on 6 and 34 instances in terms of f_{best}, and the best values on 4 and 36 instances in terms of f_{avg}, respectively. Based on the p-value, it is also clear that TPSDP significantly outperforms NSGGA on Geo instances.

It can be seen from Tables 7.1-7.13 that TPSDP significantly outperforms ITS and IMS, but performs worse than NDHA on MDG-a and MDG-c sets. Except for one DGS instance set, TPSDP can find better results compared to ITS and IMS on all instance sets in terms of both the best and average objective values. It is worth pointing out that, although TPSDP performs worse than NDHA averagely, it significantly outperforms NDHA on the EGS instances with $n = 2000$, $m = 100$, and $n = 2000$, $m = 200$ (Tables 7.10 and 7.11).

To more precisely assess the overall performance of ITS, IMS, NDHA, and the proposed TPSDP, the statistical results via the Friedman test based on the average experimen-

Table 5.3: Comparison of the TPSDP algorithm with three best performing algorithms on the 40 DGS RanInt instances.

	f_{best}				f_{avg}			
Instance	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
RanInt_n120_ds_01	**51161.00**	51112.00	51138.00	51146.00	51004.70	50907.15	51031.00	**51075.85**
RanInt_n120_ds_02	**51441.00**	51404.00	51387.00	51372.00	51322.65	51317.65	**51322.85**	51294.65
RanInt_n120_ds_03	50215.00	50245.00	**50270.00**	50248.00	50140.00	50172.20	50144.45	**50192.05**
RanInt_n120_ds_04	50429.00	50407.00	50404.00	**50436.00**	50276.20	50325.35	50320.30	**50343.85**
RanInt_n120_ds_05	49872.00	49985.00	49977.00	**50008.00**	49573.00	49791.95	49806.80	**49839.20**
RanInt_n120_ds_06	49734.00	49589.00	49667.00	**49767.00**	49556.40	49512.35	49557.25	**49602.70**
RanInt_n120_ds_07	**50306.00**	50295.00	50281.00	50282.00	49966.60	50188.10	50188.10	**50202.85**
RanInt_n120_ds_08	**50434.00**	50370.00	50415.00	50385.00	**50290.60**	50282.50	50267.50	50282.30
RanInt_n120_ds_09	**50499.00**	50375.00	50461.00	50451.00	50319.25	50304.25	50322.25	**50363.10**
RanInt_n120_ds_10	50325.00	50398.00	50390.00	**50407.00**	50218.10	50271.75	**50303.70**	50269.30
RanInt_n240_ds_01	160371.00	160454.00	**160661.00**	160596.00	159927.40	160317.35	160082.05	**160358.10**
RanInt_n240_ds_02	160211.00	160257.00	159989.00	**160468.00**	159724.25	159905.95	159796.75	**160277.70**
RanInt_n240_ds_03	160239.00	160257.00	160300.00	**160400.00**	159899.10	160092.90	159937.85	**160223.05**
RanInt_n240_ds_04	**162728.00**	162525.00	162488.00	162619.00	161836.35	162338.95	162110.40	**162420.25**
RanInt_n240_ds_05	160543.00	160732.00	160307.00	**160841.00**	160296.45	160393.15	160061.45	**160605.25**
RanInt_n240_ds_06	161020.00	161138.00	160962.00	**161334.00**	160645.15	160925.75	160614.90	**161040.55**
RanInt_n240_ds_07	160109.00	160256.00	160130.00	**160412.00**	159587.40	159878.80	159612.15	**160131.20**
RanInt_n240_ds_08	158161.00	158046.00	157990.00	**158321.00**	157736.40	157868.90	157546.45	**157980.90**
RanInt_n240_ds_09	160636.00	160707.00	160430.00	**160799.00**	160182.10	160449.40	160229.30	**160601.10**
RanInt_n240_ds_10	160301.00	**160316.00**	160234.00	160299.00	159595.50	159989.70	159835.75	**160082.30**
RanInt_n480_ds_01	390089.00	390642.00	**391214.00**	390718.00	388985.05	390124.25	389860.25	**390362.25**
RanInt_n480_ds_02	388587.00	**389439.00**	388638.00	389327.00	387425.90	**388783.45**	387951.60	388743.45
RanInt_n480_ds_03	388457.00	388808.00	387869.00	**389098.00**	387020.45	388077.00	387383.25	**388362.50**
RanInt_n480_ds_04	391882.00	392160.00	392275.00	**392628.00**	390850.40	391702.00	391440.00	**391846.30**
RanInt_n480_ds_05	389639.00	**390151.00**	389139.00	389981.00	388078.30	389183.70	388520.65	**389412.40**
RanInt_n480_ds_06	389192.00	**390209.00**	389448.00	390088.00	388288.40	389377.00	388779.60	**389520.60**
RanInt_n480_ds_07	388722.00	389817.00	389936.00	**390181.00**	388045.55	389109.00	388421.70	**389347.25**
RanInt_n480_ds_08	390599.00	391143.00	390112.00	**391339.00**	389880.90	390561.35	389508.95	**390647.10**
RanInt_n480_ds_09	388224.00	389095.00	388364.00	**389116.00**	387415.80	388433.90	387496.45	**388567.05**
RanInt_n480_ds_10	393100.00	393993.00	393543.00	**394099.00**	391335.10	393413.45	392942.90	**393694.35**
RanInt_n960_ds_01	1239428.00	1243806.00	1243362.00	**1244347.00**	1237394.90	1242271.70	1241600.70	**1242857.55**
RanInt_n960_ds_02	1238326.00	1241678.00	1240757.00	**1242006.00**	1235097.80	1240288.20	1239093.05	**1240869.75**
RanInt_n960_ds_03	1237944.00	1241600.00	1241494.00	**1242461.00**	1235989.15	1239866.85	1239749.55	**1240896.00**
RanInt_n960_ds_04	1239235.00	1241705.00	1242837.00	**1243122.00**	1236199.80	1240629.55	1239765.15	**1241778.60**
RanInt_n960_ds_05	1237695.00	1240913.00	1240818.00	**1241729.00**	1236010.60	1239506.80	1239388.40	**1240418.35**
RanInt_n960_ds_06	1235240.00	1238838.00	1237770.00	**1239217.00**	1233381.55	1237157.00	1236371.40	**1238029.70**
RanInt_n960_ds_07	1239372.00	**1242947.00**	1241256.00	1242811.00	1236525.70	1241470.40	1239950.20	**1242246.30**
RanInt_n960_ds_08	1234859.00	1238152.00	1237880.00	**1239231.00**	1233181.35	1237022.85	1236461.75	**1237830.75**
RanInt_n960_ds_09	1235805.00	1239264.00	1239270.00	**1240150.00**	1233212.20	1237940.55	1237314.60	**1238889.65**
RanInt_n960_ds_10	1238526.00	1240861.00	1241881.00	**1242428.00**	1236035.15	1240015.85	1239162.65	**1241035.10**
Avg.	459591.40	460602.225	460393.60	**460866.70**	458561.29	460004.22	459606.35	**460313.53**
#Best	6	5	3	**26**	1	1	2	**36**
p-value	0.000001	0.00002	0.000001		0	0	0	

Table 5.4: Comparison of the TPSDP algorithm with three best performing algorithms on the 40 EGS RanInt instances.

	f_{best}				f_{avg}			
Instance	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
RanInt_n120_ss_01	**47909.00**	**47909.00**	**47909.00**	**47909.00**	47898.95	47871.30	47903.25	**47909.00**
RanInt_n120_ss_02	**47826.00**	**47826.00**	**47826.00**	**47826.00**	47810.15	47763.25	47782.35	**47822.80**
RanInt_n120_ss_03	**47552.00**	**47552.00**	**47552.00**	**47552.00**	**47508.05**	47445.00	47451.55	47489.10
RanInt_n120_ss_04	**47611.00**	47515.00	**47611.00**	47547.00	**47536.05**	47472.60	47488.80	47518.05
RanInt_n120_ss_05	**47210.00**	47142.00	**47210.00**	**47210.00**	47131.25	47045.75	47111.90	**47137.30**
RanInt_n120_ss_06	**46647.00**	46585.00	**46647.00**	**46647.00**	46606.95	46524.75	46552.05	**46617.75**
RanInt_n120_ss_07	**47142.00**	47136.00	47136.00	**47142.00**	47115.20	47056.45	47062.75	**47122.30**
RanInt_n120_ss_08	**47390.00**	47356.00	47374.00	**47390.00**	**47372.45**	47319.80	47327.80	47357.55
RanInt_n120_ss_09	**47660.00**	47654.00	**47660.00**	**47660.00**	**47642.90**	47602.00	47617.20	47635.10
RanInt_n120_ss_10	**47807.00**	**47807.00**	**47807.00**	**47807.00**	**47802.55**	47766.50	47773.00	47798.15
RanInt_n240_ss_01	155566.00	155400.00	155442.00	**155577.00**	155310.30	155288.30	155210.20	**155440.20**
RanInt_n240_ss_02	155378.00	155302.00	155142.00	**155384.00**	155098.50	155104.70	154891.10	**155207.25**
RanInt_n240_ss_03	156398.00	**156415.00**	156195.00	**156415.00**	156166.55	156246.70	155999.70	**156319.40**
RanInt_n240_ss_04	156527.00	156564.00	156552.00	**156643.00**	156370.65	156407.50	156244.10	**156513.55**
RanInt_n240_ss_05	156509.00	156522.00	156466.00	**156562.00**	156221.40	**156320.45**	156057.90	156295.70
RanInt_n240_ss_06	155564.00	155594.00	155346.00	**155601.00**	155248.15	155270.10	155047.40	**155402.40**
RanInt_n240_ss_07	155736.00	155707.00	155609.00	**155791.00**	155482.50	155529.40	155309.05	**155678.40**
RanInt_n240_ss_08	**155305.00**	155297.00	155076.00	155297.00	154965.95	155039.80	154835.00	**155167.95**
RanInt_n240_ss_09	**156043.00**	156011.00	156011.00	**156043.00**	**155971.90**	155960.50	155865.10	155923.40
RanInt_n240_ss_10	155890.00	155952.00	155916.00	**155971.00**	155691.20	155740.80	155611.65	**155854.20**
RanInt_n480_ss_01	379501.00	379927.00	379131.00	**379953.00**	378835.30	**379370.55**	378735.85	379263.95
RanInt_n480_ss_02	379733.00	**380287.00**	379978.00	380180.00	378902.70	**379665.60**	379136.80	379596.55
RanInt_n480_ss_03	378511.00	**379303.00**	378690.00	378762.00	377888.45	**378573.15**	377974.65	378291.55
RanInt_n480_ss_04	378979.00	**379222.00**	378726.00	379008.00	378178.45	**378712.90**	378247.10	378628.00
RanInt_n480_ss_05	379627.00	379878.00	378883.00	**379883.00**	378692.55	**379207.30**	378485.95	379132.55
RanInt_n480_ss_06	**379492.00**	379313.00	378971.00	379354.00	378519.05	**378852.50**	378201.45	378835.95
RanInt_n480_ss_07	379474.00	**380464.00**	379606.00	379741.00	378685.95	**379363.50**	378754.45	379151.40
RanInt_n480_ss_08	379542.00	**380162.00**	379793.00	380161.00	378821.25	**379372.40**	378763.20	379341.55
RanInt_n480_ss_09	378708.00	**379065.00**	378585.00	379060.00	378016.15	**378521.05**	377894.15	378410.25
RanInt_n480_ss_10	380185.00	**380446.00**	379904.00	380248.00	379322.60	**379892.15**	379254.80	379813.25
RanInt_n960_ss_01	1218585.00	1219991.00	**1222878.00**	1220742.00	1216150.40	1218944.85	1219598.50	**1219694.55**
RanInt_n960_ss_02	1216333.00	1219901.00	1219988.00	**1220325.00**	1215087.40	1218566.25	1218353.30	**1219184.35**
RanInt_n960_ss_03	1217811.00	1220499.00	1220514.00	**1221283.00**	1216240.95	1219081.55	1219059.70	**1220117.65**
RanInt_n960_ss_04	1217737.00	1220842.00	1220381.00	**1220857.00**	1216224.10	1219226.10	1218918.05	**1219949.50**
RanInt_n960_ss_05	1216725.00	1219942.00	**1221149.00**	1220702.00	1215371.95	1218616.60	1218846.45	**1219607.40**
RanInt_n960_ss_06	1218728.00	1220880.00	**1221839.00**	1221066.00	1216392.80	1219567.75	1219645.30	**1220231.30**
RanInt_n960_ss_07	1218772.00	1220664.00	1220741.00	**1221650.00**	1216399.05	1219915.20	1219469.35	**1220481.15**
RanInt_n960_ss_08	1218247.00	1220515.00	1221006.00	**1221474.00**	1216530.40	1219550.20	1219689.85	**1220252.95**
RanInt_n960_ss_09	1216033.00	1218758.00	**1220074.00**	1219501.00	1214384.10	1217830.65	1218235.55	**1218535.95**
RanInt_n960_ss_10	1217302.00	1218473.00	1218444.00	**1219430.00**	1214593.45	1217506.05	1217296.45	**1218293.50**
Avg.	450092.38	450794.45	450794.20	**450933.85**	449354.72	450277.80	450092.57	**450475.57**
#Best	13	12	14	**26**	6	11	0	**23**
p-value	0.00001	0.001535	0.00097		0.000001	0.001698	0	

Table 5.5: Comparison of the TPSDP algorithm with three best performing algorithms on the 40 DGS RanReal instances.

	f_{best}				f_{avg}			
Instance	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
RanReal_n120_ds_01	**50549.14**	50526.74	50529.38	50549.13	50392.28	50319.91	50429.44	**50476.93**
RanReal_n120_ds_02	50904.03	50883.12	50898.02	**50926.60**	50705.44	50757.67	**50761.26**	50727.34
RanReal_n120_ds_03	49979.57	49911.88	49996.07	**50053.22**	49815.68	49837.97	49866.44	**49879.67**
RanReal_n120_ds_04	50349.01	**50363.30**	50354.01	50342.80	50120.03	50275.47	50250.84	**50277.74**
RanReal_n120_ds_05	49648.26	49642.54	49518.53	**49693.57**	49139.73	49435.96	49406.94	**49450.16**
RanReal_n120_ds_06	50219.70	50222.67	50242.53	**50258.79**	50122.13	50108.18	50132.18	**50188.09**
RanReal_n120_ds_07	50088.60	50130.76	50286.19	**50317.02**	49654.84	50078.68	50091.55	**50122.98**
RanReal_n120_ds_08	50421.78	**50479.49**	50471.62	50461.84	50321.84	50333.12	50332.10	**50385.19**
RanReal_n120_ds_09	50366.51	50335.45	50374.14	**50432.83**	50279.41	50238.52	50267.87	**50317.12**
RanReal_n120_ds_10	49753.93	49734.02	**49762.73**	49746.00	49576.61	49628.73	**49631.91**	49606.57
RanReal_n240_ds_01	160122.83	160136.43	159862.07	**160219.45**	159579.09	159915.34	159652.74	**159974.22**
RanReal_n240_ds_02	160502.19	160691.02	160355.79	**160831.92**	160072.14	160389.78	160118.10	**160643.87**
RanReal_n240_ds_03	159436.63	159547.67	159376.23	**159604.83**	159121.04	159290.76	159042.91	**159419.88**
RanReal_n240_ds_04	161167.60	161398.28	161370.66	**161649.58**	160607.87	161133.74	160836.86	**161304.29**
RanReal_n240_ds_05	**159474.95**	159197.44	159064.93	159354.10	158805.89	158908.34	158696.44	**159027.11**
RanReal_n240_ds_06	161025.25	161291.73	161008.70	**161429.53**	160536.87	160885.46	160587.54	**161149.94**
RanReal_n240_ds_07	159808.12	160125.97	160027.42	**160259.15**	159322.17	159762.92	159484.60	**159995.48**
RanReal_n240_ds_08	158543.83	158617.15	158431.21	**158631.89**	158193.17	158410.12	158187.21	**158429.30**
RanReal_n240_ds_09	159707.67	159837.30	159723.18	**159928.39**	159229.31	159560.87	159338.42	**159690.98**
RanReal_n240_ds_10	159988.80	160282.32	160253.93	**160365.46**	159643.18	160070.90	159881.13	**160209.78**
RanReal_n480_ds_01	388612.12	388621.76	388561.23	**389658.36**	386910.54	388163.72	387852.72	**388326.45**
RanReal_n480_ds_02	386295.34	387123.91	386642.47	**387382.94**	385139.30	386572.47	385813.67	**386673.60**
RanReal_n480_ds_03	387597.43	**388634.13**	388142.87	388630.06	386775.20	388053.62	387282.50	**388104.73**
RanReal_n480_ds_04	389810.19	390853.65	391121.94	**391526.90**	389097.88	390411.26	390271.95	**390762.60**
RanReal_n480_ds_05	387831.00	388290.29	387917.36	**388449.32**	386849.14	387708.10	387241.91	**387836.09**
RanReal_n480_ds_06	388715.70	389667.02	389247.42	**389711.24**	387903.70	389060.85	388394.41	**389163.80**
RanReal_n480_ds_07	388270.09	389179.47	388498.22	**389372.07**	387361.43	388354.32	387710.23	**388561.99**
RanReal_n480_ds_08	389168.12	**389612.75**	388584.82	389512.51	387796.59	388828.71	387980.52	**388984.07**
RanReal_n480_ds_09	387008.87	**388345.33**	386895.25	387715.54	385874.71	**387389.04**	386391.77	387324.65
RanReal_n480_ds_10	392010.85	392996.39	393143.90	**393957.60**	390519.66	392605.69	392503.87	**393089.61**
RanReal_n960_ds_01	1237439.17	1240609.19	1239896.29	**1240917.68**	1233783.12	1239177.39	1238296.28	**1239891.19**
RanReal_n960_ds_02	1236026.07	1239648.77	1239614.96	**1241146.98**	1233019.74	1238723.63	1237463.86	**1239515.63**
RanReal_n960_ds_03	1235299.80	**1239428.79**	1239260.16	1239069.46	1233104.16	1237473.98	1237013.93	**1237910.82**
RanReal_n960_ds_04	1235299.61	1240769.04	1240151.40	**1241057.09**	1233271.14	1239337.80	1238114.79	**1239796.93**
RanReal_n960_ds_05	1233826.33	1237195.72	1237882.14	**1238142.18**	1232196.49	1235972.43	1235674.75	**1236782.20**
RanReal_n960_ds_06	1231398.58	1234577.62	1234215.06	**1234990.49**	1229138.66	1233220.92	1232638.06	**1233837.28**
RanReal_n960_ds_07	1234834.14	**1239662.27**	1238462.61	1239376.43	1232632.44	1237837.03	1236747.79	**1238412.13**
RanReal_n960_ds_08	1229796.61	1233435.63	1233521.41	**1233702.77**	1228119.37	1232219.71	1232170.24	**1232726.36**
RanReal_n960_ds_09	1235111.29	1238647.53	1238828.95	**1238910.65**	1232422.37	1237007.12	1236978.01	**1237900.50**
RanReal_n960_ds_10	1237652.66	1241302.25	1241190.20	**1242260.21**	1235929.09	1240293.19	1239156.87	**1240658.10**
Avg.	458351.56	459548.92	459342.15	**459763.66**	457327.08	458943.84	458567.37	**459188.38**
#Best	2	7	1	**30**	0	1	2	**37**
p-value	0	0.0001	0		0	0	0	

Table 5.6: Comparison of the TPSDP algorithm with three best performing algorithms on the 40 EGS RanReal instances.

