A LP Relaxation based Matheuristic for Multi-objective Integer Programming

Duleabom An[1], Sophie N. Parragh[1][a], Markus Sinnl[1][b] and Fabien Tricoire[2][c]
[1]*Institute of Production and Logistics Management, Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria*
[2]*Institute for Transport and Logistics Management, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria*

Keywords: Three-objective Binary Integer Programming, Matheuristics, Path Relinking, Multi-objective Knapsack Problem.

Abstract: Motivated by the success of matheuristics in the single-objective domain, we propose a very simple linear programming-based matheuristic for three-objective binary integer programming. To tackle the problem, we obtain lower bound sets by means of the vector linear programming solver *Bensolve*. Then, simple heuristic approaches, such as rounding and path relinking, are applied to this lower bound set to obtain high-quality approximations of the optimal set of trade-off solutions. The proposed algorithm is compared to a recently suggested algorithm which is, to the best of our knowledge, the only existing matheuristic method for three-objective integer programming. Computational experiments show that our method produces a better approximation of the true *Pareto front* using significantly less time than the benchmark method on standard benchmark instances for the three-objective knapsack problem.

1 INTRODUCTION

Many real-world optimisation problems involve multiple conflicting objectives, concerning, e.g., costs, environmental impact or service level, and can be formulated as integer linear programs. Kolli and Evans (1999), for example, deal with facility location problems for a franchise company. When a new franchise is launched, the franchisee and the franchisor have conflicting objectives. To find the optimal location for the new store, the model maximises the number of customers while minimising conflict among existing franchises. Sawik et al. (2016) investigate vehicle routing problems occurring in the logistics of the food industry. The introduced model minimises both the total travel distance and CO_2 emissions. A home care routing and scheduling problem is investigated by Braekers et al. (2016). Given a group of nurses and a number of care tasks at the patients' home locations, the model assigns tasks to each nurse while minimising operating cost and client inconvenience with respect to the timing of the visit and the nurse performing the task. Kovacs et al. (2015) study a multi-objective consistent vehicle routing problem. To provide consistent service in the routing industry, they maximise driver consistency and arrival time consistency while minimising total routing costs.

The main goal in multi-objective (MO) optimisation is to generate the set of optimal trade-off solutions, known as *efficient* (or *Pareto optimal*) solutions. Our study focuses on binary integer programming (IP) problems with three objectives.

Motivated by the success of matheuristics in the single objective domain, we present a very simple matheuristic for three-objective integer programming. To the best of our knowledge, only one other matheuristic for three-objective integer programming has been developed so far (Pal and Charkhgard, 2019b). The algorithm we propose relies on a lower bound set which is obtained from the linear programming (LP) relaxation, using the vector linear programming solver *Bensolve* (Löhne and Weißing, 2017). The lower bound set is defined by its extreme points (and edges). Starting from those solutions which give rise to the extreme points, we apply *rounding* in combination with *path relinking* to obtain high-quality approximations of the true *Pareto fron-*

[a] https://orcid.org/0000-0002-7428-9770
[b] https://orcid.org/0000-0003-1439-8702
[c] https://orcid.org/0000-0002-3700-5134

An, D., Parragh, S., Sinnl, M. and Tricoire, F.
A LP Relaxation based Matheuristic for Multi-objective Integer Programming.
DOI: 10.5220/0010347000880098
In *Proceedings of the 10th International Conference on Operations Research and Enterprise Systems (ICORES 2021)*, pages 88-98
ISBN: 978-989-758-485-5

tier (*PF*).
The contribution of this paper is twofold:

- We show that, in the case of the three-objective assignment problem, since the extreme solutions which *Bensolve* produces are integer, it already provides high-quality approximations of the true Pareto frontier, even without additional ingredients.
- We propose the first LP relaxation-based matheuristic algorithm for three-objective integer programming, combining *Bensolve* with *rounding* and *path relinking (PR)*

The remainder of the paper is structured as follows. Section 2 briefly reviews existing work in multi-objective integer programming (MOIP) matheuristics. Section 3 provides basic concepts and background for MOIP. The proposed algorithms are described in Section 4 and empirically evaluated on the multi-objective assignment problem (MOAP) and multi-objective knapsack problem (MOKP) in Section 5. Finally, the paper concludes with a summary and suggestions for future work in Section 6.

2 RELATED WORK

Over the past years a number of generic exact methods for solving MOIP have been proposed (see, e.g., Mavrotas and Florios (2013), Zhang and Reimann (2014), Kirlik and Sayın (2014), Boland et al. (2017)). In spite of their popularity in the single objective domain, only comparably few contributions on matheuristc methods exist.

A heuristic and a matheuristic approach for bi-objective mixed binary integer linear programming are proposed by Soylu (2015). The methods are variants of variable neighbourhood search and local branching. Both algorithms collect segments of the *PF* during the search and combine them at the end. To deal with bi-objective binary IP, Leitner et al. (2016) suggest an exact method and a matheuristic framework. The general idea is inspired by the fact that efficient solutions in the same area often share common features. In each phase, their heuristic obtains feasible solutions by fixing a large number of variables and reducing the associated feasible region, respectively. Based on the *feasibility pump (FP)* idea introduced by Fischetti et al. (2005), Pal and Charkhgard (2019a) suggested a FP based heuristic for bi-objective IP and extended the method to higher dimensions in (Pal and Charkhgard, 2019b). The proposed method works in two stages. Both stages employ a FP based heuristic. In the first stage, where the purpose is to generate well spread solutions, fractional solutions are computed by means of a weighted sum method. The purpose of the second stage is to generate additional solutions. It relies on local branching in combination with the developed FP based heuristic. To the best of our knowledge, this is the only matheuristic method that deals with MOIP with more than two objectives. Therefore, we use it as a benchmark and refer to it as FPBH.