	f_{best}				f_{avg}			
Instance	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
RanReal_n120_ss_01	**47363.21**	47351.97	**47363.21**	**47363.21**	47335.38	47258.23	47266.44	**47343.68**
RanReal_n120_ss_02	**47243.16**	47188.62	**47243.16**	**47243.16**	**47198.62**	47147.90	47166.87	47197.48
RanReal_n120_ss_03	**47313.71**	**47313.71**	**47313.71**	**47313.71**	47267.06	47237.47	47262.65	**47276.01**
RanReal_n120_ss_04	**47546.81**	47500.08	**47546.81**	**47546.81**	47506.50	47437.26	47483.39	**47506.61**
RanReal_n120_ss_05	**46930.19**	**46930.19**	**46930.19**	**46930.19**	46866.39	46824.36	46839.07	**46868.57**
RanReal_n120_ss_06	**47253.47**	47201.28	47240.09	**47253.47**	47193.81	47144.95	47162.99	**47203.13**
RanReal_n120_ss_07	**47085.87**	**47085.87**	**47085.87**	**47085.87**	47039.10	47003.77	47008.85	**47046.24**
RanReal_n120_ss_08	**47460.13**	**47460.13**	**47460.13**	**47460.13**	47452.58	47437.67	47433.37	**47455.15**
RanReal_n120_ss_09	**47686.34**	**47686.34**	**47686.34**	**47686.34**	**47655.54**	47649.10	47617.92	47655.52
RanReal_n120_ss_10	**47415.35**	**47415.35**	**47415.35**	**47415.35**	**47369.89**	47351.76	47360.09	47366.15
RanReal_n240_ss_01	155135.02	155223.13	154934.03	**155246.47**	154936.92	155001.88	154807.85	**155041.88**
RanReal_n240_ss_02	155611.46	155627.77	155341.65	**155656.23**	155331.24	155356.21	155100.18	**155451.66**
RanReal_n240_ss_03	155546.38	155605.64	155532.55	**155782.29**	155327.89	155401.30	155216.30	**155566.95**
RanReal_n240_ss_04	155275.08	155300.58	155179.31	**155411.09**	155069.22	155084.06	154909.21	**155235.42**
RanReal_n240_ss_05	**154944.29**	154836.75	154640.68	154935.07	154674.58	154708.29	154475.33	**154802.65**
RanReal_n240_ss_06	155581.77	155513.94	155417.40	**155671.23**	155298.34	155217.34	155064.94	**155428.52**
RanReal_n240_ss_07	155715.34	155673.86	155572.55	**155739.51**	155458.56	155472.66	155307.99	**155515.81**
RanReal_n240_ss_08	**155675.74**	155506.95	155382.13	155604.41	155428.80	155400.67	155255.01	**155501.73**
RanReal_n240_ss_09	154884.93	155147.46	155018.41	**155174.95**	154735.78	154800.48	154587.76	**154931.66**
RanReal_n240_ss_10	155880.48	155867.28	155867.93	**155927.91**	155674.01	155686.16	155552.37	**155776.44**
RanReal_n480_ss_01	377630.06	**378212.60**	377501.66	377946.02	376880.41	377497.64	376991.96	**377533.30**
RanReal_n480_ss_02	377249.45	**378105.01**	377278.62	377578.02	376521.20	**377266.26**	376682.45	377082.09
RanReal_n480_ss_03	378603.39	**379144.00**	378455.96	378758.45	377970.16	**378419.75**	377755.01	378240.08
RanReal_n480_ss_04	377512.67	**377823.71**	377323.92	377823.18	376808.86	**377388.00**	376850.87	377375.04
RanReal_n480_ss_05	378147.28	**378711.51**	378168.88	378476.06	377218.26	**377997.49**	377618.99	377984.11
RanReal_n480_ss_06	378410.82	378958.35	378324.10	**379221.05**	377786.56	**378594.68**	377864.65	378558.50
RanReal_n480_ss_07	378527.49	**379225.64**	378387.56	378909.90	377912.85	**378640.33**	377889.51	378430.61
RanReal_n480_ss_08	377789.91	378286.67	377791.70	**378423.50**	376991.03	**377627.27**	376965.18	377515.12
RanReal_n480_ss_09	377939.10	377934.05	377826.58	**378107.96**	376793.53	**377472.65**	376811.27	377334.26
RanReal_n480_ss_10	379490.57	379475.22	379011.63	**379503.98**	378560.47	**379074.18**	378310.32	378950.56
RanReal_n960_ss_01	1213879.02	1216654.49	**1217951.51**	1217333.22	1212256.16	1215850.63	1216200.99	**1216503.80**
RanReal_n960_ss_02	1215915.21	1218254.67	1218147.35	**1218548.34**	1213827.33	1217106.92	1216961.03	**1217820.76**
RanReal_n960_ss_03	1214914.09	1217816.69	**1219490.53**	1217795.42	1213330.54	1216576.31	1216752.11	**1217087.01**
RanReal_n960_ss_04	1215327.26	1218426.41	**1219341.10**	1219093.41	1214058.92	1217511.37	1217393.22	**1217863.43**
RanReal_n960_ss_05	1213287.33	1216224.83	**1217148.08**	1216590.16	1211933.09	1214941.99	1215312.48	**1215623.14**
RanReal_n960_ss_06	1214036.77	1217191.24	1216529.75	**1217570.75**	1212304.52	1215575.71	1215466.99	**1216221.44**
RanReal_n960_ss_07	1214498.99	1217842.82	1218331.43	**1218365.27**	1212505.91	1216395.22	1216336.89	**1217176.97**
RanReal_n960_ss_08	1213346.03	1216258.57	**1216900.19**	1216296.06	1211617.39	1215015.22	1214721.77	**1215596.32**
RanReal_n960_ss_09	1214559.97	1217789.79	**1218584.31**	1218523.66	1213002.89	1216868.88	1217046.05	**1217541.00**
RanReal_n960_ss_10	1217757.59	1218829.25	**1220090.49**	1219794.66	1214065.84	1217452.85	1217159.40	**1218525.96**
Avg.	448909.29	449715.06	449718.92	**449827.66**	448179.15	449197.37	448999.24	**449378.37**
#Best	12	12	16	**25**	3	9	0	**28**
p-value	0.000002	0.002279	0.011773		0	0.0008790	0	

Table 5.7: Comparison of the TPSDP algorithm with three best performing algorithms on the 40 DGS Geo instances.

	f_{best}				f_{avg}			
Instance	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
Geo_n120_ds_01	**111922.82**	111906.95	111906.46	111867.79	**111907.34**	111754.41	111721.35	111831.56
Geo_n120_ds_02	61916.95	61906.32	**61917.89**	61893.42	61903.36	61754.30	**61910.84**	61883.69
Geo_n120_ds_03	52083.72	52075.64	**52085.12**	52069.08	52073.34	52001.65	**52076.24**	52063.21
Geo_n120_ds_04	**80801.10**	80768.89	80795.31	80734.53	**80779.74**	80755.46	80771.55	80723.34
Geo_n120_ds_05	**121775.53**	121706.99	121738.78	121675.28	**121730.93**	121680.82	121693.31	121629.18
Geo_n120_ds_06	**136875.63**	136843.28	136858.61	136809.72	**136807.14**	136712.26	136733.18	136765.59
Geo_n120_ds_07	**108576.13**	108514.53	108534.53	108476.98	108402.98	108439.54	108446.54	**108450.23**
Geo_n120_ds_08	**88230.84**	88186.63	88206.52	88160.87	**88205.86**	88172.55	88183.07	88138.78
Geo_n120_ds_09	**95492.54**	95467.64	95470.01	95430.23	**95472.45**	95389.38	95344.92	95399.47
Geo_n120_ds_10	**65559.57**	65553.37	65556.61	65530.34	65529.15	65528.71	**65532.25**	65509.39
Geo_n240_ds_01	200357.13	200330.47	**200383.83**	200324.35	200327.80	200271.23	**200354.51**	200306.53
Geo_n240_ds_02	348500.04	348483.89	**348532.13**	348401.46	347531.92	348182.61	348177.94	**348349.00**
Geo_n240_ds_03	217176.26	217149.37	**217223.32**	217126.41	217135.35	216995.12	**217172.86**	217106.84
Geo_n240_ds_04	263843.03	263806.43	**263881.19**	263777.50	263493.80	263296.25	263323.30	**263746.12**
Geo_n240_ds_05	313398.28	313385.67	**313413.22**	313322.49	313379.10	313154.89	**313383.25**	313243.57
Geo_n240_ds_06	358633.36	358554.61	**358649.94**	358469.01	**358601.10**	358504.96	358520.56	358427.69
Geo_n240_ds_07	**341992.17**	341935.62	341981.48	341844.65	340487.59	341305.63	341392.24	**341812.72**
Geo_n240_ds_08	131024.25	131027.66	131022.83	**131030.29**	131021.03	131021.13	131018.95	**131023.75**
Geo_n240_ds_09	**410563.19**	410495.74	410548.77	410421.52	409947.10	410174.19	**410405.06**	410352.54
Geo_n240_ds_10	**355254.36**	355215.02	355245.86	355088.91	**355146.15**	354885.41	354694.72	355041.24
Geo_n480_ds_01	580908.19	582322.29	**582573.07**	582325.61	580858.03	**581672.00**	581146.75	580550.25
Geo_n480_ds_02	1089035.81	1089817.97	**1090132.22**	1089600.18	1088980.33	1088879.45	**1089420.63**	1087807.40
Geo_n480_ds_03	662588.72	664189.65	**664476.64**	664110.67	661942.55	**663269.38**	663122.88	661980.97
Geo_n480_ds_04	**836599.77**	836334.15	836597.91	836150.39	**836539.29**	835905.40	835966.91	835650.75
Geo_n480_ds_05	**988501.10**	988179.39	988428.30	988022.03	987252.28	**987590.96**	986385.43	987061.64
Geo_n480_ds_06	1012582.81	1012248.52	**1012585.52**	1012102.97	1011818.62	**1011846.99**	1011153.86	1011316.47
Geo_n480_ds_07	**864994.16**	864673.18	864944.86	864476.17	**864831.32**	863835.05	863871.48	862767.22
Geo_n480_ds_08	587697.46	587736.17	**587745.62**	587666.11	587330.89	**587355.35**	587060.44	587306.03
Geo_n480_ds_09	666313.37	667228.01	**667507.06**	667176.41	666274.27	666563.53	**666916.58**	666554.82
Geo_n480_ds_10	932694.79	937346.75	**937522.52**	937198.86	929618.15	**936173.88**	935100.17	936038.48
Geo_n960_ds_01	3361972.63	**3364644.62**	3364423.08	3364410.87	3351095.76	**3364126.49**	3358725.84	3363298.19
Geo_n960_ds_02	1719726.15	1723404.86	1722222.03	**1723421.56**	1716949.23	**1722538.96**	1720720.97	1722147.18
Geo_n960_ds_03	3347824.49	3350874.67	**3351678.34**	3350670.94	3347718.39	3349888.74	3348209.61	**3350369.56**
Geo_n960_ds_04	3615142.37	3623049.49	**3623660.05**	3622771.96	3606303.51	3622385.41	3620732.16	**3622529.82**
Geo_n960_ds_05	2342436.81	2341868.22	**2342538.16**	2341851.12	**2342272.41**	2341309.18	2339890.95	2341251.76
Geo_n960_ds_06	3153310.79	3152726.84	**3153473.31**	3152437.49	3151011.34	**3152510.12**	3150985.78	3152341.11
Geo_n960_ds_07	1301823.28	1301809.20	**1301860.63**	1301802.75	1298573.57	**1300342.49**	1299168.77	1300131.80
Geo_n960_ds_08	1721957.65	1723484.60	**1723753.93**	1723466.90	1720331.72	1723172.95	**1723275.13**	1723187.11
Geo_n960_ds_09	1894819.06	1897519.30	**1897882.96**	1897475.94	1891535.57	**1896604.30**	1895275.99	1896043.56
Geo_n960_ds_10	2617068.31	2617718.10	**2618355.83**	2617616.69	2616307.08	**2617616.61**	2616579.45	2617542.40
Avg.	929049.36	929762.27	**929907.86**	929680.26	927935.69	**929339.19**	928864.16	929192.02
#Best	14	1	**23**	2	10	**12**	10	7
p-value	0.994638	1	1		0.285258	1	0.10244	

Table 5.8: Comparison of the TPSDP algorithm with three best performing algorithms on the 40 EGS Geo instances.

	f_{best}				f_{avg}			
Instance	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
Geo_n120_ss_01	**101624.92**	101600.29	101609.44	101568.85	**101610.31**	101584.09	101595.18	101553.75
Geo_n120_ss_02	54853.70	54842.74	**54859.63**	54829.46	54847.90	54832.85	**54849.38**	54823.16
Geo_n120_ss_03	47631.75	47622.06	**47632.08**	47616.07	47625.37	47615.85	**47627.03**	47609.63
Geo_n120_ss_04	**73526.44**	73490.60	73498.60	73473.81	**73501.84**	73477.07	73488.04	73452.80
Geo_n120_ss_05	**112657.57**	112616.05	112645.28	112578.33	**112631.45**	112595.13	112614.81	112567.32
Geo_n120_ss_06	**125451.95**	125402.63	125414.05	125379.76	**125404.08**	125374.61	125384.67	125346.91
Geo_n120_ss_07	**98494.00**	98474.71	98484.78	98469.59	**98482.96**	98457.89	98465.81	98433.13
Geo_n120_ss_08	79982.16	79958.56	**79983.91**	79933.50	**79966.43**	79941.96	79955.50	79911.62
Geo_n120_ss_09	**87281.56**	87249.92	87252.91	87215.47	**87259.19**	87233.69	87243.27	87203.11
Geo_n120_ss_10	**60258.25**	60236.79	60255.49	60220.80	**60248.59**	60227.62	60240.95	60212.13
Geo_n240_ss_01	**188872.18**	188838.69	188864.98	188824.79	**188858.62**	188825.47	188854.58	188807.34
Geo_n240_ss_02	**330318.81**	330272.45	330290.18	330195.79	**330296.37**	330248.40	330281.05	330174.32
Geo_n240_ss_03	207066.88	207030.05	**207080.64**	206997.54	207054.99	207019.60	**207065.84**	206987.04
Geo_n240_ss_04	**246389.29**	246330.95	246385.04	246294.05	**246367.64**	246321.35	246364.78	246277.25
Geo_n240_ss_05	**298773.97**	298704.73	298748.93	298652.52	**298741.92**	298685.77	298725.42	298613.53
Geo_n240_ss_06	338595.80	338552.44	**338600.14**	338450.50	**338565.37**	338511.72	338551.34	338432.34
Geo_n240_ss_07	326057.89	326010.51	**326058.80**	325956.71	**326034.03**	325984.60	326026.71	325908.09
Geo_n240_ss_08	**126913.16**	126901.38	126901.23	126900.72	**126909.40**	126897.92	126898.15	126897.78
Geo_n240_ss_09	**391447.26**	391395.02	391413.64	391320.95	**391417.04**	391353.81	391388.26	391275.90
Geo_n240_ss_10	**339544.99**	339474.11	339541.13	339423.70	**339507.36**	339448.43	339496.73	339380.71
Geo_n480_ss_01	552177.33	552010.90	**552203.09**	551994.36	552160.76	551992.55	**552173.72**	551968.13
Geo_n480_ss_02	**1047453.58**	1047211.56	1047397.86	1047080.67	**1047390.11**	1047134.30	1047331.55	1047008.22
Geo_n480_ss_03	633769.72	633572.36	**633774.93**	633516.84	**633738.14**	633539.96	633728.79	633483.30
Geo_n480_ss_04	**789829.74**	789616.25	789814.39	789498.20	**789783.87**	789556.59	789753.20	789451.65
Geo_n480_ss_05	**945944.75**	945666.03	945880.84	945567.35	**945895.37**	945624.87	945843.90	945511.86
Geo_n480_ss_06	**966654.45**	966378.06	966603.44	966282.28	**966585.81**	966332.85	966539.33	966218.09
Geo_n480_ss_07	**827713.64**	827467.13	827659.08	827366.82	**827658.68**	827411.54	827625.55	827312.04
Geo_n480_ss_08	556651.12	556480.79	**556686.75**	556458.51	556634.19	556464.07	**556645.73**	556438.67
Geo_n480_ss_09	636346.80	636195.91	**636368.44**	636100.10	**636324.19**	636137.04	636322.68	636074.96
Geo_n480_ss_10	**883434.68**	883151.37	883361.54	883062.80	**883368.14**	883103.62	883314.48	882995.42
Geo_n960_ss_01	3254335.92	3253781.45	**3254371.56**	3253760.52	**3254289.42**	3253693.88	3254287.80	3253760.52
Geo_n960_ss_02	**1663664.99**	1663376.22	1663623.98	1663372.45	**1663640.24**	1663345.03	1663587.64	1663372.45
Geo_n960_ss_03	3251594.97	3251016.56	**3251614.75**	3250909.92	3251527.33	3250953.20	**3251530.71**	3250909.92
Geo_n960_ss_04	**3514333.85**	3513729.27	3514273.56	3513595.55	**3514236.36**	3513621.90	3514197.42	3513595.55
Geo_n960_ss_05	2264719.61	2264330.22	**2264822.68**	2264312.03	2264692.77	2264282.64	**2264757.80**	2264312.03
Geo_n960_ss_06	**3069667.72**	3069147.40	3069651.92	3068999.68	3069588.50	3069052.35	**3069597.79**	3068999.68
Geo_n960_ss_07	**1257750.45**	1257593.57	1257652.01	1257603.02	**1257737.72**	1257586.90	1257622.12	1257603.02
Geo_n960_ss_08	**1674016.12**	1673717.34	1673966.10	1673731.89	**1673988.06**	1673692.51	1673927.01	1673731.89
Geo_n960_ss_09	1835490.33	1835159.16	**1835509.95**	1835179.11	1835443.33	1835119.73	**1835454.14**	1835179.11
Geo_n960_ss_10	**2529011.12**	2528619.11	2529001.43	2528469.54	2528929.06	2528494.42	**2528957.08**	2528469.54
Avg.	**894757.58**	894580.63	894743.98	894529.11	**894723.57**	894544.44	894707.90	894506.60
#Best	**26**	0	14	0	**30**	0	10	0
p-value	1	1	1		1	1	1	

Table 5.9: Comparison of the TPSDP algorithm with the NSGGA algorithm on the 40 EGS RanInt instances.

	f_{best}		f_{avg}	
Instance	**NSGGA**	**TPSDP**	**NSGGA**	**TPSDP**
RanInt_n120_ss_01	**47909**	**47909**	47879.50	**47909.00**
RanInt_n120_ss_02	**47826**	**47826**	47770.85	**47822.80**
RanInt_n120_ss_03	**47552**	**47552**	47423.65	**47489.10**
RanInt_n120_ss_04	**47611**	47547	47489.10	**47518.05**
RanInt_n120_ss_05	47148	**47210**	47106.45	**47137.30**
RanInt_n120_ss_06	**46647**	**46647**	46598.35	**46617.75**
RanInt_n120_ss_07	47108	**47142**	47069.45	**47122.30**
RanInt_n120_ss_08	**47390**	**47390**	47353.05	**47357.55**
RanInt_n120_ss_09	47654	**47660**	47612.95	**47635.10**
RanInt_n120_ss_10	**47807**	**47807**	47784.65	**47798.15**
RanInt_n240_ss_01	155463	**155577**	155303.41	**155440.20**
RanInt_n240_ss_02	155308	**155384**	155108.34	**155207.25**
RanInt_n240_ss_03	**156415**	**156415**	156168.34	**156319.40**
RanInt_n240_ss_04	156616	**156643**	156458.41	**156513.55**
RanInt_n240_ss_05	156431	**156562**	156135.05	**156295.70**
RanInt_n240_ss_06	155576	**155601**	155306.66	**155402.40**
RanInt_n240_ss_07	155789	**155791**	155524.41	**155678.40**
RanInt_n240_ss_08	155213	**155297**	154995.75	**155167.95**
RanInt_n240_ss_09	**156043**	**156043**	155780.09	**155923.40**
RanInt_n240_ss_10	155909	**155971**	155724.45	**155854.20**
RanInt_n480_ss_01	**380107**	379953	**379642.00**	379263.95
RanInt_n480_ss_02	**380270**	380180	379498.34	**379596.55**
RanInt_n480_ss_03	**379225**	378762	**378785.41**	378291.55
RanInt_n480_ss_04	**379483**	379008	**378967.84**	378628.00
RanInt_n480_ss_05	379828	**379883**	**379362.94**	379132.55
RanInt_n480_ss_06	**379444**	379354	**379044.84**	378835.95
RanInt_n480_ss_07	**380362**	379741	**379475.69**	379151.40
RanInt_n480_ss_08	**380200**	380161	**379496.06**	379341.55
RanInt_n480_ss_09	**379568**	379060	**378785.50**	378410.25
RanInt_n480_ss_10	**380924**	380248	**379986.09**	379813.25
RanInt_n960_ss_01	1220366	**1220742**	1219190.25	**1219694.55**
RanInt_n960_ss_02	**1220479**	1220325	1219148.38	**1219184.35**
RanInt_n960_ss_03	1220878	**1221283**	1219528.00	**1220117.65**
RanInt_n960_ss_04	1220275	**1220857**	1219292.25	**1219949.50**
RanInt_n960_ss_05	1219787	**1220702**	1218986.38	**1219607.40**
RanInt_n960_ss_06	**1221495**	1221066	1220032.25	**1220231.30**
RanInt_n960_ss_07	1221294	**1221650**	1219842.75	**1220481.15**
RanInt_n960_ss_08	**1221688**	1221474	1219910.25	**1220252.95**
RanInt_n960_ss_09	1218962	**1219501**	1217971.25	**1218535.95**
RanInt_n960_ss_10	1218750	**1219430**	1217920.00	**1218293.50**
Avg.	450920.00	**450933.85**	450386.48	**450475.57**
#Best	21	**27**	9	**31**
p-value	0.641163		0.035417	

Table 5.10: Comparison of the TPSDP algorithm with the NSGGA algorithm on the 40 EGS RanReal instances.

	f_{best}		f_{avg}	
Instance	**NSGGA**	**TPSDP**	**NSGGA**	**TPSDP**
RanReal_n120_ss_01	47358.79	**47363.21**	47299.89	**47343.68**
RanReal_n120_ss_02	**47243.16**	**47243.16**	47168.50	**47197.48**
RanReal_n120_ss_03	47280.70	**47313.71**	47210.03	**47276.01**
RanReal_n120_ss_04	**47546.82**	47546.81	47490.04	**47506.61**
RanReal_n120_ss_05	46922.95	**46930.19**	46843.10	**46868.57**
RanReal_n120_ss_06	47227.14	**47253.47**	47154.84	**47203.13**
RanReal_n120_ss_07	47060.41	**47085.87**	47008.38	**47046.24**
RanReal_n120_ss_08	**47460.14**	47460.13	47444.02	**47455.15**
RanReal_n120_ss_09	47678.04	**47686.34**	47590.68	**47655.52**
RanReal_n120_ss_10	**47415.35**	**47415.35**	47318.34	**47366.15**
RanReal_n240_ss_01	155241.95	**155246.47**	154898.16	**155041.88**
RanReal_n240_ss_02	**155732.81**	155656.23	155312.94	**155451.66**
RanReal_n240_ss_03	155680.11	**155782.29**	155423.31	**155566.95**
RanReal_n240_ss_04	155398.34	**155411.09**	155107.42	**155235.42**
RanReal_n240_ss_05	**155937.25**	154935.07	154647.33	**154802.65**
RanReal_n240_ss_06	155671.22	**155671.23**	155345.72	**155428.52**
RanReal_n240_ss_07	155550.39	**155739.51**	155362.50	**155515.81**
RanReal_n240_ss_08	155539.95	**155604.41**	155367.44	**155501.73**
RanReal_n240_ss_09	155084.09	**155174.95**	154750.25	**154931.66**
RanReal_n240_ss_10	**155927.91**	155927.91	155668.47	**155776.44**
RanReal_n480_ss_01	**378470.50**	377946.02	**377748.91**	377533.30
RanReal_n480_ss_02	**377922.50**	377578.02	**377403.06**	377082.09
RanReal_n480_ss_03	**379060.28**	378758.45	**378628.19**	378240.08
RanReal_n480_ss_04	**378238.47**	377823.18	**377640.53**	377375.04
RanReal_n480_ss_05	378371.41	**378476.06**	377880.66	**377984.11**
RanReal_n480_ss_06	379059.34	**379221.05**	**378617.50**	378558.50
RanReal_n480_ss_07	**379282.50**	378909.90	**378883.22**	378430.61
RanReal_n480_ss_08	**378562.09**	378423.50	**378001.09**	377515.12
RanReal_n480_ss_09	377883.31	**378107.96**	**377505.94**	377334.26
RanReal_n480_ss_10	**379643.12**	379503.98	**379221.75**	378950.56
RanReal_n960_ss_01	1217009.88	**1217333.22**	1216043.62	**1216503.80**
RanReal_n960_ss_02	1218401.00	**1218548.34**	1217436.00	**1217820.76**
RanReal_n960_ss_03	**1218220.50**	1217795.42	1216922.88	**1217087.01**
RanReal_n960_ss_04	1218255.25	**1219093.41**	1217311.25	**1217863.43**
RanReal_n960_ss_05	**1216714.88**	1216590.16	1215354.12	**1215623.14**
RanReal_n960_ss_06	1217160.38	**1217570.75**	1215691.00	**1216221.44**
RanReal_n960_ss_07	**1218371.00**	1218365.27	1216881.25	**1217176.97**
RanReal_n960_ss_08	**1216580.50**	1216296.06	1215594.25	**1215596.32**
RanReal_n960_ss_09	**1218706.75**	1218523.66	1217024.50	**1217541.00**
RanReal_n960_ss_10	1219759.88	**1219794.66**	1217950.50	**1218525.96**
Avg.	**449865.78**	449827.66	449303.79	**449378.37**
#Best	19	**23**	9	**31**
p-value	1		0.03098	

Table 5.11: Comparison of the TPSDP algorithm with the NSGGA algorithm on the 40 EGS Geo instances.

	f_{best}		f_{avg}	
Instance	**NSGGA**	**TPSDP**	**NSGGA**	**TPSDP**
Geo_n120_ss_01	**101590.05**	101568.85	**101556.17**	101553.75
Geo_n120_ss_02	54829.43	**54829.46**	54821.13	**54823.16**
Geo_n120_ss_03	47614.74	**47616.07**	**47611.14**	47609.63
Geo_n120_ss_04	73466.77	**73473.81**	73451.27	**73452.80**
Geo_n120_ss_05	**112600.88**	112578.33	**112568.95**	112567.32
Geo_n120_ss_06	125364.30	**125379.76**	125337.59	**125346.91**
Geo_n120_ss_07	98433.67	**98469.59**	98424.72	**98433.13**
Geo_n120_ss_08	79932.39	**79933.50**	79909.77	**79911.62**
Geo_n120_ss_09	**87223.89**	87215.47	**87203.39**	87203.11
Geo_n120_ss_10	**60220.83**	60220.80	60212.07	**60212.13**
Geo_n240_ss_01	188806.19	**188824.79**	188799.56	**188807.34**
Geo_n240_ss_02	330192.50	**330195.79**	330144.69	**330174.32**
Geo_n240_ss_03	**207001.50**	206997.54	206980.47	**206987.04**
Geo_n240_ss_04	246283.41	**246294.05**	246260.75	**246277.25**
Geo_n240_ss_05	298627.53	**298652.52**	298606.06	**298613.53**
Geo_n240_ss_06	338426.69	**338450.50**	338405.97	**338432.34**
Geo_n240_ss_07	325912.34	**325956.71**	325892.00	**325908.09**
Geo_n240_ss_08	126897.87	**126900.72**	126895.20	**126897.78**
Geo_n240_ss_09	391266.97	**391320.95**	391245.62	**391275.90**
Geo_n240_ss_10	339372.19	**339423.70**	339349.91	**339380.71**
Geo_n480_ss_01	551989.25	**551994.36**	551960.31	**551968.13**
Geo_n480_ss_02	1047009.75	**1047080.67**	1046962.69	**1047008.22**
Geo_n480_ss_03	**633530.25**	633516.84	633468.94	**633483.30**
Geo_n480_ss_04	789459.94	**789498.20**	789416.94	**789451.65**
Geo_n480_ss_05	945523.81	**945567.35**	945466.38	**945511.86**
Geo_n480_ss_06	966260.12	**966282.28**	966169.12	**966218.09**
Geo_n480_ss_07	827324.69	**827366.82**	827278.19	**827312.04**
Geo_n480_ss_08	556432.38	**556458.51**	556417.94	**556438.67**
Geo_n480_ss_09	636093.31	**636100.10**	636062.12	**636074.96**
Geo_n480_ss_10	882980.31	**883062.80**	882953.12	**882995.42**
Geo_n960_ss_01	3253493.50	**3253760.52**	3253431.50	**3253760.52**
Geo_n960_ss_02	1663315.88	**1663372.45**	1663305.75	**1663372.45**
Geo_n960_ss_03	3250801.75	**3250909.92**	3250692.75	**3250909.92**
Geo_n960_ss_04	3513405.75	**3513595.55**	3513311.50	**3513595.55**
Geo_n960_ss_05	2264305.00	**2264312.03**	2264205.50	**2264312.03**
Geo_n960_ss_06	3068891.00	**3068999.68**	3068806.50	**3068999.68**
Geo_n960_ss_07	1257589.25	**1257603.02**	1257576.62	**1257603.02**
Geo_n960_ss_08	1673716.88	**1673731.89**	1673668.00	**1673731.89**
Geo_n960_ss_09	1835104.50	**1835179.11**	1835071.62	**1835179.11**
Geo_n960_ss_10	2528411.00	**2528469.54**	2528366.75	**2528469.54**
Avg.	894492.56	**894529.11**	894456.72	**894506.60**
#Best	6	**34**	4	**36**
p-value	0.000006		0	

Table 5.12: The result via the Friedman test of ITS, IMS, NDHA, and the proposed TPSDP on total 500 benchmark instances.

Algorithm	ITS	IMS	NDHA	TPSDP
Average ranking	3.394	2.735	1.951	**1.92**
p-value	0	0	0.70419	

tal results of total 500 instances are summarized in Table 5.12. In the Friedman statistical test, an algorithm with better performance gets a lower rank. From Table 5.12, it can be found that TPSDP is competitive with NDHA and performs significantly better than ITS and IMS according to *p*-values. Furthermore, it can be seen that TPSDP ranks 1^{st} among these four algorithms according to the average ranking, which indicates that the proposed TPSDP is a promising method for solving MDGP.

Chapter 6

Parameter analysis and discussion

In this chapter, I analyze the parameter values of some critical components of the TPSDP algorithm, showing the effect of parameter values on the performance of the algorithm. Note that each experiment tests only one undetermined parameter simultaneously, keeping other parameter values as default values during this period. In addition, all the following experimental data are obtained over 20 independent runs on the selected instances. At the end of this chapter, some discussions are given to further analyze several details about TPSDP and find more valid proofs to show its effectiveness.

6.1 Parameter analysis

6.1.1 Influence of the initial population size

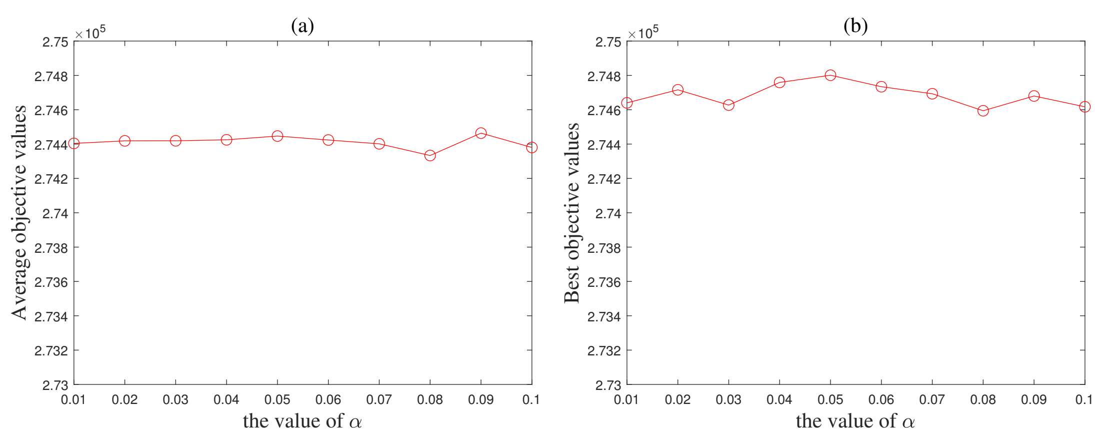


Figure 6.1: Influence of the parameter α.

The algorithm proposed in this paper is based on dynamic population size to solve MDGP. Therefore, we need to determine the initial (maximum) population size β_{max} as well as the final (minimum) population size β_{min}. The experiment of confirming the initial population size β_{max} is based on a subset of MDG-a benchmark instances with $n = 2000$, $m = 50$, $L_g = 32$, $U_g = 48$, which has also been used in [32] for parameter discussion. I adjusted the population size within a reasonable range to determine β_{max} and to analyze the impact of different β_{max} on the performance of the algorithm. Fig. **??**(a) and (b) show the average objective value (Y-axis of (a)) and the best objective value (Y-axis of (b)) for different initial population size β_{max} (X-axis), respectively. Fig. **??**(a) reveals that the experimental results obtained by the algorithm do not differ much under different β_{max}, and the algorithm performs relatively stable. As for the best objective value in Fig. **??**(b), the algorithm finds the best objective value with the highest quality when $\beta_{max} = 15$. Therefore, I set the value of β_{max} as 15 under careful consideration.

6.1.2 Influence of the parameter α in the replacement strategy

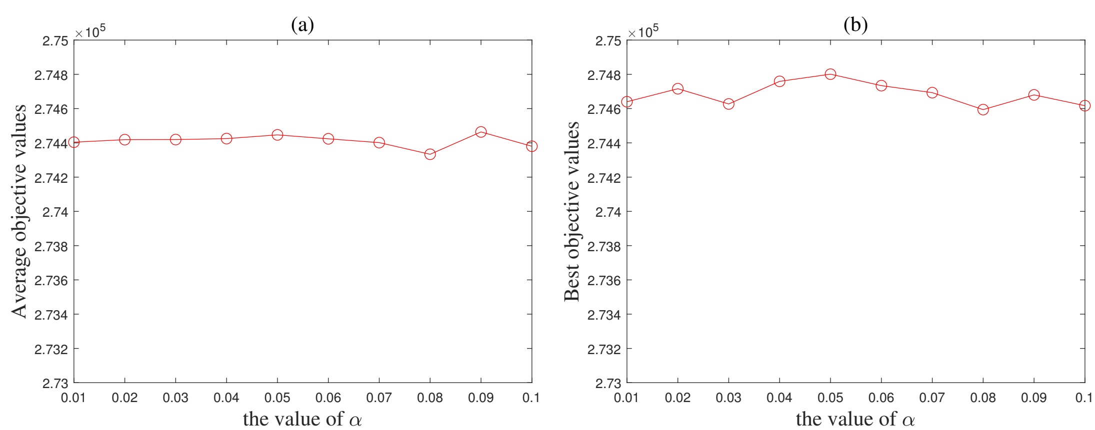


Figure 6.2: Influence of the parameter α.

In the second phase of TPSDP, a replacement strategy guides whether a new generation offspring solution can replace the corresponding parent solution. Therefore, I need to select a proper value for the parameter α in Eq. (4.6). I test TPSDP on the same benchmark instance above with parameter $\alpha \in [0.01, 0.1]$, the same interval as in [48]. Fig. 6.2 plots the variation curves of the average objective values (Y-axis of (a)) and the best objective values

(Y-axis of (b)) with different α values (X-axis). Fig. 6.2(a) indicates that TPSDP performs relatively stable in interval [0.01, 0.06], meanwhile TPSDP finds the best objective value with highest quality in $\alpha = 0.05$ in Fig. 6.2(b). To conclude, the parameter α is set to 0.05.

6.1.3 Influence of the dynamic population size

To determine the final (minimum) population size β_{min}, I choose the RanReal set as the benchmark instance for this experiment. Fig. 6.3 displays the line graphs of the experimental results of the TPSDP algorithm on 40 DGS instances and 40 EGS instances in RanReal set under different β_{min}, respectively. The black curves in Fig. 6.3 indicate the performance of the algorithm on each instance of different scales when the value of β_{min} is 1. Note that $\beta_{min} = 15$ means that TPSDP does not have the population linear descent strategy. The overlap of the four curves on small DGS and EGS instances at $n = 120$ (Fig. 6.3(a) and Fig. 6.3(b)) indicates that TPSDP is not sensitive to the change of β_{min} on them. For $n = 240$ instances, the black and green curves are lower than the red and blue curves on the DGS instance (Fig. 6.3(c)), while the black curve is lower than the other three curves on the EGS instance (Fig. 6.3(d)), which suggests that the population size dropping to 2 or 3 is more suitable for these two instances. However, on the ESG instances at $n = 240$ (Fig. 6.3(d)), the red curve is slightly higher than the blue curve. On balance, I believe that $\beta_{min} = 2$ is more appropriate for the small-scale instances with $n = 120$ and 240.

Fig. 6.3(e) and (f) show the performance comparison of TPSDP with different β_{min} for $n = 480$ instances. From these two figures, it can be seen that there is a large difference in the effectiveness of TPSDP for the same β_{min} value on DGS and EGS instances. For example, when $\beta_{min} = 1$, TPSDP achieves the best performance in the comparison on the DGS instances (black curve is above the red and blue curve), while it performs poorly on the EGS instances. In contrast, in Fig. 6.3(g) and (h), unlike the performance on Fig. 6.3(e) and (f), TPSDP becomes better as the value of β_{min} decreases. When $n = 960$, no matter on different group size and equal group size instances, the performance with $\beta_{min} = 1$ is significantly better than that with 2 or 3. In Fig. 6.3 (e), (f), (g), and (h), it should be noticed that, TPSDP without the population decline strategy performs the worst. It reveals

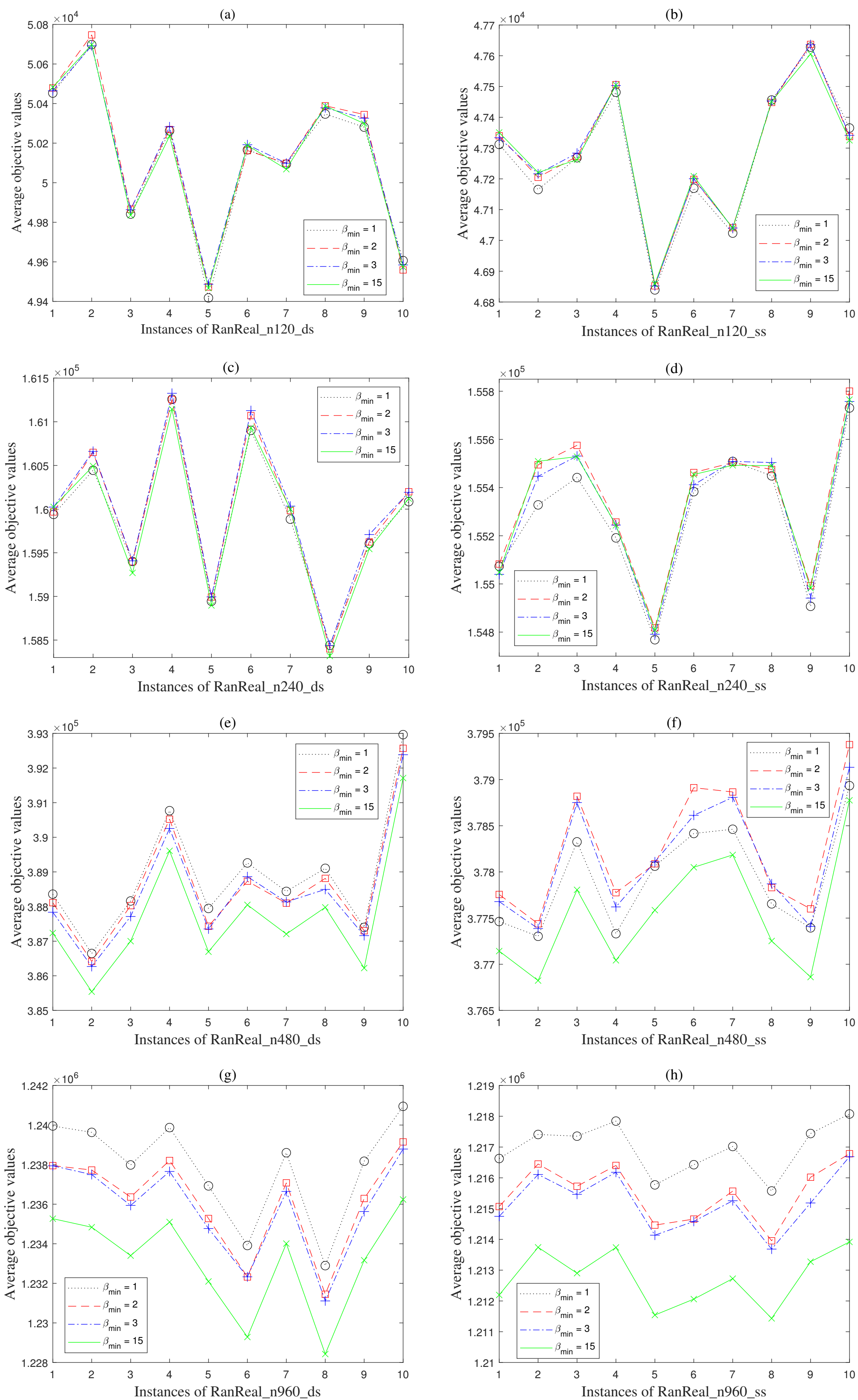


Figure 6.3: Influence of the dynamic population size.

that the proposed population decline strategy is more effective than static population size for large-scale instance. Thus, β_{min} is set to 1 when $n > 400$.