3 BASIC CONCEPTS AND BACKGROUND

The problem we consider is a MOIP, with binary integer decision variables and three objectives. In the following, we state the MOIP in its general form. We present a unified view using minimisation objectives.

$$y = min\{C(x) : Ax \geq b, x \in \{0,1\}^n\}, \qquad \text{(MOIP)}$$

where x_j, $j = 1,2,\ldots,n$, is the vector of decision variables and $X := \{Ax \geq b, x \in \{0,1\}^n\}$ is the feasible set. C is a $p \times n$ objective function matrix where c^k, $(k = 1,2,\ldots,p)$, is the k^{th} row of C. In our case, $p = 3$. Further, y is a set of points in the objective space (criterion space), each of which corresponds to at least one solution vector $x \in X$. A is an $m \times n$ constraint matrix and b is the vector of right-hand-side values for these constraints.

3.1 Pareto Dominance

In MO optimisation, the quality of a solution is determined by *Pareto* dominance. Suppose there are two solutions x and x' of Problem (MOIP). Then, x *dominates* x' if and only if $c^k(x) \leq c^k(x')$ for all $k \in \{1,\ldots,p\}$ and $c^k(x) < c^k(x')$ for at least one k. If there does not exist any x' that dominates x, then x is *Pareto optimal*. If x^* is a *Pareto optimal* (efficient) solution, then y^* is a non-dominated point. The set of all non-dominated points is called the *Pareto front*.

3.2 Weighted Sum Method and Supported Solutions

The weighted sum method is a commonly used approach in solving MO optimisation problems. It combines multiple objectives into one as follows:

$$F(x) = w_1 f_1(x) + w_2 f_2(x) + \cdots + w_p f_p(x)$$

$$\sum_{i=1}^{p} w_i = 1, \;\; w_i \in (0,1),$$

where w_i is the weight of objective function i. If a solution x^* can be found by the weighted sum method

(i.e. optimising a convex combination of all objective functions), it is called a *supported* efficient solution. Otherwise, it is a *non-supported* efficient solution.

3.3 LP Relaxation and Bound Set

The notion of bound set was introduced by Ehrgott and Gandibleux (2007). In the context of Problem (MOIP), a lower bound (LB) set and an upper bound (UB) set provide information about the variable range that efficient solutions of the MOIP can attain. One common way to obtain a LB set for a minimisation problem is to solve the LP relaxation of the original problem. For constructing the LP relaxation of Problem (MOIP), the integrality conditions are removed, i.e. $0 \leq x_j \leq 1,\ j = 1,\ldots,n$ and *Bensolve* can then be used to compute the LB set. In our heuristics, we will use the solutions associated with the LB set. In a slight abuse of notation, we will refer to these solutions as LB set.

3.4 Benchmark Problems

The multi-objective assignment problem (MOAP) and the multi-objective knapsack problem (MOKP) are commonly used benchmark problems in MO. We use standard benchmark instances of these two problems for our computational experiments.

3.4.1 Multi-objective Assignment Problem

The well-known assignment problem is a type of transportation problem: a certain number of tasks $l \in \{1,\ldots,n\}$ and agents $r \in \{1,\ldots,n\}$ are given. The decision variable x_{rl} denotes whether task l is assigned to the agent r (x_{rl}=1) or not (x_{rl}=0). When a task is allocated to an agent, the corresponding non-negative costs $c^1_{rl},\ldots,c^p_{rl}$ are incurred. The MOAP can be stated as follows:

$$\min \sum_{r=1}^{n}\sum_{l=1}^{n} c^j_{rl}x_{rl} \quad j = 1,\ldots,p \tag{1}$$

$$\text{s.t.} \sum_{l=1}^{n} x_{rl} = 1 \quad r = 1,\ldots,n \tag{2}$$

$$\sum_{r=1}^{n} x_{rl} = 1 \quad l = 1,\ldots,n \tag{3}$$

$$x_{rl} \in \{0,1\} \quad r,l = 1,\ldots,n. \tag{4}$$

The objective of the MOAP is to find an optimal assignment of all tasks to agents while minimising p cost functions (1). Equation (2) ensures that each agent is assigned to only one task. Equation (3) limits each task to be assigned to one agent only.

It is well-known that the constraint matrix of AP is totally unimodular, then every vertex of the LP relaxation is an integer vector. Thus, by solving the LP relaxation we can naturally obtain integer solutions of the AP.

Although *Bensolve* only computes supported efficient solutions, these solutions already provide a high-quality approximation of PF. The corresponding computational results are given in Table 2 in Section 5. In conclusion, the MOAP may not be a suitable benchmark for MOIP heuristics.

3.4.2 Multi-objective Knapsack Problem

In the multi-objective knapsack problem, a set of items is given, each with a certain weight w_r, and we must select a subset of these items such that the total weight does not exceed a given capacity W. The decision variable x_r denotes whether item r is selected for the knapsack (x_r=1) or not (x_r=0). Each item r also has profits $v^1_r \ldots v^p_r$. Here, W, v_r, and w_r are non-negative integer values. The MOKP model is stated as follows:

$$\max \sum_{r=1}^{n} v^j_r x_r \quad j = 1,\ldots,p \tag{5}$$

$$\text{s.t.} \sum_{r=1}^{n} w_r x_r \leq W \tag{6}$$

$$x_r \in \{0,1\} \quad r = 1,\ldots,n \tag{7}$$

Equation (5) denotes the objective functions maximising the p total profits of selected items. Equation (6) is the capacity constraint. The total weight of the items placed in the knapsack cannot exceed the given capacity.