6.1.4 Influence of the undirected perturbation strength

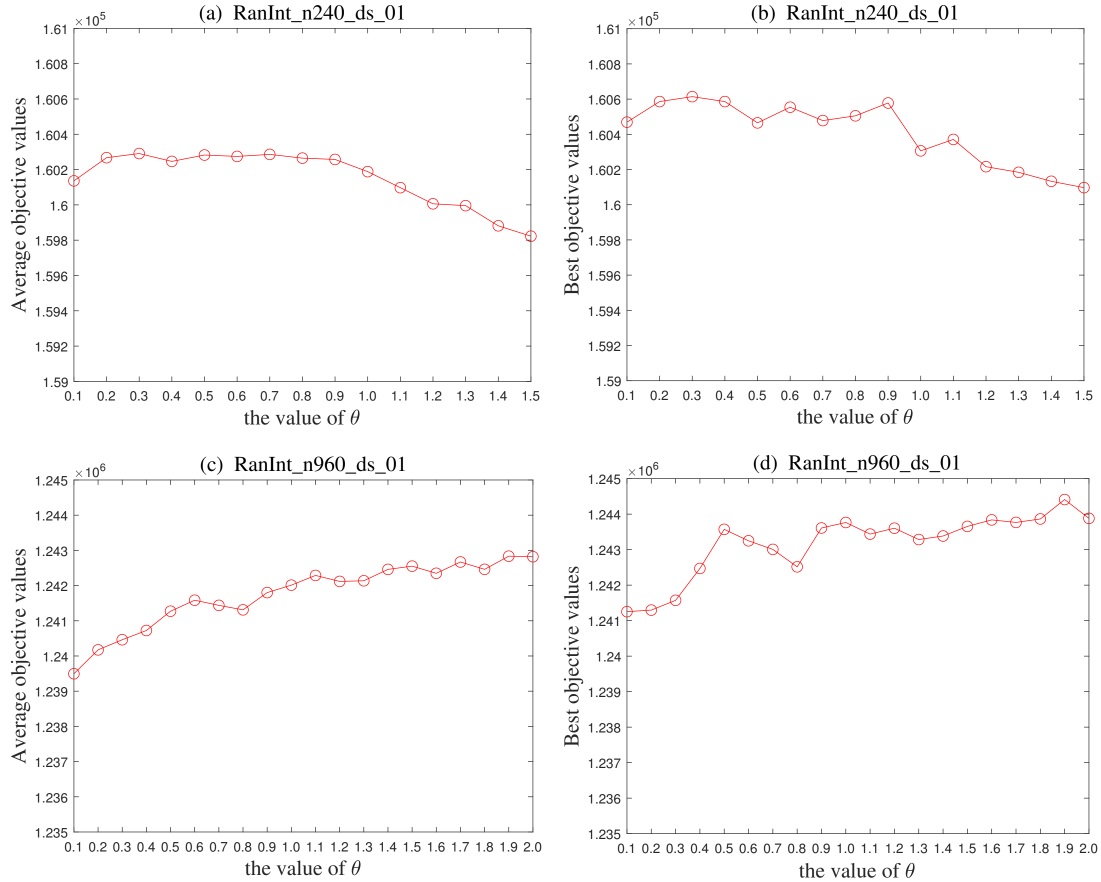


Figure 6.4: Influence of the strength of the undirected perturbation.

Table 6.1: Configuration of θ in four TPSDP variants.

Algorithm	Description	the vaule of θ
TPSDP-Fix	TPSDP with fixed θ	{0.3, 1.9}
TPSDP-Rand	TPSDP with θ taking a random value within an interval in each iteration	[0.1, 1.2] / [1.0, 2.0]
TPSDP-Inc	TPSDP with θ increasing linearly with time in an interval	[0.1, 1.2] / [1.0, 2.0]
TPSDP-Dec	TPSDP with θ decreasing linearly with time in an interval	[1.2, 0.1] / [2.0, 1.0]

In this study, the exploration ability of the proposed TPSDP heavily relies on the undirected perturbation operator. θ as the key parameter controlling the undirected perturbation

Table 6.2: Comparison of TPSDP with different strength of the undirected perturbation on the 36 small-scale instances.

	f_{best}				f_{avg}			
Instance	**TPSDP-Fix**	**TPSDP-Rand**	**TPSDP-Inc**	**TPSDP-Dec**	**TPSDP-Fix**	**TPSDP-Rand**	**TPSDP-Inc**	**TPSDP-Dec**
Geo_n120_ds_01	111863.44	111846.69	111857.52	**111867.79**	**111837.46**	111823.64	111832.50	111831.56
Geo_n120_ds_02	61894.39	**61904.69**	61899.95	61893.42	**61887.09**	61885.93	61887.04	61883.69
Geo_n120_ds_03	52069.83	**52073.59**	52068.96	52069.08	**52065.89**	52063.03	52063.49	52063.21
Geo_n240_ds_01	200317.16	200313.71	200320.24	**200324.35**	200302.92	200297.28	**200303.15**	200271.23
Geo_n240_ds_02	**348402.30**	348369.13	348380.81	348401.46	**348356.50**	348332.46	348345.79	348182.61
Geo_n240_ds_03	217134.00	217114.48	**217134.40**	217126.41	**217111.48**	217098.61	217108.14	216995.12
RanInt_n120_ds_01	**51171.00**	51127.00	51132.00	51146.00	**51099.10**	51043.35	51047.45	51075.85
RanInt_n120_ds_02	**51480.00**	51332.00	51417.00	51372.00	**51351.55**	51241.25	51288.95	51294.65
RanInt_n120_ds_03	**50264.00**	50246.00	50242.00	50248.00	**50215.60**	50161.30	50165.75	50192.05
RanInt_n240_ds_01	**160630.00**	160585.00	160427.00	160596.00	160257.45	160260.75	160204.80	**160358.10**
RanInt_n240_ds_02	160445.00	160286.00	160290.00	**160468.00**	160155.35	160107.10	160065.50	**160277.70**
RanInt_n240_ds_03	160217.00	160175.00	160232.00	**160400.00**	160017.35	160046.75	159968.30	**160223.05**
RanReal_n120_ds_01	**50560.66**	50535.22	50554.32	50549.13	**50487.69**	50437.03	50450.32	50476.93
RanReal_n120_ds_02	**50936.27**	50803.55	50918.09	50926.60	**50766.93**	50673.71	50686.23	50727.34
RanReal_n120_ds_03	50016.40	49996.07	**50053.22**	**50053.22**	**49930.35**	49854.88	49869.97	49879.67
RanReal_n240_ds_01	160183.33	160054.79	160098.40	**160219.45**	159834.67	159870.11	159802.61	**159974.22**
RanReal_n240_ds_02	160813.52	160745.87	160754.10	**160831.92**	160506.06	160510.89	160420.42	**160643.87**
RanReal_n240_ds_03	**159827.70**	159586.93	159453.51	159604.83	159406.48	159309.81	159217.48	**159419.88**
Geo_n120_ss_01	**101584.26**	101582.53	101573.00	101568.85	101559.17	**101560.11**	101557.12	101553.75
Geo_n120_ss_02	**54836.06**	54831.20	54830.60	54829.46	**54825.81**	54821.64	54823.20	54823.16
Geo_n120_ss_03	**47617.78**	47617.22	47615.53	47616.07	47611.12	**47611.56**	47610.80	47609.63
Geo_n240_ss_01	188813.95	188812.08	188817.50	**188824.79**	188806.46	188807.20	**188807.87**	188807.34
Geo_n240_ss_02	**330196.39**	330195.11	330184.95	330195.79	330173.45	330166.38	330164.22	**330174.32**
Geo_n240_ss_03	207002.67	207005.67	**207009.95**	206997.54	**206990.24**	206987.60	206989.56	206987.04
RanInt_n120_ss_01	**47909.00**	**47909.00**	**47909.00**	**47909.00**	47876.65	**47909.00**	47903.90	**47909.00**
RanInt_n120_ss_02	**47826.00**	**47826.00**	**47826.00**	**47826.00**	47803.30	**47825.85**	47817.10	47822.80
RanInt_n120_ss_03	**47552.00**	**47552.00**	**47552.00**	**47552.00**	47486.10	**47514.10**	47505.45	47489.10
RanInt_n240_ss_01	155516.00	155565.00	155550.00	**155577.00**	155166.55	155349.00	155307.80	**155440.20**
RanInt_n240_ss_02	155356.00	155356.00	155378.00	**155384.00**	155021.95	**155210.70**	155149.70	155207.25
RanInt_n240_ss_03	**156415.00**	**156415.00**	**156415.00**	**156415.00**	155925.10	156293.35	156193.55	**156319.40**
RanReal_n120_ss_01	**47363.21**	**47363.21**	**47363.21**	**47363.21**	47325.02	47342.87	47327.26	**47343.68**
RanReal_n120_ss_02	**47243.16**	**47243.16**	**47243.16**	**47243.16**	47187.05	**47204.68**	47196.32	47197.48
RanReal_n120_ss_03	**47313.71**	**47313.71**	**47313.71**	**47313.71**	47248.25	**47286.91**	47274.42	47276.01
RanReal_n240_ss_01	155178.33	155209.64	155203.48	**155246.47**	154819.78	**155055.10**	154916.56	155041.88
RanReal_n240_ss_02	155549.49	155633.66	155641.95	**155656.23**	155316.08	155451.55	155381.43	**155451.66**
RanReal_n240_ss_03	155609.44	155755.96	155617.45	**155782.29**	155319.13	155545.54	155465.45	**155566.95**
Avg.	122419.68	122396.72	122396.61	**122427.73**	122279.20	122304.47	122281.10	**122327.54**
#Best	19	9	10	**20**	**13**	9	2	**13**
p-value	0.706381	0.000155	0.002854		0.107325	0.045247	0.00198	

Table 6.3: Comparison of TPSDP with different strength of the undirected perturbation on the 36 large-scale instances.

	f_{best}				f_{avg}			
Instance	**TPSDP-Fix**	**TPSDP-Rand**	**TPSDP-Inc**	**TPSDP-Dec**	**TPSDP-Fix**	**TPSDP-Rand**	**TPSDP-Inc**	**TPSDP-Dec**
Geo_n480_ds_01	582292.22	582287.80	582288.05	**582325.61**	**580966.93**	580621.81	580813.85	580550.25
Geo_n480_ds_02	1089249.81	**1089621.80**	1089607.60	1089600.18	**1088343.32**	1088334.42	1088343.06	1087807.40
Geo_n480_ds_03	664108.21	664097.19	**664126.97**	664110.67	**662452.96**	662074.21	662436.61	661980.97
Geo_n960_ds_01	3364355.99	**3364433.85**	3364298.36	3364410.87	**3363851.70**	3363298.36	3363548.29	3363298.19
Geo_n960_ds_02	1723405.65	1723412.21	1723421.36	**1723421.56**	**1722649.21**	1722543.19	1722242.16	1722147.18
Geo_n960_ds_03	3350671.59	3350635.84	**3350675.75**	3350670.94	3350547.22	3350346.75	**3350563.74**	3350369.56
RanInt_n480_ds_01	390529.00	**390925.00**	390652.00	390718.00	389886.45	390348.90	390023.05	**390362.25**
RanInt_n480_ds_02	389286.00	**389472.00**	389097.00	389327.00	388324.70	388607.40	388511.05	**388743.45**
RanInt_n480_ds_03	388829.00	**389190.00**	388478.00	389098.00	387740.75	388091.35	388031.35	**388362.50**
RanInt_n960_ds_01	**1244412.00**	1244163.00	1242919.00	1244347.00	1242835.90	1242449.40	1242095.05	**1242857.55**
RanInt_n960_ds_02	1241596.00	1241630.00	1241519.00	**1242006.00**	1240518.25	1240632.55	1240279.75	**1240869.75**
RanInt_n960_ds_03	**1242506.00**	1241526.00	1241411.00	1242461.00	1240584.15	1240271.20	1239876.95	**1240896.00**
RanReal_n480_ds_01	388420.39	388699.88	389035.23	**389658.36**	387893.20	388185.89	388117.48	**388326.45**
RanReal_n480_ds_02	387369.53	387259.20	386800.32	**387382.94**	386209.74	386437.34	386340.20	**386673.60**
RanReal_n480_ds_03	388553.72	388529.60	**388642.13**	388630.06	387679.14	387942.76	387809.79	**388104.73**
RanReal_n960_ds_01	**1241028.37**	1240636.08	1240431.95	1240917.68	1239625.89	1239434.48	1238675.74	**1239891.19**
RanReal_n960_ds_02	1240188.45	1240698.20	1239382.87	**1241146.98**	1239065.73	1239217.37	1238429.71	**1239515.63**
RanReal_n960_ds_03	**1239246.15**	1238929.69	1237666.62	1239069.46	1237905.28	1237748.58	1237164.61	**1237910.82**
Geo_n480_ss_01	**552010.33**	551991.18	551992.51	551994.36	551967.96	551970.21	**551971.27**	551968.13
Geo_n480_ss_02	1047074.05	1047070.47	**1047106.34**	1047080.67	1046997.53	**1047014.55**	1047014.34	1047008.22
Geo_n480_ss_03	633516.54	633504.14	633516.55	**633516.84**	633481.35	633479.49	**633484.04**	633483.30
Geo_n960_ss_01	3253684.64	3253661.38	3253682.11	**3253760.52**	3253561.75	3253580.05	3253559.54	**3253760.52**
Geo_n960_ss_02	1663365.47	1663380.98	**1663381.31**	1663372.45	1663347.62	1663360.16	1663354.88	**1663372.45**
Geo_n960_ss_03	**3250912.00**	3250893.64	3250897.45	3250909.92	3250793.85	3250817.72	3250812.06	**3250909.92**
RanInt_n480_ss_01	379673.00	379503.00	**379999.00**	379953.00	378485.80	378971.75	**379298.60**	379263.95
RanInt_n480_ss_02	379526.00	379889.00	380160.00	**380180.00**	378830.10	379369.05	379443.50	**379596.55**
RanInt_n480_ss_03	378362.00	378793.00	**378795.00**	378762.00	377763.45	378115.25	**378330.70**	378291.55
RanInt_n960_ss_01	1220755.00	**1220781.00**	1219814.00	1220742.00	1219599.80	1219349.95	1218856.75	**1219694.55**
RanInt_n960_ss_02	1220221.00	1219811.00	1219518.00	**1220325.00**	**1219193.10**	1218969.65	1218549.00	1219184.35
RanInt_n960_ss_03	1221107.00	1220420.00	1220654.00	**1221283.00**	1219942.75	1219367.60	1219252.85	**1220117.65**
RanReal_n480_ss_01	377407.10	**378399.41**	377882.61	377946.02	376814.89	377382.42	377352.89	**377533.30**
RanReal_n480_ss_02	377007.52	**377680.41**	377570.07	377578.02	376329.17	376992.31	376899.89	**377082.09**
RanReal_n480_ss_03	378000.20	378514.99	378653.23	**378758.45**	377388.28	378042.57	378042.17	**378240.08**
RanReal_n960_ss_01	1217570.56	**1217671.97**	1216768.40	1217333.22	1216484.81	1216173.71	1215807.84	**1216503.80**
RanReal_n960_ss_02	**1218634.04**	1218579.18	1218594.13	1218548.34	1217491.75	1217570.30	1217008.15	**1217820.76**
RanReal_n960_ss_03	1218177.48	**1218266.83**	1217961.70	1217795.42	**1217306.93**	1216899.92	1216562.84	1217087.01
Avg.	1126195.89	1126248.86	1126038.88	**1126365.04**	1125357.26	1125389.24	1125247.33	**1125544.05**
#Best	7	10	7	**12**	7	1	5	**23**
p-value	0.007052	0.077155	0.000502		0.006415	0.000139	0.000752	

strength needs to be analyzed in detail. Therefore, this section analyzes the impact of θ on TPSDP under different strategies. Four strategies are considered to control the value of θ: fixed, random, increase, and decrease. To determine the values for these strategies, I test the influence of different θ (within a reasonable range) on a small-scale instance (i.e., RanInt_n240_ds_01), and a large-scale instance (i.e., RanInt_n960_ds_01). The computational results are shown in Fig. 6.4(a), (b) and Fig. 6.4(c), (d), respectively.

One observes from Fig. 6.4 that the performance of the proposed TPSDP is significantly influenced by the value of θ. First, small values of θ corresponding to a slight perturbation generally lead to a high performance on small-scale instances (Fig. 6.4(a) and (b)). In contrast, large values of θ corresponding to a violent perturbation are more appropriate for large-scale instances (Fig. 6.4(c) and (d)). Furthermore, from Fig. 6.4, we can also observe that the best performance of TPSDP occurs when θ reaches 0.3 and 1.9 for $n = 240$ and $n = 960$ instances, respectively, according to the obtained average and best objective values. Therefore, the first strategy sets the fixed θ value to 0.3 for the instances with $n < 400$ and 1.9 for other instances. Moreover, for the remaining three strategies, I decide to set the variation range of θ as [0.1, 1.2] for the instances with $n < 400$, and [1.0, 2.0] for other instances.

To find the most suitable strategy of θ, the proposed TPSDP adopts these four different strategies to compare on some test instances. Table 6.1 lists detailed information of each strategy. The test instances select the first three instances of RanInt, RanReal, and Geo benchmarks. Tables 6.2 and 6.3 show the comparison results obtained by each strategy on small-scale and large-scale instances, respectively. As Table 6.2 indicates, on small-scale DGS and EGS instances, TPSDP-fix and TPSDP-Dec obtain significant advantages in comparison with the other two strategies. Although the p-value obtained by the Wilcoxon signed-rank test greater than 0.05 confirms no significant difference between them, TPSDP-Dec is slightly better than TPSDP-fix in terms of #Best and Avg. Hence, in this study, θ linearly decreases in the interval [1.2, 0.1] with time for the instances with $n < 400$.

Table 6.3 shows that, for the large-scale DGS and EGS instances, TPSDP-Dec significantly outperforms the other three strategies. Among them, in terms of the p-value of f_{best}, although TPSDP-rand has achieved similar performance to TPSDP-Dec, the TPSDP-rand has inferior stability. In terms of f_{avg}, the p-values from the Wilcoxon signed-rank tests are reported, which suggests that TPSDP-Dec is significantly better than the other strategies. Therefore, θ of our algorithm linearly decreases in the interval [2.0, 1.0] with time for the instances with $n > 400$.

6.1.5 Influence of the directed perturbation strength

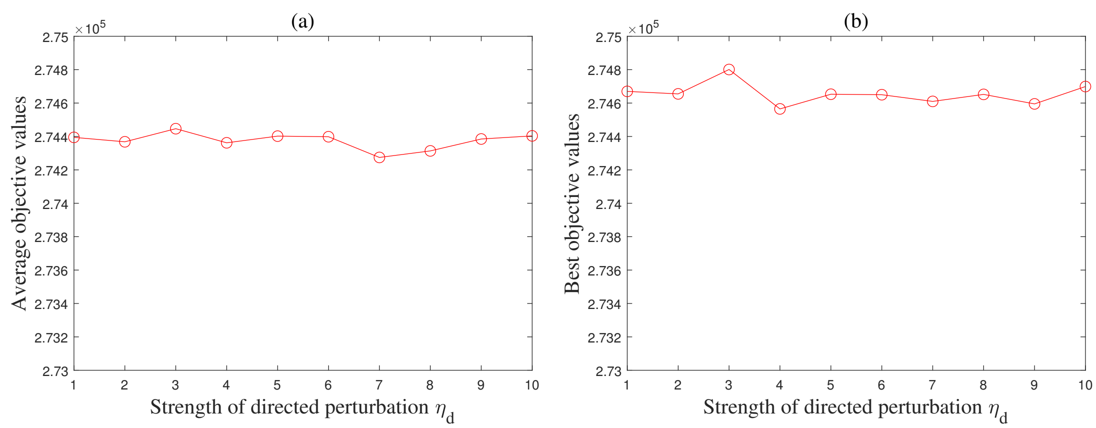


Figure 6.5: Influence of the strength of the directed perturbation.

The last parameter to be determined is the strength η_d of the directed perturbation operator in TPSDP. Like above, the benchmark instances are still the subset of $m = 50$, $L_g = 32$, $U_g = 48$ in MDG-a. The X-axis in Fig. 6.5 represents the value of η_d to be tested, while the Y-axis in Fig. 6.5(a) and (b) represent the average objective value and the best objective value, respectively. From Fig. 6.5(a), it can be observed that the curve has fluctuation, and the algorithm performs the best when the value of η_d is 3. Furthermore, it can be found in Fig. 6.5(b) that there is a large difference between the best objective values in the interval [1, 4], and the best objective value with the highest quality corresponds to a value of η_d = 3. Therefore, η_d is set to 3 in the proposed TPSDP in terms of the overall performance.

6.2 Discussion

6.2.1 Discussion of the method of decreasing the population size

For making more effective use of limited computing resources, TPSDP adopts a population linear decline strategy such that the population decreases with time. In this section, I discuss the way in which solutions to be discarded when the population is reduced. Three different strategies including TPSDP-O, TPSDP-OD, and TPSDP-D are considered. In TPSDP-O, only the objective value of the solutions is considered, i.e., the solution with the smallest objective value is discarded. TPSDP-OD considers both the objective value and the distance between solutions and the optimal solution, i.e., a solution with the lowest value calculated by Eq. (4.6) is discarded. In TPSDP-D, only the distance between solutions is considered. To be specific, the distance from each solution to the optimal solution is calculated by Eq. (4.7), and the one closest to the optimal solution is discarded. These three strategies are used in TPSDP, and then tested on 72 instances to verify which one is the best.

Tables 6.4 and 6.5 disclose the comparison results obtained by TPSDP-O, TPSDP-OD, and TPSDP-D on the given instances. Results on 36 DGS instances reported in Table 6.4 show that TPSDP-O can find the best value on 14 instances in terms of f_{best}, which is better than 12 of TPSDP-OD, and 10 of TPSDP-D. In terms of f_{avg}, TPSDP-O is tied with TPSDP-D, but superior to TPSDP-OD, and they yield the best results on 14, 8, and 14 instances, respectively. Tables 6.5 indicates that for the EGS instances, TPSDP-O can find the best f_{best} and f_{avg} results on 19 and 16 instances, which is better than that of TPSDP-OD and TPSDP-D, respectively. Although there is no significant difference among these three strategies on both DGS and EGS instances according to p-value, TPSDP-O can find the best solution on most of instances, and has lower computational complexity compared with TPSDP-OD and TPSDP-O. Therefore, TPSDP-O is adopted in the proposed population decrease strategy.

6.2.2 Discussion of the importance of components of TPSDP

In this part, some discussions are given to further analyze the importance of each phase of the proposed TPSDP. Experimental results of seven different algorithms, including the proposed TPSDP and six variants that adopt only one or two phases, on Geo, RanReal, and RanInt instances are listed in Tables 6.6, 6.7 and 6.8, respectively. For simplisity, *phases* 1, 2, and 3 represent the undirected perturbation phase, population reconstruction phase, and directed perturbation phase, respectively. To more precisely assess the performance of each algorithm, I use the Friedman test as a statistical analysis method to give a statistical result and rank each one according to their experimental results. The average ranking of seven algorithms on the given test instances are records in the row ‘ Average ranking’, where the lower rank, the better performance on the test instances. In the last row of the table, p-value is obtained by the Friedman test to verify whether there is a significant difference between the top-1 algorithm and its peers in terms of f_{avg}.

Table 6.6 summarizes the experimental results of TPSDP and its variants on Geo instances. From it, it can be found that algorithms with high exploitation ability like ‘*phase* 2+3’ and ‘*phase* 3’ can obtain best values on more instances, confirming the indispensability of *phase* 2 and *phase* 3 for Geo-type benchmark instances. Furthermore, ‘*phase* 2+3’ and TPSDP get the 1^{st} and 2^{rd} rank among all algorithms, indicating that the algorithm consists of ‘*phase* 2+3’ gets the best performance, followed by TPSDP.

Tables 6.7 and 6.8 report the experimental results of TPSDP and its variants on RanReal and RanInt instances. Contrary to the performance of Geo instances, *phase*1 plays a more important role in these two test sets. Moreover, for large-scale instances, simpler and exploration-biased algorithms, such as ‘*phase* 1+2’, ‘*phase* 1+3’, tend to yield more best values than other variants. The proposed TPSDP, which balances exploration and exploitation, is outstanding in small-scale instances. Besides, although the p-values of the second, third and fifth algorithms are greater than 0.05, which shows that TPSDP is not significantly different from these algorithms, the lowest rank indicates that TPSDP has the best performance on all instances.

According to Tables 6.6, 6.7 and 6.8, it can be concluded that TPSDP is the best performing algorithm overall on these three benchmark sets. Each phase in the algorithm has its importance, and the three phases complement each other to enhance the performance of the proposed TPSDP.

6.2.3 Discussion of rationality of TPSDP

As described in [32], in MDGP, high-quality local optimum solutions are not uniformly distributed but tend to cluster in the same or neighbor regions. Therefore, the distance between high-quality local optimum solutions in a particular region is generally small, which reveals that it is necessary to enhance the ability of the algorithm to exploit nearby better local optimum solutions in the current region. Moreover, the local optimum solutions located in distinct regions are usually far away from each other, which means that the search algorithm must have the ability to explore from one region across to another more distant region if it wants to search for higher quality solutions in other regions.

Based on the above facts, the rationality of the TPSDP algorithm is evident. In the first phase, since the initial input solutions are already local optimal in their respective current regions, the undirected perturbation operator helps them jump out of their respective regions and move to other distant regions. This process expands the search scope of the algorithm in the solution space and reinforces the exploration capability of the algorithm. The directed perturbation operator used in the third phase is less perturbative than the undirected perturbation and focuses more on exploiting better local optimum solutions clustered around the local optima. This phase aims to strengthen the local exploitation capability of the algorithm. The second phase is equivalent to the transition phase between the first phase and the third phase. The crossover process of each solution in the population can facilitate the interaction of information between different local optimal solutions, and thus visit more promising regions. The iterative execution of the three phases addresses the tradeoff between diversification and intensification of the algorithm search process, thus allowing the TPSDP algorithm to search for higher-quality solutions.

Table 6.4: Comparison of TPSDP with three method of decreasing the population size on the 36 DGS instances.

	f_{best}			f_{avg}		
Instance	**TPSDP-O**	**TPSDP-OD**	**TPSDP-D**	**TPSDP-O**	**TPSDP-OD**	**TPSDP-D**
Geo_n120_ds_01	111856.62	111852.28	**111866.09**	111829.61	111825.34	**111831.59**
Geo_n120_ds_02	61897.26	61892.77	**61899.54**	61885.03	61884.62	**61885.69**
Geo_n120_ds_03	52069.58	**52072.80**	52068.88	52062.79	52062.32	**52063.21**
Geo_n240_ds_01	**200330.83**	200318.64	200328.60	**200304.96**	200302.45	200302.75
Geo_n240_ds_02	348387.33	**348396.91**	348384.95	348344.29	348346.62	**348354.41**
Geo_n240_ds_03	**217136.17**	217116.89	217122.69	**217110.07**	217101.41	217107.41
Geo_n480_ds_01	582295.29	**582333.25**	582318.38	**580879.66**	580664.81	580864.34
Geo_n480_ds_02	**1089645.66**	1089611.96	1089590.66	**1088735.39**	1088307.00	1088300.34
Geo_n480_ds_03	**664113.65**	664082.35	664089.51	661709.64	**661715.99**	661708.67
Geo_n960_ds_01	3364324.33	**3364430.73**	3364314.77	3363512.42	**3363600.33**	3363155.90
Geo_n960_ds_02	1723413.83	**1723425.49**	1723421.37	1722434.21	**1722608.26**	1722124.33
Geo_n960_ds_03	**3350750.01**	3350706.53	3350669.34	3349845.47	**3350584.08**	3350099.90
RanReal_n120_ds_01	**50601.64**	50552.98	50595.30	50478.89	**50490.51**	50481.80
RanReal_n120_ds_02	50839.04	**50932.09**	50929.62	50746.99	**50748.02**	50728.66
RanReal_n120_ds_03	49952.03	**49955.07**	49955.00	49866.08	49861.11	**49878.16**
RanReal_n240_ds_01	**160248.01**	160177.97	160236.88	159967.54	159958.67	**159972.94**
RanReal_n240_ds_02	**160935.80**	160829.53	160794.19	**160649.22**	160588.65	160617.09
RanReal_n240_ds_03	**159735.48**	159584.04	159723.06	**159397.52**	159358.04	159344.21
RanReal_n480_ds_01	388727.06	**389012.98**	388877.51	388151.16	388347.98	**388389.65**
RanReal_n480_ds_02	387456.01	387315.75	**387734.96**	386645.62	386680.26	**386765.37**
RanReal_n480_ds_03	**389035.97**	388649.74	388725.83	**388224.90**	388060.05	388146.62
RanReal_n960_ds_01	1240505.69	1240797.08	**1242177.36**	1239589.88	1239723.89	**1239765.30**
RanReal_n960_ds_02	1240487.47	1240751.33	**1240976.92**	1239232.07	1239299.70	**1239596.09**
RanReal_n960_ds_03	**1239399.76**	1238677.02	1239279.59	1238111.31	1238031.49	**1238355.87**
RanInt_n120_ds_01	51146.00	51140.00	**51161.00**	**51075.85**	51067.45	51074.55
RanInt_n120_ds_02	51426.00	51388.00	**51480.00**	51274.15	**51303.95**	51294.65
RanInt_n120_ds_03	50258.00	**50260.00**	50258.00	50190.35	50187.40	**50205.65**
RanInt_n240_ds_01	160673.00	**160720.00**	160543.00	**160463.10**	160399.70	160350.70
RanInt_n240_ds_02	160439.00	160432.00	**160467.00**	**160260.20**	160207.90	160217.90
RanInt_n240_ds_03	160360.00	160320.00	**160393.00**	**160191.55**	160174.10	160104.35
RanInt_n480_ds_01	391004.00	**391430.00**	391161.00	390400.35	**390511.25**	390414.75
RanInt_n480_ds_02	**389889.00**	389601.00	389428.00	**388978.40**	388816.25	388835.05
RanInt_n480_ds_03	**389145.00**	388923.00	389142.00	**388264.95**	388211.80	388210.50
RanInt_n960_ds_01	1244260.00	1243736.00	**1244438.00**	**1243067.75**	1242760.30	1242851.75
RanInt_n960_ds_02	**1242579.00**	1242121.00	1242386.00	1240864.90	1240717.60	**1240924.55**
RanInt_n960_ds_03	1241504.00	**1243611.00**	1242981.00	1240433.50	1240516.70	**1240834.55**
Avg.	633800.76	633809.95	**633886.64**	**633199.44**	633195.17	633198.87
#Best	**14**	12	10	**14**	8	**14**
p-value		0.2004	1		0.427554	0.819799

Table 6.5: Comparison of TPSDP with three method of decreasing the population size on the 36 EGS instances.

	f_{best}			f_{avg}		
Instance	**TPSDP-O**	**TPSDP-D**	**TPSDP-OD**	**TPSDP-O**	**TPSDP-OD**	**TPSDP-D**
Geo_n120_ss_01	**101595.35**	101571.83	101573.32	**101557.88**	101554.91	101554.31
Geo_n120_ss_02	54828.04	54827.33	**54834.26**	54820.82	**54822.35**	54821.55
Geo_n120_ss_03	**47618.29**	47613.68	47614.15	**47611.03**	47610.13	47610.67
Geo_n240_ss_01	188818.23	**188820.82**	188817.06	188807.90	188807.03	**188808.35**
Geo_n240_ss_02	330201.63	**330217.66**	330211.60	330165.08	**330170.40**	330167.82
Geo_n240_ss_03	207000.90	**207005.11**	207003.60	206988.39	206988.15	**206990.73**
Geo_n480_ss_01	**552017.42**	551990.49	551991.84	**551975.01**	551967.91	551968.73
Geo_n480_ss_02	1047039.66	**1047083.26**	1047056.54	1047003.95	**1047012.66**	1047007.24
Geo_n480_ss_03	633501.32	633515.29	**633518.44**	633478.65	633483.96	**633485.82**
Geo_n960_ss_01	3253623.12	3253651.40	**3253669.94**	3253562.37	3253570.07	**3253574.79**
Geo_n960_ss_02	**1663389.17**	1663378.93	1663373.45	**1663357.67**	1663351.44	1663352.30
Geo_n960_ss_03	**3250931.07**	3250920.09	3250846.24	3250815.34	**3250825.72**	3250798.55
RanReal_n120_ss_01	**47363.21**	**47363.21**	**47363.21**	**47339.88**	47334.16	47333.78
RanReal_n120_ss_02	**47243.16**	**47243.16**	**47243.16**	**47205.03**	47203.74	47199.82
RanReal_n120_ss_03	**47313.71**	**47313.71**	**47313.71**	47272.67	**47284.89**	47276.32
RanReal_n240_ss_01	**155246.47**	155241.93	155214.65	**155082.72**	155064.22	155039.51
RanReal_n240_ss_02	**155670.84**	155656.23	155585.69	155494.41	155466.29	**155496.25**
RanReal_n240_ss_03	**155765.30**	155704.75	155699.22	**155575.34**	155544.05	155543.41
RanReal_n480_ss_01	378237.23	378306.93	**378623.12**	377583.69	**377708.85**	377501.85
RanReal_n480_ss_02	377656.61	**377846.64**	377768.66	377144.60	377241.93	**377324.90**
RanReal_n480_ss_03	**378781.51**	378705.63	378644.02	378168.56	378245.86	**378250.78**
RanReal_n960_ss_01	**1218220.05**	1217601.56	1217667.28	1216601.20	**1216700.76**	1216641.42
RanReal_n960_ss_02	1219277.34	**1220101.38**	1218916.60	1217968.76	1217979.31	**1218033.57**
RanReal_n960_ss_03	1218593.64	1218555.78	**1218608.23**	**1217429.29**	1217149.45	1217385.24
RanInt_n120_ss_01	**47909.00**	**47909.00**	**47909.00**	**47909.00**	**47909.00**	47903.80
RanInt_n120_ss_02	**47826.00**	**47826.00**	**47826.00**	**47825.05**	47824.25	47823.00
RanInt_n120_ss_03	**47552.00**	**47552.00**	**47552.00**	47477.35	**47485.15**	47484.65
RanInt_n240_ss_01	155526.00	155577.00	**155588.00**	**155450.10**	155432.75	155412.65
RanInt_n240_ss_02	155358.00	**155378.00**	155358.00	155230.55	**155261.05**	155175.20
RanInt_n240_ss_03	**156415.00**	**156415.00**	**156415.00**	**156328.35**	156288.60	156289.90
RanInt_n480_ss_01	**380081.00**	379887.00	379999.00	379357.75	**379419.45**	379411.80
RanInt_n480_ss_02	380581.00	380218.00	**380737.00**	**379793.80**	379768.10	379546.70
RanInt_n480_ss_03	378997.00	**379288.00**	379114.00	378448.45	378426.15	**378562.15**
RanInt_n960_ss_01	1220727.00	**1221176.00**	1220768.00	1219704.65	1219714.25	**1219798.90**
RanInt_n960_ss_02	**1220336.00**	1220325.00	1220270.00	**1219444.90**	1219265.10	1219282.20
RanInt_n960_ss_03	1221586.00	1221301.00	**1221632.00**	**1220065.60**	1219849.45	1219943.50
Avg.	615078.53	**615085.80**	615064.61	**614723.49**	614714.77	614716.73
#Best	**19**	16	15	**16**	11	10
p-value		0.78067	0.819799		0.928219	0.555763

Table 6.6: Comparison of seven TPSDP variants on the 24 Geo instances.

	f_{avg}						
Instance	**TPSDP**	$phase1+2$	$phase1+3$	$phase2+3$	$phase1$	$phase2$	$phase3$
Geo_n120_ds_01	111831.56	111830.66	111546.42	111841.41	111313.91	**111857.53**	111248.40
Geo_n120_ds_02	61883.69	61882.08	61739.78	**61887.50**	61863.92	61866.16	61628.89
Geo_n120_ds_03	52063.21	52064.27	51808.16	**52068.54**	51923.28	52060.92	51740.77
Geo_n240_ds_01	200306.53	200299.12	199868.24	**200321.90**	199980.36	200283.11	199778.62
Geo_n240_ds_02	348349.00	348345.58	347389.26	348364.26	347580.05	**348418.44**	346769.33
Geo_n240_ds_03	217106.84	217107.65	216944.08	**217126.21**	217024.98	217070.47	216766.51
Geo_n480_ds_01	580550.25	**581189.85**	580665.88	578604.17	581036.63	577297.27	579288.71
Geo_n480_ds_02	1087807.40	**1088822.38**	1088333.07	1085402.88	1088670.21	1084225.18	1085326.78
Geo_n480_ds_03	661980.97	**662813.32**	661511.11	660176.24	662318.04	659698.33	660358.85
Geo_n960_ds_01	3363298.19	3363227.35	3363233.57	3351189.58	**3363432.32**	3346451.50	3349293.02
Geo_n960_ds_02	1722147.18	1722166.75	1722346.80	1718740.12	**1722610.64**	1716501.14	1717001.96
Geo_n960_ds_03	3350369.56	3350414.89	**3350491.80**	3341812.31	3350441.01	3340560.26	3340423.82
Geo_n120_ss_01	101553.75	101558.07	101555.72	101564.93	101562.28	101517.38	**101571.09**
Geo_n120_ss_02	54823.16	54820.35	54822.05	54826.97	54822.64	54815.82	**54829.05**
Geo_n120_ss_03	47609.63	47611.53	47611.53	**47614.17**	47611.04	47605.02	47613.75
Geo_n240_ss_01	188807.34	188803.99	188810.56	188823.10	188806.01	188801.10	**188828.23**
Geo_n240_ss_02	330174.32	330162.32	330170.67	330192.18	330170.42	330156.19	**330197.32**
Geo_n240_ss_03	206987.04	206983.13	206993.41	206998.68	206990.75	206984.23	**207003.81**
Geo_n480_ss_01	551968.13	551949.81	551972.32	**552010.25**	551952.80	551786.34	552005.70
Geo_n480_ss_02	1047008.22	1046966.50	1047020.38	**1047104.07**	1046969.50	1046528.48	1047088.69
Geo_n480_ss_03	633483.30	633463.71	633488.16	633513.28	633470.01	633231.74	**633519.02**
Geo_n960_ss_01	**3253760.52**	3253473.10	3253560.02	3253733.54	3253486.97	3252719.64	3253755.38
Geo_n960_ss_02	1663372.45	1663317.55	1663350.39	1663443.68	1663321.37	1663155.93	**1663448.39**
Geo_n960_ss_03	3250909.92	3250721.32	3250810.36	3251005.74	3250721.13	3250017.66	**3251010.85**
Avg.	962006.34	**962083.14**	961918.49	960765.24	962003.34	960150.41	960437.37
#Best	1	3	1	7	2	2	**8**
Average ranking	3.4583	4.0833	4	**2.5833**	4.08333	5.9167	3.875
p-value	0.160581	0.016157	0.023103	-	0.016157	0.000000	0.038333

Table 6.7: Comparison of seven TPSDP variants on the 24 RanReal instances.