In this paper, we convert a maximisation objective function into a minimisation one by multiplying it by -1.

4 LP RELAXATION-BASED MATHEURISTIC

This section provides the overview of the proposed algorithm and its main ingredients.

4.1 Algorithmic Framework

The proposed matheuristic algorithm follows a two-stage approach. At the first stage, we obtain LB sets and adapt them to feasible integer rounded sets. A variant of the weighted sum method by Özpeynirci and Köksalan (2010), *Bensolve* by Löhne

and Weißing (2017) and *Inner approximation* by Csirmaz (2020) have been investigated to find LB sets first. Among the three methods, *Bensolve* shows competitive performance in the experiment and produces all the required bound set information we need. The performance comparison can be found in the Appendix. In the context of MOKP, once LB sets (L) are obtained, we round down the fractional variable to produce a feasible solution. These are referred to as integer rounded (IR) sets. At the second stage, *PR* is employed to improve the solution quality. Until reaching the iteration limit, the *PR* process repeats. If *PR* finds a new solution, it is stored in the archive *candX*. Based on initial experiments, we set the limit of iterations of *PR* to *the number of solutions of the IR set times 50*. Since the dominance relation is not checked in $\tilde{X}$ during the search, dominated solutions are filtered after the algorithm terminates, and the set of integer feasible solutions X is returned. Algorithm 1 describes the general framework of the proposed algorithm.

Algorithm 1: The LP relaxation-based matheuristic framework.

Input: L: points describing an LB set
1 *candX*: an archive of newly found feasible IP solutions by *PR*
2 $\tilde{X}$: an empty list
3 $IR \leftarrow RoundingDown(L)$;
4 $i = 0$;
5 **for** $i <$ *iteration limit* **do**
6 $\quad candX \leftarrow PathRelinking(IR, S_i, S_g, IGPair)$;
7 $\quad \tilde{X} \leftarrow \tilde{X} \cup candX$;
8 $\quad i = i + 1$;
9 $X \leftarrow DominanceCheck(IR \cup \tilde{X})$;
Output: X

4.2 Path Relinking

Although IR sets obtained from *rounding* are feasible, they are not necessarily of high-quality. Therefore, we employ *PR* to increase the number of solutions and improve the quality of the approximate *PF*. *PR* was originally introduced by Glover (1997). The main idea of it is that there should be common characteristics among high-quality solutions. The method produces new solutions by exploring solution "paths" between pairs of known solutions. To generate a new solution, *PR* chooses two solutions from a set of initial solutions; an initiating solution (S_i) and a guiding solution (S_g) represent the starting and ending points of the path, respectively. Then it explores the trajectories that link the pair of solutions in a neighbourhood space. In each step, a certain number of intermediate paths, called a neighbourhood, are created. For the next step, the new initiating solution is selected in there. As the search proceeds, more attributes of the guiding solutions are gradually passed on to intermediate solutions. The search continues until the initiating solution becomes identical to the guiding solution.

The method has been applied successfully to MO problems such as the travelling salesman problem (Jaszkiewicz, 2005), the dial-a-ride problem (Parragh et al., 2009), and a school bus routing problem (de Souza Lima et al., 2017). It also produces high-quality solutions on large MOKP instances within a reasonable timeframe (Beausoleil et al., 2008; Martí et al., 2015). For these reasons, we combine *PR* with the first stage approach.

For illustration purposes, we provide a short example describing the *PR* process on a MOKP with three objectives. Suppose we have a four-item MOKP in which the initiating solution is (0 0 1 0) and the guiding solution is (1 1 0 0). In this case, the only selected item in common is the fourth item. Therefore, three intermediate paths are created by the following rules. To create one path, one variable value in the initiating solution is switched. To be specific, if the item is placed in the knapsack and its value is 1, it is taken out of the knapsack and the value changes to 0. If the item is not chosen and the value equals 0, it is placed in the knapsack and the value changes to 1. This procedure applies to all the different variable values. The outcome of the process is seen in the first row (Neighbourhood) of Table 1. When the P matrix (8) represents the profits of the three objectives, the sets of total profits that correspond to the first neighbourhood are [7,6,8], [5,4,6], and [0,0,0]. Since the dominance relation among the solution points is clear, the first path (1 0 1 0) is selected as the new initiating solution (marked with * in Table 1). This process repeats until the initiating solution becomes identical to the guiding solution. Table 1 illustrates the transforming process of *PR* in this example.

$$P = \begin{pmatrix} 4 & 2 & 3 & 6 \\ 5 & 3 & 1 & 8 \\ 6 & 4 & 2 & 7 \end{pmatrix} \tag{8}$$

Table 1: *Path relinking* procedure.

Initiating	Guiding	Neighbourhood		
0 0 1 **0**	1 1 0 **0**	1 0 1 0*	0 1 1 0	0 0 0 0
1 0 1 **0**	**1** 1 0 **0**	1 1 1 0*	1 0 0 0	
1 1 1 **0**	**1 1** 0 0	1 1 0 0*		
1 1 0 0	**1 1 0 0**			

In our *PR* process, an infeasible intermediate solution is not stored in the solution archive, but still can be a new initiating solution. This is for letting the algorithm explore a broader search region so that it possibly finds diverse solutions.

Depending on the specific problem, the following components of *PR* can be designed differently in the algorithm.

1) Building an initial solution set
2) Selecting S_i and S_g
3) Generating intermediate paths (a neighbourhood)
4) Choosing the next initiating solution

In this study:

1) We use IR sets as the initial solution set.
2) S_i and S_g are chosen either randomly or based on *similarity* between them.
3) A new neighbourhood is generated as many times as the number of different variable values.
4) The next S_i is determined either by dominance relation based analysis or randomly.

The description of the entire *PR* implementation for the proposed algorithm is given in Algorithm 2.

The algorithm maintains two archives *candX* and *IGPair*. Each archive stores newly found solutions and used S_i-S_g pairs during the *PR* process. Since the performance of *PR* can vary depending on the choice of the S_i-S_g pair, we investigate different ways, and call *SelectionRuleI*. The first approach is to select two random solutions from *IR*. For the second and third methods, S_i is chosen at random in *IR* first, then the *similarity* between S_i and all the other solutions in *IR* is calculated. The *similarity* is measured by counting the number of variables which have equal values. Thus, higher values imply a greater *similarity*. The second method selects S_g by finding the most similar solution to S_i. The third approach finds the most different solution to S_i for S_g. When S_i and S_g are similar, fewer intermediate paths are likely to be created. However, when they are different, the *PR* process can take longer as it explores relatively diverse intermediate paths.
Once S_i and S_g are chosen by *SelectionRuleI* (line 4), *PR* continues until one of the following two termination conditions is met (line 5):

- The initial and guiding solutions are identical.
- The pair of S_i and S_g has already been chosen.

To decide the number of intermediate paths, we need to find the positions of the differing variable values in S_i and S_g. They are stored in Δitems as indices (lines 6-7). Once Δitems is defined, we generate the neighbourhood by the following rules: For each index in Δitems, the variable values in S_i change one by one (line 8). Let us suppose items 1 and 5 have a different value, $\Delta\text{items} = \{1,5\}$. If item 1 is selected in S_i (variable value=1), then it is taken out of it. If it is out of the knapsack (variable value=0), then it is added to it. The same process is conducted on item 5. Once intermediate paths are built, we select the next S_i. In order to avoid local optima, the best neighbour is not systematically selected. Rather, it is selected with a certain probability, set to 0.7 after initial experiments. Otherwise, another neighbour is selected randomly. In the case where we do want to select the best neighbour, we analyse the dominance relation among intermediate solutions and select the best solution as S_i (line 11). For this, *SelectionRuleII* is introduced. If one non-dominated solution exists in the neighbourhood, it becomes the next initiating solution. If there are mutually non-dominating solutions, we check the *improved ratio* of each solution point. The approach for finding the most-improved point is explained in detail in Section 4.2.1. For the case where we do not want to select the best neighbour, the analysis is not needed and simply one random solution is selected from the neighbourhood (lines 12-13). Before moving on to the next iteration, the algorithm checks the feasibility of S_i (line 14). If S_i is feasible and not included in *IR*, it is stored in the archive *candX* (line 16). The infeasible S_i is not archived, though, it is still used for the next iterations as it might help to find additional feasible solutions in yet unexplored parts of the search region. *IR* is updated by adding S_i (line 15). To prevent the case in which the same pair of initial and guiding solution is chosen, the current $[S_i, S_g]$ pair is archived after every *PR* iteration (line 16). The algorithm returns the set of integer feasible solutions *candX*.

4.2.1 *ImprovedND* Operation

The *ImprovedND* operation figures out which path shows the biggest improvement compared to the current solution S_i. Algorithm 3 shows a precise description of the operation.

Let *ND* be a set of non-dominated intermediate points and $objS_i$ be the objective values of S_i. To record the improvement of each non-dominated point, we create two matrices and one list. The *ratio_table* stores the ratio of a non-dominated point to the current point $objS_i$, for each objective (lines 4-6). Afterwards, we assign a rank to each column of the *ratio_table* and enter rankings into the *rank_table* (lines 7-10). The rankings of each non-dominated point are added up (lines 11-12) and stored in *ND_degree*. The non-dominated path with the highest degree is set to S_i (line 13).

Algorithm 2: The framework of path relinking.

Input: IR, S_i, S_g, $IGPair$
$IGPair$: an archive of S_i-S_g pairs
$candX \leftarrow \emptyset$;
$IGPair \leftarrow \emptyset$;
Select S_i,S_g from IR following *SelectionRuleI*;
while $S_i \neq S_g$ *and* $[S_i, S_g] \notin IGPair$ **do**
  Δ items $\leftarrow$ index set of different variable values;
  $n \leftarrow$ #indices in $\Delta items$;
  Create n neighbourhood;
  The best move analysis:
  if *rand()* < 0.7 **then**
    S_i is chosen by *SelectionRuleII*
  else
    $S_i \leftarrow$ randomly choose one solution from the neighbourhood;
  Feasibility check of S_i;
  if S_i *is feasible and* $S_i \notin IR$ **then**
    $candX \leftarrow S_i$;
    $IR \leftarrow IR \cup S_i$;
$IGPair \leftarrow [S_i, S_g]$;
Output: $candX$

Algorithm 3: *ImprovedND*.