	f_{avg}						
Instance	**TPSDP**	*phase*1 + 2	*phase*1 + 3	*phase*2 + 3	*phase*1	*phase*2	*phase*3
RanReal_n120_ds_01	**50476.93**	50457.95	50434.49	50235.90	50393.46	50119.42	49445.37
RanReal_n120_ds_02	50727.34	50647.20	50730.95	50482.04	**50744.93**	50334.21	49811.73
RanReal_n120_ds_03	**49879.67**	49840.79	49864.56	49722.59	49856.85	49665.63	48901.63
RanReal_n240_ds_01	**159974.22**	159911.38	159912.32	159094.66	159884.99	158671.87	156906.62
RanReal_n240_ds_02	**160643.87**	160607.73	160417.83	159813.12	160279.28	159146.97	157367.89
RanReal_n240_ds_03	**159419.88**	159321.42	159266.43	158571.05	159246.94	158253.49	156644.47
RanReal_n480_ds_01	388326.45	387979.90	**388427.09**	382494.84	388105.64	374302.49	382691.31
RanReal_n480_ds_02	386673.60	386229.39	**386797.82**	380619.88	386275.74	373188.26	380994.76
RanReal_n480_ds_03	388104.73	387679.09	**388187.85**	382147.71	387508.04	374506.22	382543.27
RanReal_n960_ds_01	1239891.19	1239877.23	1239622.00	1226878.50	**1240187.54**	1202857.12	1224837.86
RanReal_n960_ds_02	1239515.63	1239735.32	1239622.33	1225129.68	**1239860.81**	1204259.10	1225481.82
RanReal_n960_ds_03	1237910.82	**1238315.83**	1238279.22	1224390.38	1238258.24	1202591.78	1224838.11
RanReal_n120_ss_01	**47343.68**	47329.92	47298.59	46998.16	47256.51	46600.96	46766.73
RanReal_n120_ss_02	47197.48	**47202.90**	47174.81	46861.17	47165.42	46536.49	46868.29
RanReal_n120_ss_03	47276.01	**47283.19**	47266.45	47010.85	47244.20	46631.55	46887.51
RanReal_n240_ss_01	155041.88	**155090.59**	155002.80	153808.11	155023.28	152953.38	153560.78
RanReal_n240_ss_02	155451.66	**155490.10**	155378.64	154237.95	155339.97	153287.80	153856.33
RanReal_n240_ss_03	**155566.95**	155555.63	155474.56	154225.62	155475.45	153363.70	153807.15
RanReal_n480_ss_01	377533.30	376880.99	**377571.01**	375729.29	376767.27	364968.68	375857.25
RanReal_n480_ss_02	377082.09	376584.86	**377105.88**	375278.37	376527.54	364357.94	374476.06
RanReal_n480_ss_03	378240.08	377637.78	**378246.49**	376648.86	377433.12	366317.01	376687.69
RanReal_n960_ss_01	1216503.80	1216532.90	1216439.09	1209137.30	**1216786.63**	1184058.73	1210441.27
RanReal_n960_ss_02	1217820.76	1217729.74	**1217828.43**	1212651.32	1217809.63	1183602.14	1213017.91
RanReal_n960_ss_03	1217087.01	1217231.53	**1217411.70**	1211737.69	1217288.48	1183808.16	1211735.95
Avg.	454320.38	454214.72	**454323.39**	450579.38	454196.66	441849.30	450184.49
#Best	7	5	**8**	0	4	0	0
Average ranking	**2.0417**	2.5	2.3333	5.4167	3.125	6.75	5.8333
p-value	-	0.462359	0.639994	0	0.082352	0	0

Table 6.8: Comparison of seven TPSDP variants on the 24 RanInt instances.

	f_{avg}						
Instance	**TPSDP**	*phase*1 + 2	*phase*1 + 3	*phase*2 + 3	*phase*1	*phase*2	*phase*3
RanInt_n120_ds_01	**51075.85**	51048.80	51049.75	50868.55	51029.25	50713.90	49871.85
RanInt_n120_ds_02	**51294.65**	51266.50	51293.70	51069.15	51279.80	50994.75	50375.15
RanInt_n120_ds_03	**50192.05**	50158.75	50181.80	50027.50	50175.80	49890.65	49077.05
RanInt_n240_ds_01	160358.10	**160383.60**	160340.55	159318.30	160271.90	158786.95	157544.80
RanInt_n240_ds_02	**160277.70**	160244.50	160104.45	159314.80	159927.95	159040.40	157089.00
RanInt_n240_ds_03	**160223.05**	160122.95	160130.55	159173.55	160097.05	158767.05	156816.25
RanInt_n480_ds_01	390362.25	389970.60	**390592.50**	383944.55	389983.05	375807.80	384366.05
RanInt_n480_ds_02	388743.45	388476.20	**389079.95**	383277.80	388214.75	375324.20	382117.60
RanInt_n480_ds_03	**388362.50**	387679.00	388289.70	381820.70	387652.45	374105.05	381989.10
RanInt_n960_ds_01	1242857.55	1243067.85	1243189.80	1228315.80	**1243191.15**	1206882.55	1229971.90
RanInt_n960_ds_02	1240869.75	**1241168.80**	1240956.00	1227200.60	1241082.05	1203859.05	1226079.65
RanInt_n960_ds_03	**1240896.00**	1240787.20	1240661.25	1227297.25	1240742.05	1204952.25	1226001.95
RanInt_n120_ss_01	**47909.00**	**47909.00**	47894.25	47586.05	47893.70	47181.25	47423.15
RanInt_n120_ss_02	**47822.80**	47814.80	47819.50	47471.90	47781.80	47060.05	47382.80
RanInt_n120_ss_03	**47489.10**	47457.40	47454.60	47233.50	47441.70	46827.25	47135.05
RanInt_n240_ss_01	155440.20	**155446.85**	155403.70	154322.10	155341.30	152930.35	153800.70
RanInt_n240_ss_02	155207.25	**155259.40**	155167.15	153838.50	155129.70	152942.40	153586.35
RanInt_n240_ss_03	156319.40	**156343.35**	156285.55	155106.05	156221.30	153640.00	154188.75
RanInt_n480_ss_01	379263.95	378716.45	**379319.75**	377559.75	378557.25	366746.55	377004.90
RanInt_n480_ss_02	379596.55	379254.40	**379733.90**	377345.40	379089.85	366459.15	376733.90
RanInt_n480_ss_03	378291.55	377899.75	**378545.35**	376848.10	378024.10	365596.10	376488.70
RanInt_n960_ss_01	1219694.55	1219886.10	**1219929.25**	1214135.05	1219905.90	1185752.05	1214104.80
RanInt_n960_ss_02	1219184.35	**1219497.75**	1219487.40	1213667.05	1219441.75	1185538.70	1213840.75
RanInt_n960_ss_03	1220117.65	**1220216.70**	1219878.40	1214731.05	1219894.55	1186619.75	1214050.85
Avg.	455493.72	455419.86	**455532.87**	451728.04	455348.76	442767.43	451126.71
#Best	**10**	8	6	0	1	0	0
Average ranking	**1.9375**	2.3958	2.25	5.1667	3.4167	6.75	6.0833
p-value	-	0.462359	0.61629	0	0.017695	0	0

Chapter 7

Conclusion and future works

The paper presented a three-phase search approach with dynamic population size (TPSDP) for solving the maximally diverse grouping problem (MDGP). The three phases of TPSDP coordinate with each other and help achieve a desirable balance between diversification and intensification during the search process. Moreover, TPSDP also integrates a decline of population size strategy to avoid the waste of computing resources on non-promising solutions and ensures the algorithm is more effective. Extensive computational results based on widely used benchmark sets (RanInt, RanReal, Geo, MDG-a, and MDG-c) indicated that TPSDP is highly competitive compared to best-performing MDGP algorithms. Also, TPSDP significantly outperforms its peers, especially on the small-scale instances, but only performs worse than NDHA on the large-scale instances. Furthermore, to illuminate the adequacy of the TPSDP algorithm, I carried out some experiments to analyze the influence of some crucial parameters. Also, I discussed the importance and rationality of the structure of the proposed algorithm.

The ideas of the population-based method and the framework of the three-phase search approach are rather general. It is practicable to use these two methods to solve other CO problems, such as the clique partitioning problem (CPP) what I am studying. For the purpose of proposing more general and understandable algorithms and solving large-scale problems more effectively, further research could be done in the theoretical explanation of the working principle of heuristic search and designing more efficient and effective local search.

Table 7.1: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large DGS instances with n = 2000, m = 10, L_g = 173, and U_g = 227.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	10	173	227	1134788	1135640	**1135964**	1135772	1134068.30	1134977.10	**1135455.30**	1135436.05
MDG-a_22	10	173	227	1134340	1135172	**1135868**	1135545	1133858.05	1134680.25	**1135162.65**	1135136.00
MDG-a_23	10	173	227	1134013	1134871	1135121	**1135303**	1133485.95	1134396.25	1134685.60	**1134827.20**
MDG-a_24	10	173	227	1134308	1135264	**1135584**	1135567	1133832.65	1134779.35	**1135122.35**	1135088.85
MDG-a_25	10	173	227	1135107	1135859	**1136225**	1136098	1134295.70	1135342.50	1135504.50	**1135617.00**
MDG-a_26	10	173	227	1134167	1135330	1135373	**1135449**	1133689.60	1134808.30	1134880.35	**1135013.35**
MDG-a_27	10	173	227	1134392	1134825	**1135624**	1135623	1133472.15	1134426.50	**1134738.15**	1134731.80
MDG-a_28	10	173	227	1134564	1135370	**1135830**	1135505	1133780.15	1134842.90	**1135127.60**	1135055.85
MDG-a_29	10	173	227	1134701	1135799	**1135978**	1135925	1134086.40	1135029.00	**1135320.90**	1135234.35
MDG-a_30	10	173	227	1134523	1135152	**1135486**	1135468	1133765.65	1134712.40	1134873.25	**1135068.95**
MDG-a_31	10	173	227	1135233	1136064	1136026	**1136119**	1134395.90	1135351.50	1135580.25	**1135749.20**
MDG-a_32	10	173	227	1134596	1135504	**1135847**	1135549	1134041.75	1135057.80	**1135235.95**	1135199.65
MDG-a_33	10	173	227	1135317	1135217	**1135544**	1135331	1133825.30	1134733.50	**1135157.65**	1134948.75
MDG-a_34	10	173	227	1134692	1135496	**1135890**	1135756	1134165.85	1135148.75	**1135527.75**	1135338.75
MDG-a_35	10	173	227	1134486	1135402	**1135670**	1135317	1133773.20	1134753.55	**1135099.55**	1135004.45
MDG-a_36	10	173	227	1134428	1135364	**1135846**	1135475	1133975.75	1134902.10	**1135243.05**	1135079.95
MDG-a_37	10	173	227	1135195	1135953	**1136590**	1135958	1134208.20	1135380.60	**1135806.55**	1135576.70
MDG-a_38	10	173	227	1134872	1135777	**1136123**	1135679	1134253.95	1135104.85	**1135510.45**	1135342.20
MDG-a_39	10	173	227	1134272	1135340	**1135724**	1135561	1133667.75	1134745.10	**1135073.45**	1134887.85
MDG-a_40	10	173	227	1134904	1136117	**1136593**	1136473	1134551.85	1135631.90	1135972.50	**1135976.25**
Avg.				1134644.90	1135475.80	**1135845.30**	1135673.65	1133959.71	1134940.21	**1135253.89**	1135215.66
#Best				0	0	**17**	3	0	0	**14**	6
p-value				0.000082	0.000315	1		0.000082	0.000082	1	

Table 7.2: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large DGS instances with n = 2000, m = 25, L_g = 51, and U_g = 109.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	25	51	109	540045	541275	**541857**	541441	539589.00	540616.50	**541292.20**	540884.55
MDG-a_22	25	51	109	539996	541035	**541775**	541298	539431.75	540424.00	**541256.25**	540843.65
MDG-a_23	25	51	109	539871	540792	**541495**	541041	539349.70	540423.70	**541063.30**	540638.35
MDG-a_24	25	51	109	540067	541017	**541722**	541101	539559.70	540637.95	**541278.80**	540828.65
MDG-a_25	25	51	109	539985	541111	**541808**	541357	539628.10	540697.45	**541474.65**	540915.70
MDG-a_26	25	51	109	539857	540986	**541661**	541166	539533.85	540557.35	**541288.95**	540871.25
MDG-a_27	25	51	109	539692	540849	**541478**	540864	539238.45	540375.35	**541099.60**	540621.80
MDG-a_28	25	51	109	540098	541260	**541730**	541317	539512.30	540571.75	**541239.90**	540822.60
MDG-a_29	25	51	109	540220	540889	**541592**	541208	539689.40	540569.30	**541349.05**	540867.15
MDG-a_30	25	51	109	540105	540838	**541849**	541043	539444.00	540514.15	**541305.75**	540784.35
MDG-a_31	25	51	109	540320	541095	**541879**	541474	539717.70	540768.15	**541451.20**	541058.20
MDG-a_32	25	51	109	540134	541032	**541635**	541215	539723.50	540650.45	**541167.35**	540885.90
MDG-a_33	25	51	109	540039	540863	**541834**	541439	539442.50	540520.40	**541232.45**	540893.10
MDG-a_34	25	51	109	540316	540843	**541601**	541230	539691.35	540546.50	**541279.15**	540985.65
MDG-a_35	25	51	109	540108	540977	**541836**	541581	539486.35	540405.60	**541177.25**	540867.95
MDG-a_36	25	51	109	539950	541045	**541729**	541446	539534.40	540497.20	**541267.85**	540904.20
MDG-a_37	25	51	109	540364	541339	**541931**	541651	539914.40	540750.00	**541397.65**	541007.20
MDG-a_38	25	51	109	539987	541284	**541686**	541417	539487.05	540689.40	**541399.70**	540918.40
MDG-a_39	25	51	109	539993	541124	**541477**	541205	539419.00	540480.50	**541026.85**	540760.10
MDG-a_40	25	51	109	540277	541382	**542067**	541573	539846.95	540951.15	**541667.35**	541160.40
Avg.				540071.20	541051.80	**541732.10**	541303.35	539561.97	540582.34	**541285.76**	540875.96
#Best				0	0	**20**	0	0	0	**20**	0
p-value				0.000082	0.000082	1		0.000082	0.000082	1	

Table 7.3: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large DGS instances with n = 2000, m = 50, L_g = 26, and U_g = 54.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	50	26	54	291299	292539	**293496**	292735	290987.30	292296.95	**293091.45**	292562.80
MDG-a_22	50	26	54	291903	292590	**293428**	292915	291108.75	292283.15	**292969.80**	292630.20
MDG-a_23	50	26	54	291583	292613	**293277**	292797	291054.60	292241.75	**292933.45**	292496.50
MDG-a_24	50	26	54	291369	292485	**293328**	292951	291026.45	292239.90	**292958.15**	292597.65
MDG-a_25	50	26	54	291523	292766	**293264**	292901	291075.95	292403.35	**293046.30**	292596.15
MDG-a_26	50	26	54	291578	292515	**293403**	292981	291110.15	292317.60	**293029.95**	292618.35
MDG-a_27	50	26	54	291281	292691	**293257**	292857	291018.75	292266.70	**292825.15**	292482.35
MDG-a_28	50	26	54	291409	292511	**293355**	292861	290989.40	292273.45	**293080.40**	292595.55
MDG-a_29	50	26	54	291396	292770	**293387**	292976	291042.05	292320.90	**293079.80**	292692.60
MDG-a_30	50	26	54	291536	292770	**293524**	293000	291149.35	292324.20	**292998.10**	292546.65
MDG-a_31	50	26	54	291593	292782	**293380**	293329	291198.20	292424.40	**293185.35**	292743.10
MDG-a_32	50	26	54	291607	292655	**293426**	292851	291102.25	292324.95	**292747.05**	292595.85
MDG-a_33	50	26	54	291491	292526	**293171**	292850	291056.55	292354.85	**293003.60**	292555.35
MDG-a_34	50	26	54	291601	292546	**293515**	292963	291105.85	292300.35	**293138.45**	292692.75
MDG-a_35	50	26	54	291478	292801	**293370**	292827	291060.85	292258.00	**292922.50**	292571.50
MDG-a_36	50	26	54	291655	292506	**293258**	292840	291148.00	292277.10	**293008.40**	292561.00
MDG-a_37	50	26	54	291528	292736	**293378**	292872	291182.70	292369.20	**293036.45**	292651.90
MDG-a_38	50	26	54	291468	292789	**293477**	292926	291120.55	292320.20	**293018.30**	292639.00
MDG-a_39	50	26	54	291353	292570	**293260**	292834	291048.10	292226.65	**292847.60**	292552.65
MDG-a_40	50	26	54	291614	292885	**293428**	292946	291212.85	292444.00	**293225.55**	292742.50
Avg.				291513.25	292652.30	**293369.10**	292910.60	291089.93	292313.38	**293007.29**	292606.22
#Best				0	0	**20**	0	0	0	**20**	0
p-value				0.000082	0.00007	1		0.000082	0.000082	1	

Table 7.4: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large DGS instances with n = 2000, m = 50, L_g = 32, and U_g = 48.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	50	32	48	272980	274316	**274975**	274801	272714.15	273977.60	**274591.45**	274446.55
MDG-a_22	50	32	48	273028	274357	**274923**	274490	272601.50	274021.90	**274556.10**	274331.15
MDG-a_23	50	32	48	273164	274442	**274805**	274520	272639.05	273998.10	**274482.05**	274253.25
MDG-a_24	50	32	48	273142	274453	**275024**	274698	272662.15	273970.60	**274579.15**	274431.05
MDG-a_25	50	32	48	273057	274560	**275095**	274578	272691.90	274131.80	**274652.55**	274349.55
MDG-a_26	50	32	48	273231	274271	274786	**274840**	272671.95	274003.00	**274536.95**	274418.75
MDG-a_27	50	32	48	272859	274303	274622	**274649**	272554.40	273933.70	**274439.60**	274311.00
MDG-a_28	50	32	48	273217	274175	**274893**	274647	272622.65	273966.85	**274571.35**	274283.85
MDG-a_29	50	32	48	273158	274278	**274853**	274647	272752.95	274040.45	**274553.50**	274349.40
MDG-a_30	50	32	48	273119	274398	**274990**	274800	272668.10	273966.70	**274597.25**	274370.90
MDG-a_31	50	32	48	273067	274440	**274926**	274641	272703.15	274107.30	**274720.10**	274516.50
MDG-a_32	50	32	48	273201	274381	**274886**	274689	272582.95	273975.80	**274383.95**	274362.50
MDG-a_33	50	32	48	273155	274437	**275054**	274623	272690.45	274042.10	**274573.75**	274368.75
MDG-a_34	50	32	48	273249	274334	**274936**	274551	272677.60	274081.15	**274646.75**	274383.85
MDG-a_35	50	32	48	273030	274159	**274794**	274533	272662.90	273949.80	**274454.70**	274248.95
MDG-a_36	50	32	48	273103	274240	**274918**	274745	272670.90	274032.65	**274597.70**	274320.75
MDG-a_37	50	32	48	273292	274288	**275142**	274581	272711.25	274049.75	**274678.25**	274366.90
MDG-a_38	50	32	48	273426	274381	**274966**	274653	272799.25	274043.70	**274537.10**	274410.85
MDG-a_39	50	32	48	273130	274214	**274743**	274472	272660.60	273949.70	**274440.00**	274299.05
MDG-a_40	50	32	48	273289	274428	**275067**	274796	272827.20	274133.80	**274701.75**	274508.20
Avg.				273144.85	274342.75	**274919.90**	274647.70	272678.25	274018.82	**274564.70**	274366.59
#Best				0	0	**18**	2	0	0	**20**	0
p-value				0.000082	0.000082	1		0.000082	0.000082	1	

Table 7.5: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large DGS instances with n = 2000, m = 100, L_g = 13, and U_g = 27.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	100	13	27	159385	160213	**160955**	160569	159011.05	159911.55	**160686.55**	160353.45
MDG-a_22	100	13	27	159187	160213	**160911**	160519	158956.75	159925.80	**160612.90**	160346.10
MDG-a_23	100	13	27	159359	160035	**160983**	160583	158998.45	159850.20	**160575.80**	160310.75
MDG-a_24	100	13	27	159372	160101	**160814**	160548	159000.95	159920.25	**160594.25**	160346.55
MDG-a_25	100	13	27	159319	160167	**160803**	160600	158996.70	159904.55	**160569.60**	160414.75
MDG-a_26	100	13	27	159249	160125	**160927**	160606	158997.20	159920.60	**160627.65**	160390.70
MDG-a_27	100	13	27	159284	160099	**160867**	160495	158945.75	159858.60	**160483.90**	160305.15
MDG-a_28	100	13	27	159358	160157	**160997**	160472	158998.10	159923.75	**160601.30**	160314.60
MDG-a_29	100	13	27	159226	160192	**160888**	160598	159032.55	159877.65	**160572.00**	160389.70
MDG-a_30	100	13	27	159342	160151	**160813**	160510	158960.15	159945.90	**160585.50**	160315.10
MDG-a_31	100	13	27	159354	160125	**161100**	160600	159086.90	159983.00	**160725.45**	160410.80
MDG-a_32	100	13	27	159466	160062	**160802**	160463	159057.25	159900.75	**160424.70**	160324.05
MDG-a_33	100	13	27	159395	160210	**161064**	160512	158979.00	159910.15	**160652.00**	160371.05
MDG-a_34	100	13	27	159332	160104	**160791**	160556	159007.10	159927.45	**160564.45**	160401.20
MDG-a_35	100	13	27	159384	160103	**160863**	160530	158975.05	159893.15	**160573.90**	160296.35
MDG-a_36	100	13	27	159354	160123	**161000**	160563	159028.90	159872.05	**160679.00**	160406.35
MDG-a_37	100	13	27	159375	160093	**160871**	160644	159019.25	159904.55	**160619.20**	160438.90
MDG-a_38	100	13	27	159427	160143	**160796**	160573	159042.05	159912.85	**160523.65**	160362.50
MDG-a_39	100	13	27	159450	160097	**160812**	160435	158979.50	159837.35	**160440.45**	160320.25
MDG-a_40	100	13	27	159614	160231	**161047**	160596	159113.30	159955.20	**160702.95**	160416.70
Avg.				159361.60	160137.20	**160905.20**	160548.60	159009.30	159906.77	**160590.76**	160361.75
#Best				0	0	**20**	0	0	0	**20**	0
p-value				0.000082	0.000082	1		0.000082	0.000082	1	