Input: $objS_i$, ND
ratio_table: $|ND| \times p$ matrix
rank_table: $|ND| \times p$ matrix
ND_degree : list of size $|ND|$
for i=1,...,$|ND|$ **do**
  for j=1,...,p **do**
    $ratio\_table[i][j] = \frac{ND[i][j]}{objS_i[j]}$
for i=1,...,$|ND|$ **do**
  for j=1,...,p **do**
    rank_table[i][j] = rank of $ND[i][j]$ in j^{th} column
for i=1,...,$|ND|$ **do**
  ND_degree[i] $\leftarrow$ sum(*rank_table* i^{th} column)
$S_i \leftarrow ND$ with the highest degree
Output: S_i

5 COMPUTATIONAL EXPERIMENTS

We evaluate the performance of the different versions of our heuristic method, focusing on the comparison with FPBH (Pal and Charkhgard, 2019b). The variant with *rounding down* is named *RD*. The following *PR* variants are tested. The first three *PR* variants randomly choose the next initiating solution during the iterations.

- *PRrand*: This version randomly selects both the initiating and guiding solutions from the initial solution set.
- *PR*sim: This version randomly chooses the initiating solution, then finds the most similar solution among the remaining solution set for the guiding solution.
- *PR*dif: This version finds the most different solution to the already chosen initiating solution for the guiding solution.

The three other variants consider the improvement among mutually non-dominating intermediate paths to choose the next initiating solution.

- *PI*: This version selects the most improved intermediate solution as the next initiating solution.
- *PI*sim: This variant uses *ImprovedND* within *PR*sim.
- *PI*dif: This variant uses *ImprovedND* in *PR*dif.

The proposed algorithms use *Bensolve* by Löhne and Weißing (2017) to obtain the bound sets. The heuristic integration (*rounding down* and *PR*), is implemented in Julia. For the benchmark algorithm, we used the Julia implementation of FPBH (with the default setting) which is publicly available at `https://github.com/aritrasep/FPBH.jl`. All experiments of matheuristics are carried out on Intel® Core™ i5-8250U CPU running at 1.60GHz with 16GB RAM. The exact MOIP solver proposed by Kirlik and Sayın (2014) (KS) is also used in the experiment to obtain the true *PF* for comparison purpose. KS is run on Quad-core X5570 Xeon CPUs @2.93GHz with 48GB RAM. The KS results are for reference only (i.e. not for benchmarking).

The test instances we use are the same ones on which FPBH is tested. The instances were generated by Kirlik and Sayın (2014) and are publicly available at `http://home.ku.edu.tr/~moolibrary/`. Each problem class has 100 instances divided into 10 subclasses, each of which contains 10 instances. MOAP instances are formed in the number of tasks (to be assigned) which varies from 5 to 50 in increments of 5. MOKP instances are classified by the number of items which varies from 10 to 100 in increments of 10.

5.1 Performance Measure: Hypervolume Indicator

One widely used indicator to measure the quality of a solution set in MO optimisation is the hypervolume (HV) indicator. HV measures the volume of the dominated space of all the solutions contained in a solution set. To calculate the dominated space, a reference point must be used. Usually, a reference point is the "worst possible" point in the objective space. In this study, all the HV values are calculated with normalised objective values.

Let Y_N^k be a set of k^{th} objective values of the true PF and $y \in \mathbb{R}^p$ be an arbitrary non-dominated point obtained from a heuristic algorithm. Then, the normalised values of the obtained point are:

$$\frac{y^k - min(Y_N^k)}{max(Y_N^k) - min(Y_N^k)} \quad k = 1, \ldots, p.$$

As all non-dominated points are normalised, their values exist in [0,1]. Therefore, the reference point is (1,1,1). Higher HV values indicate a better approximation. We used the publicly available HV computing program provided by Fonseca et al. (2006) at `http://lopez-ibanez.eu/hypervolume#intro` to obtain HV values.

5.2 Results and Discussion

We report the following results of each algorithm: the number for solutions ($|Y|$), CPU time (sec), HV value, and HV as a percentage of the HV indicator value for the exact non-dominated set as provided by Kirlik and Sayın (2014). All the figures are average results over 10 test instances. The figures of *PR* variants and FPBH are averaged over 10 runs for each instance because they have random components. The results of the experiments on MOAP and MOKP are reported in Tables 2-5.

Bensolve already finds integer feasible solutions for all the MOAP instances. In addition, it shows better performance than FPBH in all the subclasses. Therefore, we do not apply our matheuristic to MOAP instances.

In general, both the number of solutions and computation time increase as the size of instances becomes larger. The difference between FPBH and *Bensolve* is clearly noticeable in Table 2, which shows that *Bensolve* outperforms FPBH in all the MOAP instances. Furthermore, the difference becomes greater as the size of the instances grows. For example, for the instances with more than 15 tasks (n≥15), the number of solutions of *Bensolve* is more than double that of FPBH. Further, solutions are found in significantly less time. Hence, not only the quantity but also the quality of the solutions of *Bensolve* are better than FPBH. It reaches more than 99% of the maximum

Table 2: Comparing algorithm performance on MOAP for p=3, * indicates optimal values, best heuristic values are in bold.

n	$\|Y\|$			CPUtime(sec)			HV			HV(%)	
	KS*	FPBH	*Bensolve*	KS	FPBH	*Bensolve*	KS	FPBH	*Bensolve*	FPBH	*Bensolve*
5	14.1	6.7	**7.5**	0.11	0.05	**0.001**	6.77	6.60	**6.71**	97.49	**99.11**
10	176.8	21.0	**39.0**	10.71	0.28	**0.019**	7.23	6.92	**7.18**	95.71	**99.31**
15	674.9	40.4	**83.1**	92.51	1.17	**0.068**	7.33	6.96	**7.29**	94.95	**99.45**
20	1860.5	62.5	**161.3**	359.07	2.04	**0.222**	7.40	6.99	**7.37**	94.46	**99.59**
25	3567.8	90.6	**253.1**	872.19	4.99	**0.596**	7.46	7.05	**7.44**	94.50	**99.73**
30	6181.3	140.0	**379.4**	1859.74	15.59	**1.157**	7.48	7.04	**7.46**	94.12	**99.73**
35	8972.3	163.2	**501.4**	3285.57	26.39	**2.082**	7.50	7.06	**7.48**	94.13	**99.73**
40	14679.7	242.9	**699.1**	6425.98	57.95	**3.824**	7.53	7.09	**7.51**	94.16	**99.73**
45	17702.2	238.6	**838.0**	9239.01	82.16	**6.097**	7.56	7.10	**7.54**	93.92	**99.74**
50	24916.8	337.5	**1034.8**	15814.82	119.52	**9.739**	7.58	7.12	**7.56**	93.93	**99.74**

Table 3: Comparing $|Y|$ of algorithms on MOKP for p=3, * indicates optimal values, best heuristic values are in bold.

				PR variants					
n	KS*	FPBH	*RD*	*PRrand*	*PR*sim	*PR*dif	*PI*	*PI*sim	*PI*dif
10	9.8	5.1	4.3	5.7	4.8	4.8	**6.3**	4.6	4.6
20	38.0	18.1	10.7	22.4	20.7	19.0	**26.1**	20.4	18.6
30	115.8	43.2	20.7	47.9	46.9	45.6	**58.6**	46.3	45.0
40	311.2	95.7	33.8	95.8	96.4	91.3	**122.0**	97.8	93.2
50	444.2	111.8	41.7	119.1	118.1	114.9	**143.5**	118.4	112.6
60	917.1	195.1	71.5	209.1	209.8	203.6	**266.2**	207.8	208.4
70	1643.4	348.2	90.2	263.0	264.4	259.8	**353.4**	271.3	264.9
80	2295.8	**439.0**	113.1	305.1	301.1	310.1	399.9	313.7	309.2
90	3107.8	**501.9**	130.6	322.7	319.4	327.0	412.7	338.7	343.3
100	5849.0	**919.2**	176.7	442.8	453.0	439.8	581.3	469.4	471.9

Table 4: Comparing CPU time (sec) of algorithms on MOKP for p=3, * indicates optimal values, best heuristic values are in bold.

				PR variants					
n	KS*	FPBH	*RD*	*PRrand*	*PR*sim	*PR*dif	*PI*	*PI*sim	*PI*dif
10	0.140	0.023	**0.001**	0.002	0.004	0.003	0.006	0.002	0.002
20	1.030	0.080	**0.004**	0.020	0.056	0.044	0.067	0.051	0.036
30	5.540	0.324	**0.008**	0.085	0.314	0.281	0.307	0.312	0.268
40	23.23	1.071	**0.013**	0.224	0.949	0.896	0.824	0.885	0.763
50	40.07	1.941	**0.019**	0.379	1.752	1.629	1.517	1.684	1.416
60	116.0	5.332	**0.041**	1.105	5.490	5.095	4.872	5.445	4.686
70	283.5	12.68	**0.054**	2.118	10.69	9.818	10.72	10.64	9.118
80	440.0	20.77	**0.079**	3.533	18.18	16.21	17.49	17.98	15.31
90	833.9	42.17	**0.102**	6.121	28.89	26.50	30.04	28.18	24.41
100	2478.4	82.54	**0.129**	11.23	59.78	57.50	66.97	60.57	53.24

HV throughout all the problem sets. Furthermore, the HV value increases as the problem size gets larger. On the other hand, the highest HV value of FPBH is 97.49% in the smallest problem class (n=5), and it decreases as the problem size increases. This suggests that MOAP is not a suitable benchmark problem for MOIP heuristics.

In the case of the MOKP, FPBH and the *PR* variants are competitive in terms of the number of solutions for instances with fewer than or equal to 70 items (n$\leq$ 70). For $n \geq 80$, FPBH generates more solutions than *PR* variants. *PR* variants take less computation time than FPBH in all instances. Notably, *PRrand* takes less than a quarter of the time than FPBH does. Skipping *SelectionRules* and the best move analysis, but instead relying on randomness helps to reduce CPU time. Except for *PI*, embedding the *ImprovedND* operation into the *PR* heuristics does not bring any noticeable difference in the number of solutions or run time. *PI* finds the most solutions among all *PR* variants while its CPU time increases fivefold. We also observe that it finds better solutions while spending longer running time in Table 5. *RD* always produces the fewest solutions among the heuristic methods. However, it takes considerably less CPU time. For instance, it takes less than 1 second regardless of the problem size as it is seen in Table 4. Although FPBH finds more solutions for larger instances, every *PR* variant achieves a higher HV than FPBH, except for the smallest problem class with n=10. In particular, *PI* generates the highest quality solutions throughout the experiments. *RD* also outperforms FPBH for the instances with more than 30 items.

We observe that the *RD* and *PR* variants do not show better performance than FPBH on the smallest instances. The reason for this may be the structure of the test instances. When a problem size is small, a small fractional value has a relatively big impact on each objective. When a fractional value is rounded down, the solution quality deteriorates comparably more on smaller instances. In addition, the number of initially provided LB set solutions is limited in smaller instances. For these reasons, *RD* has a

Table 5: Comparing HV(%) of algorithms on MOKP for p=3, * indicates optimal values, best heuristic values are in bold.