Table 7.6: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large DGS instances with n = 2000, m = 200, L_g = 6, and U_g = 14.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	200	6	14	88816	88917	**89166**	88711	88596.05	88756.20	**88985.10**	88534.90
MDG-a_22	200	6	14	88688	88907	**89312**	88693	88552.00	88740.25	**89028.80**	88532.90
MDG-a_23	200	6	14	88711	88882	**89144**	88788	88550.75	88745.30	**88943.15**	88544.00
MDG-a_24	200	6	14	88798	88855	**89274**	88609	88537.10	88758.35	**88969.10**	88510.40
MDG-a_25	200	6	14	88737	88943	**89283**	88665	88572.10	88779.85	**88947.70**	88537.95
MDG-a_26	200	6	14	88780	88881	**89482**	88746	88607.70	88764.05	**89000.35**	88577.05
MDG-a_27	200	6	14	88704	88855	**89172**	88506	88530.30	88720.85	**88923.70**	88434.90
MDG-a_28	200	6	14	88841	88914	**89323**	88649	88606.6	88743.70	**89033.20**	88538.15
MDG-a_29	200	6	14	88820	89085	**89307**	88574	88513.85	88740.40	**88922.40**	88495.65
MDG-a_30	200	6	14	88666	88877	**89239**	88626	88525.15	88766.45	**88982.80**	88528.95
MDG-a_31	200	6	14	88772	88958	**89138**	88658	88571.3	88805.35	**89021.15**	88543.80
MDG-a_32	200	6	14	88783	88931	**89281**	88675	88570.45	88816.70	**89067.35**	88522.75
MDG-a_33	200	6	14	88803	88888	**89329**	88665	88551.95	88792.95	**89045.25**	88553.80
MDG-a_34	200	6	14	88676	88931	**89183**	88666	88546.35	88783.10	**88968.70**	88561.40
MDG-a_35	200	6	14	88774	88938	**89253**	88681	88579.35	88778.10	**89030.85**	88555.80
MDG-a_36	200	6	14	88729	89006	**89358**	88648	88591.6	88855.85	**88998.05**	88505.90
MDG-a_37	200	6	14	88738	89008	**89357**	88654	88547.75	88843.90	**88945.90**	88550.40
MDG-a_38	200	6	14	88745	89034	**89516**	88658	88525.4	88859.50	**89069.90**	88541.50
MDG-a_39	200	6	14	88860	89008	**89293**	88733	88571.4	88796.25	**89007.45**	88519.65
MDG-a_40	200	6	14	88734	88975	**89216**	88714	88553.4	88819.35	**89015.55**	88582.15
Avg.				88758.75	88939.65	**89281.30**	88665.95	88560.03	88783.32	**88995.32**	88533.60
#Best				0	0	**20**	0	0	0	**20**	0
p-value				1	1	1		1	1	1	

Table 7.7: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large EGS instances with n = 2000, m = 10.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	10	200	200	1115956	1116736	**1117425**	1117031	1115369.15	1116255.05	**1116827.75**	1116572.10
MDG-a_22	10	200	200	1115649	1116539	1116911	**1117024**	1115247.65	1116141.60	**1116521.65**	1116399.30
MDG-a_23	10	200	200	1115440	1116259	**1116733**	1116458	1114994.40	1115850.75	**1116131.20**	1116131.10
MDG-a_24	10	200	200	1116114	1116805	1116906	**1116928**	1115390.75	1116170.55	1116386.25	**1116503.25**
MDG-a_25	10	200	200	1116511	1117136	**1117366**	1117267	1115687.45	1116448.00	**1116950.95**	1116817.65
MDG-a_26	10	200	200	1115881	1116688	**1116986**	1116758	1115247.15	1115980.55	**1116442.90**	1116340.20
MDG-a_27	10	200	200	1115352	1116299	**1116759**	1116539	1114912.05	1115799.90	**1116194.95**	1115964.60
MDG-a_28	10	200	200	1115619	1116610	**1117128**	1116881	1115145.90	1116048.90	**1116699.00**	1116436.75
MDG-a_29	10	200	200	1116073	1116813	**1117338**	1117249	1115433.90	1116396.05	**1116890.00**	1116711.30
MDG-a_30	10	200	200	1115725	1116384	**1116742**	1116690	1115225.45	1116044.75	**1116404.80**	1116387.15
MDG-a_31	10	200	200	1116325	1117449	**1117744**	1117267	1115803.25	1116749.35	**1117171.65**	1117005.70
MDG-a_32	10	200	200	1116292	1116612	**1117319**	1117028	1115437.25	1116232.20	**1116617.00**	1116593.15
MDG-a_33	10	200	200	1115535	1116594	**1117245**	1116791	1115055.75	1116097.35	**1116484.35**	1116353.30
MDG-a_34	10	200	200	1116100	1116965	**1117286**	1117099	1115587.95	1116432.75	**1116848.95**	1116641.80
MDG-a_35	10	200	200	1115504	1116521	**1116750**	1116709	1115043.60	1115942.60	**1116410.15**	1116375.50
MDG-a_36	10	200	200	1115998	1116634	**1116838**	1116723	1115200.45	1116065.75	**1116442.15**	1116365.00
MDG-a_37	10	200	200	1116442	1117093	**1117758**	1117440	1115858.45	1116489.40	**1116996.30**	1116951.95
MDG-a_38	10	200	200	1116147	1116797	**1117320**	1117209	1115418.90	1116400.90	**1116804.10**	1116673.15
MDG-a_39	10	200	200	1115617	1116707	**1117173**	1117064	1114995.10	1115918.15	**1116404.45**	1116264.95
MDG-a_40	10	200	200	1116533	1117223	**1117912**	1117769	1116044.15	1116810.95	**1117360.25**	1117240.85
Avg.				1115940.65	1116743.20	**1117181.95**	1116996.20	1115354.94	1116213.78	**1116649.44**	1116536.44
#Best				0	0	**18**	2	0	0	**19**	1
p-value				0.000082	0.000204	1		0.000082	0.000082	1	

Table 7.8: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large EGS instances with n = 2000, m = 25.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	25	80	80	486934	487747	**488254**	487870	486130.70	487286.05	**487907.65**	487516.65
MDG-a_22	25	80	80	486582	487444	**488152**	487782	486018.55	487162.55	**487677.15**	487378.30
MDG-a_23	25	80	80	486486	487378	**487996**	487654	485821.30	487177.10	**487614.95**	487256.45
MDG-a_24	25	80	80	486653	487560	**488152**	487740	486088.40	487254.85	**487762.25**	487389.60
MDG-a_25	25	80	80	486706	487728	**488149**	487864	486135.90	487374.15	**487846.35**	487499.05
MDG-a_26	25	80	80	486648	487550	**488316**	487710	486047.50	487176.85	**487804.85**	487400.05
MDG-a_27	25	80	80	486311	487238	**487877**	487541	485694.15	486961.65	**487555.40**	487216.80
MDG-a_28	25	80	80	486534	487455	**488098**	487835	486046.55	487169.75	**487798.20**	487412.55
MDG-a_29	25	80	80	486591	487639	**488150**	487767	486069.95	487303.30	**487879.90**	487379.25
MDG-a_30	25	80	80	486366	487606	**488136**	487617	485880.00	487188.25	**487723.60**	487334.00
MDG-a_31	25	80	80	486825	487732	**488532**	487931	486335.10	487501.15	**488079.15**	487674.70
MDG-a_32	25	80	80	486586	487729	**488319**	487782	486111.45	487264.40	**487884.70**	487364.40
MDG-a_33	25	80	80	486429	487529	**488349**	487581	485901.80	487238.00	**487856.85**	487357.90
MDG-a_34	25	80	80	486683	487530	**488413**	487886	486158.05	487338.85	**487899.35**	487505.80
MDG-a_35	25	80	80	486721	487537	**488186**	487663	485903.35	487134.95	**487761.80**	487352.40
MDG-a_36	25	80	80	486807	487474	**488224**	487829	486009.60	487143.00	**487749.50**	487509.50
MDG-a_37	25	80	80	486721	487652	**488414**	487824	486189.45	487338.35	**487960.75**	487470.85
MDG-a_38	25	80	80	486575	487576	**488412**	487686	486105.35	487277.65	**487874.80**	487398.90
MDG-a_39	25	80	80	486525	487450	**488341**	487609	485976.70	487152.95	**487769.05**	487362.30
MDG-a_40	25	80	80	486898	487890	**488481**	488299	486126.10	487536.85	**488098.25**	487799.25
Avg.				486629.05	487572.20	**488247.55**	487773.50	486037.50	487249.03	**487825.23**	487428.94
#Best				0	0	**20**	0	0	0	**20**	0
p-value				0.000082	0.000082	1		0.000082	0.000082	1	

Table 7.9: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large EGS instances with n = 2000, m = 50.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	50	40	40	264298	265670	266007	**266081**	263551.45	265341.90	**265751.60**	265718.70
MDG-a_22	50	40	40	264359	265605	**266189**	265989	263609.80	265291.10	**265689.15**	265603.50
MDG-a_23	50	40	40	264006	265567	**266029**	265798	263688.35	265294.40	**265647.90**	265602.55
MDG-a_24	50	40	40	264027	265564	**266125**	265957	263563.75	265257.25	265702.65	**265721.70**
MDG-a_25	50	40	40	264208	265631	**266102**	266013	263703.25	265357.50	**265754.70**	265717.20
MDG-a_26	50	40	40	264421	265440	**266376**	265973	263736.05	265239.70	**265784.45**	265681.10
MDG-a_27	50	40	40	264131	265474	265864	**265972**	263485.50	265105.60	265552.60	**265631.45**
MDG-a_28	50	40	40	264168	265535	**266172**	265978	263723.30	265285.60	265691.20	**265703.45**
MDG-a_29	50	40	40	264096	265659	**266180**	265986	263775.10	265346.10	**265790.60**	265674.05
MDG-a_30	50	40	40	264329	265538	265915	**265942**	263689.50	265310.10	265602.75	**265654.45**
MDG-a_31	50	40	40	264274	265779	**266356**	266186	263755.00	265459.95	265836.80	**265844.95**
MDG-a_32	50	40	40	264189	265448	**266206**	266033	263694.70	265243.60	**265727.55**	265714.55
MDG-a_33	50	40	40	264125	265590	**266276**	265840	263808.05	265331.10	**265691.30**	265644.00
MDG-a_34	50	40	40	264305	265595	**266002**	265972	263702.55	265398.15	**265777.20**	265735.65
MDG-a_35	50	40	40	264171	265543	**266053**	265798	263701.70	265254.85	**265697.80**	265563.85
MDG-a_36	50	40	40	264335	265573	**266058**	266006	263700.80	265305.65	265639.35	**265653.15**
MDG-a_37	50	40	40	264041	265614	**266044**	265842	263686.85	265271.60	**265748.25**	265708.65
MDG-a_38	50	40	40	264039	265627	**266089**	265890	263613.75	265301.80	**265720.40**	265673.20
MDG-a_39	50	40	40	264128	265542	**266030**	265990	263631.30	265268.75	**265694.00**	265674.30
MDG-a_40	50	40	40	264225	265703	**266205**	266091	263812.05	265433.20	**265890.05**	265846.55
Avg.				264193.75	265584.85	**266113.90**	265966.85	263681.64	265304.90	**265719.52**	265688.35
#Best				0	0	**17**	3	0	0	**14**	6
p-value				0.000082	0.000082	1		0.000082	0.000082	1	

Table 7.10: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large EGS instances with $n = 2000$, $m = 100$.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	100	20	20	144528	145112	145526	**146161**	143874.95	144875.15	145197.45	**145975.35**
MDG-a_22	100	20	20	144258	145092	145627	**146144**	143799.95	144840.70	145214.45	**145962.30**
MDG-a_23	100	20	20	144297	145120	145444	**146168**	143873.60	144853.40	145098.75	**145935.40**
MDG-a_24	100	20	20	144169	145053	145574	**146234**	143935.05	144890.85	145215.90	**145926.35**
MDG-a_25	100	20	20	144400	145157	145476	**146152**	143958.70	144912.55	145205.05	**145964.40**
MDG-a_26	100	20	20	144512	145150	145531	**146293**	144016.45	144971.25	145191.95	**145993.80**
MDG-a_27	100	20	20	144322	144974	145408	**146051**	143849.60	144807.75	145149.50	**145862.70**
MDG-a_28	100	20	20	144360	145078	145501	**146123**	143938.80	144870.00	145200.65	**145953.50**
MDG-a_29	100	20	20	144230	145256	145528	**146157**	143940.20	144893.90	145227.75	**145964.80**
MDG-a_30	100	20	20	144042	145218	145449	**146265**	143838.40	144873.25	145206.30	**145976.80**
MDG-a_31	100	20	20	144278	145092	145672	**146114**	144021.25	144931.85	145277.55	**145965.65**
MDG-a_32	100	20	20	144521	145160	145472	**146182**	143909.65	144887.05	145162.10	**145988.40**
MDG-a_33	100	20	20	144283	145244	145650	**146219**	143967.40	144904.75	145232.10	**145970.60**
MDG-a_34	100	20	20	144432	145196	145482	**146253**	144013.30	144878.00	145195.55	**145981.75**
MDG-a_35	100	20	20	144197	145154	145473	**146116**	143871.35	144846.15	145172.45	**145923.70**
MDG-a_36	100	20	20	144436	145274	145641	**146323**	143999.25	144881.45	145185.00	**145916.75**
MDG-a_37	100	20	20	144141	145194	145524	**146130**	143916.30	144931.60	145213.10	**145956.90**
MDG-a_38	100	20	20	144416	145243	145607	**146107**	144036.55	144933.25	145276.55	**145968.45**
MDG-a_39	100	20	20	144321	145037	145706	**146088**	143925.85	144811.90	145191.25	**145937.40**
MDG-a_40	100	20	20	144266	145168	145565	**146329**	143993.50	144922.65	145242.65	**146005.60**
Avg.				144320.45	145148.60	145542.80	**146180.45**	143934.01	144885.87	145202.80	**145956.53**
#Best				0	0	0	**20**	0	0	0	**20**
p-value				0.000082	0.00007	0.000076		0.000082	0.000082	0.000082	

Table 7.11: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large EGS instances with n = 2000, m = 200.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-a_21	200	10	10	77354	76941	78341	**78370**	77040.75	76824.40	78172.60	**78260.50**
MDG-a_22	200	10	10	77333	76900	78338	**78349**	77103.75	76817.75	78150.85	**78254.80**
MDG-a_23	200	10	10	77227	76903	78200	**78352**	77067.70	76805.95	78081.85	**78244.45**
MDG-a_24	200	10	10	77517	77023	78349	**78403**	77034.65	76823.40	78127.75	**78246.85**
MDG-a_25	200	10	10	77376	76919	78237	**78344**	77127.85	76849.90	78110.25	**78247.45**
MDG-a_26	200	10	10	77482	76935	78295	**78374**	77143.05	76833.20	78147.55	**78273.00**
MDG-a_27	200	10	10	77313	76863	**78347**	78340	77038.00	76764.40	78124.55	**78182.60**
MDG-a_28	200	10	10	77334	76949	78303	**78393**	77075.20	76831.35	78187.00	**78241.50**
MDG-a_29	200	10	10	77265	76939	78277	**78414**	77087.80	76839.75	78164.50	**78265.80**
MDG-a_30	200	10	10	77470	76947	78297	**78405**	77159.15	76835.40	78146.95	**78262.20**
MDG-a_31	200	10	10	77351	76999	78362	**78420**	77162.85	76850.30	78161.55	**78272.25**
MDG-a_32	200	10	10	77296	76934	78227	**78312**	77048.60	76818.25	78132.55	**78240.80**
MDG-a_33	200	10	10	77308	76931	**78342**	**78342**	77073.95	76823.80	78193.15	**78246.90**
MDG-a_34	200	10	10	77317	76972	**78366**	78358	77136.70	76828.15	78172.60	**78254.65**
MDG-a_35	200	10	10	77380	76937	78286	**78424**	77063.50	76832.10	78149.60	**78255.85**
MDG-a_36	200	10	10	77301	76937	78304	**78397**	77098.10	76814.90	78180.85	**78260.05**
MDG-a_37	200	10	10	77289	76919	78332	**78592**	77078.75	76844.35	78173.45	**78279.95**
MDG-a_38	200	10	10	77428	76908	78355	**78399**	77124.25	76810.05	78166.45	**78238.30**
MDG-a_39	200	10	10	77337	76925	**78355**	78330	77082.40	76816.65	78150.80	**78245.10**
MDG-a_40	200	10	10	77385	76963	78356	**78381**	77133.00	76860.00	78180.95	**78289.65**
Avg.				77353.15	76937.20	78313.45	**78384.95**	77094.00	76826.20	78153.79	**78253.13**
#Best				0	0	4	**17**	0	0	0	**20**
p-value				0.00006	0.000076	0.000372		0.000082	0.000082	0.000082	

Table 7.12: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large DGS instances with n = 3000, m = 50,L_g = 48, and U_g = 72.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-c_1	50	48	72	57945495	58183439	**58353327**	58241123	57893672.20	58157522.75	**58322559.10**	58202064.25
MDG-c_2	50	48	72	57950256	58177131	**58357064**	58239003	57889624.55	58145198.80	**58299240.25**	58184308.30
MDG-c_3	50	48	72	57939891	58170977	**58341358**	58213359	57881918.30	58123100.95	**58295779.25**	58172416.45
MDG-c_4	50	48	72	57942801	58201217	**58336151**	58222418	57883281.15	58144707.25	**58284423.15**	58174668.30
MDG-c_5	50	48	72	57927489	58148831	**58320349**	58176735	57861217.45	58116749.55	**58274259.15**	58156640.65
MDG-c_6	50	48	72	57916710	58144669	**58312474**	58201738	57862922.65	58112448.75	**58277656.70**	58155940.30
MDG-c_7	50	48	72	57939818	58163126	**58325629**	58213211	57863448.25	58117706.90	**58285170.90**	58166420.95
MDG-c_8	50	48	72	57934738	58156017	**58319208**	58199212	57861358.45	58126408.10	**58280932.95**	58161427.05
MDG-c_9	50	48	72	57944061	58153700	**58297084**	58183364	57861925.30	58103468.35	**58262931.10**	58141485.70
MDG-c_10	50	48	72	57934502	58150988	**58325424**	58216621	57860647.25	58122795.25	**58282754.25**	58161295.85
MDG-c_11	50	48	72	57918627	58157713	**58323946**	58186546	57862895.75	58109035.30	**58284788.55**	58155667.35
MDG-c_12	50	48	72	57904829	58138299	**58328166**	58212761	57861797.20	58113264.25	**58286736.60**	58158936.45
MDG-c_13	50	48	72	57948819	58169372	**58336456**	58216847	57877734.70	58122095.95	**58281207.75**	58178609.75
MDG-c_14	50	48	72	57981820	58162079	**58382020**	58207649	57865649.40	58127767.30	**58288165.40**	58167793.25
MDG-c_15	50	48	72	57931740	58186521	**58350512**	58227209	57870836.15	58118934.05	**58304797.25**	58169282.40
MDG-c_16	50	48	72	58010385	58177602	**58337723**	58218667	57889027.40	58130542.25	**58295022.90**	58179813.45
MDG-c_17	50	48	72	57926117	58144993	**58328280**	58212777	57875597.75	58110074.60	**58269424.55**	58153727.9
MDG-c_18	50	48	72	57900242	58140926	**58276589**	58163891	57850459.15	58104895.35	**58251973.45**	58138017.35
MDG-c_19	50	48	72	57940090	58167833	**58327771**	58225444	57869054.10	58131811.00	**58287729.90**	58179391.30
MDG-c_20	50	48	72	57944047	58171452	**58341928**	58241035	57873653.75	58131518.15	**58292051.90**	58174394.25
Avg.				57939123.85	58163344.25	**58331072.95**	58210980.50	57870836.05	58123502.24	**58285380.25**	58166615.06
#Best				0	0	**20**	0	0	0	**20**	0
p-value				0.000082	0.000082	1		0.000082	0.000082	1	

Table 7.13: Comparison of the TPSDP algorithm with three best performing algorithms on 20 large EGS instances with n = 3000, m = 50.