				PR variants					
n	KS*	FPBH	*RD*	*PRrand*	*PR*sim	*PR*dif	*PI*	*PI*sim	*PI*dif
10	6.58	**6.28 (95.4)**	5.91 (89.8)	6.02 (91.5)	5.96 (90.6)	5.95 (90.4)	6.03 (91.6)	5.94 (90.3)	5.93 (90.1)
20	6.97	6.70 (96.1)	6.61 (94.8)	6.75 (96.8)	6.72 (96.4)	6.71 (96.3)	**6.77 (97.1)**	6.71 (96.3)	6.70 (96.1)
30	7.21	6.92 (96.0)	6.96 (96.5)	7.05 (97.8)	7.04 (97.6)	7.04 (97.6)	**7.06 (97.9)**	7.04 (97.6)	7.04 (97.6)
40	7.14	6.86 (96.1)	6.95 (97.3)	7.01 (98.2)	7.01 (98.2)	7.01 (98.2)	**7.03 (98.5)**	7.01 (98.2)	7.01 (98.2)
50	7.23	7.00 (96.8)	7.05 (97.5)	7.09 (98.1)	7.09 (98.1)	7.09 (98.1)	**7.10 (98.2)**	7.09 (98.1)	7.09 (98.1)
60	7.18	6.95 (96.8)	7.03 (97.9)	7.06 (98.3)	7.06 (98.3)	7.06 (98.3)	**7.07 (98.5)**	7.06 (98.3)	7.06 (98.3)
70	7.19	6.98 (97.1)	7.05 (98.1)	7.08 (98.5)	7.08 (98.5)	7.08 (98.5)	**7.09 (98.6)**	7.08 (98.5)	7.08 (98.5)
80	7.22	7.05 (97.6)	7.10 (98.3)	**7.12 (98.6)**	7.11 (98.5)	**7.12 (98.6)**	**7.12 (98.6)**	7.11 (98.5)	**7.12 (98.6)**
90	7.20	7.01 (97.4)	7.08 (98.3)	**7.10 (98.6)**	**7.10 (98.6)**	**7.10 (98.6)**	**7.10 (98.7)**	**7.10 (98.6)**	**7.10 (98.6)**
100	7.19	6.99 (97.2)	7.08 (98.5)	7.09 (98.6)	7.09 (98.6)	7.09 (98.6)	**7.10 (98.7)**	7.09 (98.6)	7.09 (98.6)

large HV gap. The quality of *RD* also influences that of *PR* variants. If very small IR sets are provided, the choice of the S_i-S_g pair is restricted. This causes a limited number of new paths to be generated.

6 CONCLUSION

In this study, we propose a LP relaxation-based matheuristic for three-objective binary integer programming. The proposed algorithm relies on a high-performing vector LP solver, *Bensolve*, which provides bound sets, and two simple heuristics, *rounding down* and path relinking (*PR*). In the computational study, we show that simple *rounding down* can already find high-quality solutions in most instances. After embedding *PR* with the first stage, the proposed heuristic generates more solutions, which show higher quality than that of FPBH in most problem classes. The number of solutions and CPU times of the *PR* variants are similar to each other. Notably, the *PRrand* algorithm takes less than a quarter of the computation time FPBH does. The biggest advantage of the proposed algorithm is that it can find high-quality approximations fast, which also shows its effectiveness.

For future work, we plan to extend the algorithm to deal with general MOIPs. Further, it could be tailored to real-world applications such as supply chain network design. As the size of real-world problems be much larger, approaches to further reduce the computation time will be investigated. In addition, the proposed method can be embedded into an interactive algorithm in order to facilitate decision making.

REFERENCES

Beausoleil, R. P., Baldoquin, G., and Montejo, R. A. (2008). Multi-start and path relinking methods to deal with multiobjective knapsack problems. *Annals of Operations Research*, 157(1):105–133.

Boland, N., Charkhgard, H., and Savelsbergh, M. (2017). The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs. *European Journal of Operational Research*, 260(3):873–885.

Braekers, K., Hartl, R. F., Parragh, S. N., and Tricoire, F. (2016). A bi-objective home care scheduling problem: Analyzing the trade-off between costs and client inconvenience. *European Journal of Operational Research*, 248(2):428–443.

Csirmaz, L. (2020). Inner approximation algorithm for solving linear multiobjective optimization problems. *Optimization*, pages 1–25.

de Souza Lima, F. M., Pereira, D. S., da Conceição, S. V., and de Camargo, R. S. (2017). A multi-objective capacitated rural school bus routing problem with heterogeneous fleet and mixed loads. *4OR*, 15(4):359–386.

Ehrgott, M. and Gandibleux, X. (2007). Bound sets for biobjective combinatorial optimization problems.

Computers & Operations Research, 34(9):2674–2694.

Fischetti, M., Glover, F., and Lodi, A. (2005). The feasibility pump. *Mathematical Programming*, 104(1):91–104.

Fonseca, C. M., Paquete, L., and López-Ibánez, M. (2006). An improved dimension-sweep algorithm for the hypervolume indicator. In *2006 IEEE international conference on evolutionary computation*, pages 1157–1163. IEEE.

Glover, F. (1997). Tabu search and adaptive memory programming—advances, applications and challenges. In *Interfaces in computer science and operations research*, pages 1–75. Springer.

Jaszkiewicz, J. (2005). Path relinking for multiple objective combinatorial optimization: Tsp case study. In *The 16th Mini-EURO Conference and 10th Meeting of EWGT (Euro Working Group Transportation)*.

Kirlik, G. and Sayın, S. (2014). A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3):479–488.

Kolli, S. and Evans, G. W. (1999). A multiple objective integer programming approach for planning franchise expansion. *Computers & Industrial Engineering*, 37(3):543–561.

Kovacs, A. A., Parragh, S. N., and Hartl, R. F. (2015). The multi-objective generalized consistent vehicle routing problem. *European Journal of Operational Research*, 247(2):441–458.

Leitner, M., Ljubić, I., Sinnl, M., and Werner, A. (2016). Ilp heuristics and a new exact method for bi-objective 0/1 ilps: Application to fttx-network design. *Computers & Operations Research*, 72:128–146.

Löhne, A. and Weißing, B. (2017). The vector linear program solver bensolve–notes on theoretical background. *European Journal of Operational Research*, 260(3):807–813.

Martí, R., Campos, V., Resende, M. G., and Duarte, A. (2015). Multiobjective grasp with path relinking. *European Journal of Operational Research*, 240(1):54–71.

Mavrotas, G. and Florios, K. (2013). An improved version of the augmented ε-constraint method (augmecon2) for finding the exact pareto set in multi-objective integer programming problems. *Applied Mathematics and Computation*, 219(18):9652–9669.

Özpeynirci, Ö. and Köksalan, M. (2010). An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, 56(12):2302–2315.

Pal, A. and Charkhgard, H. (2019a). A feasibility pump and local search based heuristic for bi-objective pure integer linear programming. *INFORMS Journal on Computing*, 31(1):115–133.

Pal, A. and Charkhgard, H. (2019b). Fpbh: A feasibility pump based heuristic for multi-objective mixed integer linear programming. *Computers & Operations Research*, 112:104760.

Parragh, S. N., Doerner, K. F., Hartl, R. F., and Gandibleux, X. (2009). A heuristic two-phase solution approach for the multi-objective dial-a-ride problem. *Networks: An International Journal*, 54(4):227–242.

Sawik, B., Faulin, J., Perez-Bernabeu, E., and Serrano-Hernandez, A. (2016). Bi-objective optimization models for green vrp approaches. *ICIL 2016*, page 247À254.

Soylu, B. (2015). Heuristic approaches for biobjective mixed 0–1 integer linear programming problems. *European Journal of Operational Research*, 245(3):690–703.

Zhang, W. and Reimann, M. (2014). A simple augmented εconstraint method for multi-objective mathematical integer programming problems. *European Journal of Operational Research*, 234(1):15–24.

APPENDIX

Table 6 shows the comparison of three state-of-the-art algorithms that can deal with multi-objective linear programming. *Bensolve* (Ben) by Löhne and Weißing (2017) and *Inner solver* (Inner) by Csirmaz (2020) are implemented in C and publicly available at `http://www.bensolve.org/` and https://github.com/lcsirmaz/inner, respectively. The algorithm suggested by Özpeynirci and Köksalan (2010) (ÖK) is implemented in Julia. GLPK is used as LP solver. The time limit of the experiment is 3600 seconds. All the experiments are carried out on a Quad-core X5570 Xeon CPUs @2.93GHz with 48GB RAM. The figures are the average results of 10 test instances over 10 runs. As benchmark instances, we used the multi-objective assignment problem (MOAP), the multi-objective knapsack problem (MOKP), and multi-objective general integer linear programming problems (MOILP) which are all generated by Kirlik and Sayın (2014) and available at http://home.ku.edu.tr/ moolibrary/. Each problem class is divided into subclasses. The subclasses are categorised by the number of items. For the MOAP, it is categorised by 5/10/15/30/50, whereas for the MOKP and MOILP, the subclasses are 10/30/50/70/100. Each subclass has 10 instances; thus, there are 50 instances per class in total. We report the CPU time (sec) and the number of LPs. n/a. indicates that the algorithm did not terminate within the time limit. ^{*number} indicates the number of instances solved out of 10 instances.

Throughout the experiment, Ben and Inner are highly competitive. In terms of the number of solved LPs, the *Inner solver* comprehensively outperforms the other two methods. Overall, the *Inner solver* performs the best. However, it does not provide all

Table 6: Comparing algorithms on MOLP instances with p=3.

Problem	#item	CPUtime(sec)			#solved LP		
		Ben	Inner	ÖK	Ben	Inner	ÖK
MOAP	5	0.004	**0.003**	2.46	30.2	**23.0**	42.0
	10	0.03	**0.02**	30.20	117.2	**110.1**	1916.8
	15	0.10	**0.07**	844.04	237.6	**230.5**	12845.7
	30	**1.57**	1.59	n/a.	1015	**1008**	n/a.
	50	**13.51**	19.09	n/a.	2705	**2698**	n/a.
MOKP	10	0.003	**0.002**	4.27	46.0	**39.0**	300.7
	30	0.02	**0.01**	320.98	223.1	**216.0**	7749.0
	50	0.03	**0.02**	320.98	464.6	**454.7**	n/a.
	70	**0.07**	**0.07**	n/a.	978.6	**920.0**	n/a.
	100	0.16	**0.14**	n/a.	1962.0	**1397.0**	n/a.
MOILP	10	0.003	**0.001**	3.71[*7]	28.1	**18.3**	12.97[*7]
	30	**0.01**	0.05	449.9[*8]	79.5	**70.1**	3481.8[*8]
	50	0.02	**0.01**	1295.4[*2]	134.71	**124.3**	3764.5[*2]
	70	0.04	**0.02**	32.1[*2]	178.2	**167.0**	857.0[*2]
	100	0.07	**0.03**	3082.2[*1]	237.4	**223.53**	12835.0[*1]

the bound set information we need for our heuristic.
Thus, we use *Bensolve* to obtain the initial bound sets.