Instance				f_{best}				f_{avg}			
Graph	m	L_g	U_g	**ITS**	**IMS**	**NDHA**	**TPSDP**	**ITS**	**IMS**	**NDHA**	**TPSDP**
MDG-c_1	50	60	60	56022225	56293075	**56420443**	56342168	55969259.05	56264739.40	**56373021.45**	56307221.50
MDG-c_2	50	60	60	56031948	56314401	**56397784**	56374669	55961075.80	56262042.25	**56351399.05**	56309399.65
MDG-c_3	50	60	60	56001751	56283795	**56410198**	56320245	55910535.55	56248929.20	**56346013.80**	56281097.95
MDG-c_4	50	60	60	56005257	56287274	**56375606**	56315572	55922664.65	56247930.85	**56346801.10**	56290599.10
MDG-c_5	50	60	60	55990760	56263713	**56380138**	56327200	55919956.90	56227945.35	**56329247.90**	56272512.20
MDG-c_6	50	60	60	55962420	56264269	**56353510**	56293487	55909168.85	56229132.45	**56323859.90**	56270551.95
MDG-c_7	50	60	60	55975570	56264667	**56381298**	56310811	55913141.65	56238977.35	**56332139.55**	56280772.70
MDG-c_8	50	60	60	56006984	56272094	**56370974**	56343156	55910890.35	56235146.75	**56328351.10**	56276918.25
MDG-c_9	50	60	60	56002073	56257881	**56350261**	56298798	55905842.80	56216309.00	**56315049.55**	56260624.60
MDG-c_10	50	60	60	55964543	56261337	**56369187**	56325952	55909596.65	56226019.35	**56316037.20**	56279829.00
MDG-c_11	50	60	60	56010877	56262799	**56374962**	56333818	55930405.85	56237929.25	**56322488.25**	56274470.60
MDG-c_12	50	60	60	55994413	56281715	**56388108**	56305414	55925052.45	56243671.10	**56331949.60**	56273345.20
MDG-c_13	50	60	60	56007494	56279255	**56393146**	56318406	55918195.15	56247326.40	**56345243.75**	56288795.70
MDG-c_14	50	60	60	55997667	56291993	**56411582**	56336228	55946080.20	56246983.75	**56352125.65**	56290913.00
MDG-c_15	50	60	60	55992707	56276430	**56365603**	56335516	55926300.70	56236820.45	**56332909.95**	56295684.45
MDG-c_16	50	60	60	56011405	56276199	**56400249**	56332417	55924195.65	56245849.10	**56346501.65**	56292399.40
MDG-c_17	50	60	60	55994715	56259679	**56374799**	56289065	55918999.90	56231608.30	**56315744.65**	56259674.60
MDG-c_18	50	60	60	55964888	56242334	**56352503**	56298885	55895009.50	56214364.85	**56311484.50**	56249494.20
MDG-c_19	50	60	60	56000882	56292680	**56380686**	56312207	55921372.20	56249335.65	**56353801.30**	56286150.70
MDG-c_20	50	60	60	56046425	56264742	**56377541**	56321944	55947549.30	56230349.05	**56348220.55**	56294755.20
Avg.				55999250.20	56274516.60	**56381428.90**	56321797.90	55924264.66	56239070.49	**56336119.52**	56281760.50
#Best				0	0	**20**	0	0	0	**20**	0
p-value				0.000082	0.000082	1		0.000082	0.000082	1	

Bibliography

[1] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[2] A. Schrijver, "On the history of combinatorial optimization (till 1960)," *Handbooks in Operations Research and Management Science*, vol. 12, pp. 1–68, 2005.

[3] M. Jünger, G. Reinelt, and G. Rinaldi, "The traveling salesman problem," *Handbooks in Operations Research and Management Science*, vol. 7, pp. 225–330, 1995.

[4] R. A. Walker and S. Chaudhuri, "Introduction to the scheduling problem," *IEEE Design & Test of Computers*, vol. 12, no. 2, pp. 60–69, 1995.

[5] A. J. Kleywegt and J. D. Papastavrou, "The dynamic and stochastic knapsack problem," *Operations Research*, vol. 46, no. 1, pp. 17–35, 1998.

[6] A. Lodi, S. Martello, and M. Monaci, "Two-dimensional packing problems: A survey," *European Journal of Operational Research*, vol. 141, no. 2, pp. 241–252, 2002.

[7] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of combinatorial optimization*. Springer, 1999, pp. 1–74.

[8] S. Z. Selim and K. Alsultan, "A simulated annealing algorithm for the clustering problem," *Pattern Recognition*, vol. 24, no. 10, pp. 1003–1008, 1991.

[9] T. R. Jensen and B. Toft, *Graph coloring problems*. John Wiley & Sons, 2011.

[10] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

[11] H. Marchand, A. Martin, R. Weismantel, and L. Wolsey, “Cutting planes in integer and mixed integer programming,” *Discrete Applied Mathematics*, vol. 123, no. 1-3, pp. 397–446, 2002.

[12] R. Bellman, “Dynamic programming,” *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[13] S. A. Cook, “The complexity of theorem-proving procedures,” in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.

[14] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*. Cambridge university press, 2011.

[15] A. Vince, “A framework for the greedy algorithm,” *Discrete Applied Mathematics*, vol. 121, no. 1-3, pp. 247–260, 2002.

[16] P. Biswas, T.-C. Liang, K.-C. Toh, Y. Ye, and T.-C. Wang, “Semidefinite programming approaches for sensor network localization with noisy distance measurements,” *IEEE Transactions on Automation Science and Engineering*, vol. 3, no. 4, pp. 360–371, 2006.

[17] S. I. Gass, *Linear programming: methods and applications*. Courier Corporation, 2003.

[18] S. H. Zanakis, J. R. Evans, and A. A. Vazacopoulos, “Heuristic methods and applications: a categorized survey,” *European Journal of Operational Research*, vol. 43, no. 1, pp. 88–110, 1989.

[19] N. Kokash, “An introduction to heuristic algorithms,” *Department of Informatics and Telecommunications*, pp. 1–8, 2005.

[20] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[21] I. H. Osman and G. Laporte, “Metaheuristics: A bibliography,” 1996.

[22] M. Dorigo, “Optimization, learning and natural algorithms,” *Ph. D. Thesis, Politecnico di Milano*, 1992.

[23] M. Dorigo and T. Stützle, “Ant colony optimization: overview and recent advances,” *Handbook of Metaheuristics*, pp. 311–351, 2019.

[24] R. Poli, J. Kennedy, and T. Blackwell, “Particle swarm optimization,” *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[25] D. Wang, D. Tan, and L. Liu, “Particle swarm optimization algorithm: an overview,” *Soft Computing*, vol. 22, no. 2, pp. 387–408, 2018.

[26] D. Karaboga *et al.*, “An idea based on honey bee swarm for numerical optimization,” Technical report-tr06, Erciyes university, engineering faculty, computer . . . , Tech. Rep., 2005.

[27] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[28] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[29] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[30] H. R. Lourenço, O. C. Martin, and T. Stützle, “Iterated local search,” in *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.

[31] Z.-H. Fu and J.-K. Hao, “A three-phase search approach for the quadratic minimum spanning tree problem,” *Engineering Applications of Artificial Intelligence*, vol. 46, pp. 113–130, 2015.

[32] X. Lai and J.-K. Hao, “Iterated maxima search for the maximally diverse grouping problem,” *European Journal of Operational Research*, vol. 254, no. 3, pp. 780–800, 2016.

[33] K. Singh and S. Sundar, “A new hybrid genetic algorithm for the maximally diverse grouping problem,” *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 10, pp. 2921–2940, 2019.

[34] R. Weitz and S. Lakshminarayanan, “An empirical comparison of heuristic methods for creating maximally diverse groups,” *Journal of the Operational Research Society*, vol. 49, no. 6, pp. 635–646, 1998.

[35] J. Desrosiers, N. Mladenović, and D. Villeneuve, “Design of balanced mba student teams,” *Journal of the Operational Research Society*, vol. 56, no. 1, pp. 60–66, 2005.

[36] D. Krass and A. Ovchinnikov, “Constrained group balancing: Why does it work,” *European Journal of Operational Research*, vol. 206, no. 1, pp. 144–154, 2010.

[37] H. K. Yeoh and M. I. Mohamad Nor, “An algorithm to form balanced and diverse groups of students,” *Computer Applications in Engineering Education*, vol. 19, no. 3, pp. 582–590, 2011.

[38] J. Johnes, “Operational research in education,” *European Journal of Operational Research*, vol. 243, no. 3, pp. 683–696, 2015.

[39] Y. Chen, Z.-P. Fan, J. Ma, and S. Zeng, “A hybrid grouping genetic algorithm for reviewer group construction problem,” *Expert Systems with Applications*, vol. 38, no. 3, pp. 2401–2411, 2011.

[40] R. Weitz and S. Lakshminarayanan, “An empirical comparison of heuristic and graph theoretic methods for creating maximally diverse groups, vlsi design, and exam scheduling,” *Omega*, vol. 25, no. 4, pp. 473–482, 1997.

[41] J. Bhadury, E. J. Mighty, and H. Damar, “Maximizing workforce diversity in project teams: A network flow approach,” *Omega*, vol. 28, no. 2, pp. 143–153, 2000.

[42] T. A. Feo and M. Khellaf, “A class of bounded approximation algorithms for graph partitioning,” *Networks*, vol. 20, no. 2, pp. 181–195, 1990.

[43] T. Arani and V. Lotfi, "A three phased approach to final exam scheduling," *IIE Transactions*, vol. 21, no. 1, pp. 86–96, 1989.

[44] R. R. Weitz and M. T. Jelassi, "Assigning students to groups: a multi-criteria decision support system approach," *Decision Sciences*, vol. 23, no. 3, pp. 746–757, 1992.

[45] M. Gallego, M. Laguna, R. Martí, and A. Duarte, "Tabu search with strategic oscillation for the maximally diverse grouping problem," *Journal of the Operational Research Society*, vol. 64, no. 5, pp. 724–734, 2013.

[46] G. Palubeckis, E. Karčiauskas, and A. Riškus, "Comparative performance of three metaheuristic approaches for the maximally diverse grouping problem," *Information Technology and Control*, vol. 40, no. 4, pp. 277–285, 2011.

[47] D. Uroŝevic, "Variable neighborhood search for maximum diverse grouping problem," *Yugoslav Journal of Operations Research*, vol. 24, no. 1, pp. 21–33, 2014.

[48] J. Brimberg, N. Mladenović, and D. Urošević, "Solving the maximally diverse grouping problem by skewed general variable neighborhood search," *Information Sciences*, vol. 295, pp. 650–675, 2015.

[49] G. Palubeckis, A. Ostreika, and D. Rubliauskas, "Maximally diverse grouping: an iterated tabu search approach," *Journal of the Operational Research Society*, vol. 66, no. 4, pp. 579–592, 2015.

[50] X. Lai, J.-K. Hao, Z.-H. Fu, and D. Yue, "Neighborhood decomposition based variable neighborhood search and tabu search for maximally diverse grouping," *European Journal of Operational Research*, vol. 289, no. 3, pp. 1067–1086, 2021.

[51] Z.-P. Fan, Y. Chen, J. Ma, and S. Zeng, "Erratum: A hybrid genetic algorithmic approach to the maximally diverse grouping problem," *Journal of the Operational Research Society*, vol. 62, no. 7, pp. 1423–1430, 2011.

[52] F. J. Rodriguez, M. Lozano, C. García-Martínez, and J. D. González-Barrera, "An artificial bee colony algorithm for the maximally diverse grouping problem," *Information Sciences*, vol. 230, pp. 183–196, 2013.

[53] C. R. Reeves, *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., 1993.

[54] F. W. Glover and G. A. Kochenberger, *Handbook of metaheuristics*. Springer Science & Business Media, 2006, vol. 57.

[55] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[56] J. Brimberg, "A variable neighborhood algorithm for solving the continuous location-allocation problem," *Studies in Locational Analysis*, vol. 10, pp. 1–12, 1996.

[57] W. C. Davidon, "Variable metric method for minimization," *SIAM Journal on Optimization*, vol. 1, no. 1, pp. 1–17, 1991.

[58] R. Fletcher and M. J. Powell, "A rapidly convergent descent method for minimization," *The Computer Journal*, vol. 6, no. 2, pp. 163–168, 1963.

[59] N. Mladenovic, "A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization," in *papers presented at Optimization Days*, vol. 112, 1995.

[60] C. Audet, V. Béchard, and S. L. Digabel, "Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search," *Journal of Global Optimization*, vol. 41, no. 2, pp. 299–318, 2008.

[61] C. Audet, J. Brimberg, P. Hansen, S. L. Digabel, and N. Mladenović, "Pooling problem: Alternate formulations and solution methods," *Management Science*, vol. 50, no. 6, pp. 761–776, 2004.

[62] G. Caporossi, D. Alamargot, and D. Chesnet, "Using the computer to study the dynamics of the handwriting processes," in *International Conference on Discovery Science*. Springer, 2004, pp. 242–254.

[63] F. Carrabs, J.-F. Cordeau, and G. Laporte, “Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading,” *Informs Journal on Computing*, vol. 19, no. 4, pp. 618–632, 2007.

[64] E. Carrizosa, B. Martín-Barragán, F. Plastria, and D. Romero Morales, “On the selection of the globally optimal prototype subset for nearest-neighbor classification,” *Informs Journal on Computing*, vol. 19, no. 3, pp. 470–479, 2007.

[65] P. Hansen and N. Mladenović, “Developments of variable neighborhood search,” in *Essays and surveys in metaheuristics*. Springer, 2002, pp. 415–439.

[66] M. A. Lejeune, “A variable neighborhood decomposition search method for supply chain management planning problems,” *European Journal of Operational Research*, vol. 175, no. 2, pp. 959–976, 2006.

[67] C. Zhang, Z. Lin, and Z. Lin, “Variable neighborhood search with permutation distance for qap,” in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2005, pp. 81–88.

[68] M. E. Aydin and T. C. Fogarty, “A distributed evolutionary simulated annealing algorithm for combinatorial optimisation problems,” *Journal of Heuristics*, vol. 10, no. 3, pp. 269–292, 2004.

[69] D. T. Connolly, “An improved annealing scheme for the qap,” *European Journal of Operational Research*, vol. 46, no. 1, pp. 93–100, 1990.

[70] P. J. Van Laarhoven, E. H. Aarts, and J. K. Lenstra, “Job shop scheduling by simulated annealing,” *Operations Research*, vol. 40, no. 1, pp. 113–125, 1992.

[71] R. Battiti, “Reactive search: Toward self-tuning heuristics,” *Modern Heuristic Search Methods*, vol. 4, pp. 61–83, 1996.

[72] R. Battiti and G. Tecchiolli, “The reactive tabu search,” *ORSA Journal on Computing*, vol. 6, no. 2, pp. 126–140, 1994.

[73] R. Battiti and M. Protasi, “Reactive search, a history-sensitive heuristic for max-sat,” *Journal of Experimental Algorithmics (JEA)*, vol. 2, pp. 2–es, 1997.

[74] M. Dell’Amico, A. Lodi, and F. Maffioli, “Solution of the cumulative assignment problem with a well-structured tabu search method,” *Journal of Heuristics*, vol. 5, no. 2, pp. 123–143, 1999.

[75] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[76] R. C. Eberhart, Y. Shi, and J. Kennedy, *Swarm Intelligence (Morgan Kaufmann series in evolutionary computation)*. Morgan Kaufmann Publishers, 2001.

[77] R. V. Kulkarni and G. K. Venayagamoorthy, “Particle swarm optimization in wireless-sensor networks: A brief survey,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 2, pp. 262–267, 2010.

[78] M. R. AlRashidi and M. E. El-Hawary, “A survey of particle swarm optimization applications in electric power systems,” *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 913–918, 2008.

[79] S. Alam, G. Dobbie, Y. S. Koh, P. Riddle, and S. U. Rehman, “Research on particle swarm optimization based clustering: a systematic review of literature and techniques,” *Swarm and Evolutionary Computation*, vol. 17, pp. 1–13, 2014.

[80] K. Jebari and M. Madiafi, “Selection methods for genetic algorithms,” *International Journal of Emerging Sciences*, vol. 3, no. 4, pp. 333–344, 2013.

[81] D. Datta, A. R. Amaral, and J. R. Figueira, “Single row facility layout problem using a permutation-based genetic algorithm,” *European Journal of Operational Research*, vol. 213, no. 2, pp. 388–394, 2011.

[82] C. K. H. Lee, “A review of applications of genetic algorithms in operations management,” *Engineering Applications of Artificial Intelligence*, vol. 76, pp. 1–12, 2018.

[83] A. Hiassat, A. Diabat, and I. Rahwan, “A genetic algorithm approach for location-inventory-routing problem with perishable products,” *Journal of Manufacturing Systems*, vol. 42, pp. 93–103, 2017.

[84] G. Sermpinis, C. Stasinakis, K. Theofilatos, and A. Karathanasopoulos, “Modeling, forecasting and trading the eur exchange rates with hybrid rolling genetic algorithms—support vector regression forecast combinations,” *European Journal of Operational Research*, vol. 247, no. 3, pp. 831–846, 2015.

[85] S. Jiang, K.-S. Chin, L. Wang, G. Qu, and K. L. Tsui, “Modified genetic algorithm-based feature selection combined with pre-trained deep neural network for demand forecasting in outpatient department,” *Expert Systems with Applications*, vol. 82, pp. 216–230, 2017.

[86] A. Diabat and R. Deskoores, “A hybrid genetic algorithm based heuristic for an integrated supply chain problem,” *Journal of Manufacturing Systems*, vol. 38, pp. 172–180, 2016.

[87] J. Shi, Z. Liu, L. Tang, and J. Xiong, “Multi-objective optimization for a closed-loop network design problem using an improved genetic algorithm,” *Applied Mathematical Modelling*, vol. 45, pp. 14–30, 2017.

[88] C. M. Bishop, “Neural networks and their applications,” *Review of Scientific Instruments*, vol. 65, no. 6, pp. 1803–1832, 1994.

[89] J. J. Hopfield and D. W. Tank, ““neural” computation of decisions in optimization problems,” *Biological Cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[90] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” *Advances in Neural Information Processing Systems*, vol. 28, 2015.

Acknowledgements

First of all, I would like to give my heartfelt thanks to all the people who have ever helped me during the writing of this thesis and my entire PhD career.

I would like to express my sincere gratitude to my supervisor, Prof. Tang and Associate Prof. Gao of Toyama University, for guiding me with their patient instruction and offering constructive opinions on my thesis. Without them, I can't study abroad and enter the complex and magical computer world. I consider it a great privilege to have got the benefit of their professional guidance.

I would like to extend my special thanks to my classmate Zonghui Cai. He is not only a gentle partner, but also a teacher in my study, a comrade in arms in life, and a man who knows how to respect others. In my studies, he never tires of teaching me obscure knowledge, thinking with me and helping me solve problems. In life, it is also because of his attentive company and meticulous care that I can spend this period of study abroad safely and happily.

My deepest thanks would go to my beloved family, especially my favorite mother, Ms. Lu, for their loving considerations and great confidence in me. Through these years, they have been supporting and encouraging me financially and spiritually, and they will always be my most solid backing. I also own my appreciation to my best friend Ying Wang and college roommates Qin Luo, Huimin Ding, and Xue Peng. During the five years of studying in Japan, I can't forget their warm company. They are the best listeners and enlighteners. Whenever I fall into anxiety and depression, they give me selfless companionship and firm strength. They let me know myself, accept myself, persist in myself, and do not lose my direction in the long journey of life. They are the treasure in my life.

At last but not least, I also want to thanks myself. I want to say to myself, cherish

yourself, always be proud of yourself, you are the best.