Multi-objective integer programming: An improved recursive algorithm

Ozlen, Melih; Burton, Benjamin; MacRae, Cameron

Ozlen, M., Burton, B., & MacRae, C. (2014). Multi-objective integer programming: An improved recursive algorithm. Journal of Optimization Theory and Applications, 160(2), 470–482.
https://doi.org/10.1007/s10957-013-0364-y
Document Version: Accepted Manuscript

Published Version: https://doi.org/10.1007/s10957-013-0364-y

Multi-objective integer programming: An improved recursive algorithm

Melih Ozlen, Benjamin A. Burton, Cameron A. G. MacRae

Abstract

This paper introduces an improved recursive algorithm to generate the set of all nondominated objective vectors for the Multi-Objective Integer Programming (MOIP) problem. We significantly improve the earlier recursive algorithm of Özlen and Azizoğlu by using the set of already solved subproblems and their solutions to avoid solving a large number of IPs. A numerical example is presented to explain the workings of the algorithm, and we conduct a series of computational experiments to show the savings that can be obtained. As our experiments show, the improvement becomes more significant as the problems grow larger in terms of the number of objectives.

Keywords: Multiple objective programming, Integer programming

1 Introduction

Multi-Objective Integer Programming (MOIP) has drawn the attention of researchers in recent years, as discussed in section 2. MOIP is seen as an extension to classical Integer Programming (IP) which has already been used in a wide variety of decision making environments including logistics, planning, location/allocation, scheduling, routing, and so on. Multiple objectives enable decision makers to consider not just a single objective but a *set* of objectives simultaneously, such as cost, profit, waste, environmental impact, risk, etc.

In this study we deal with the noninteractive and exact solution of the MOIP problem, in the case where there is no information available on the form of the utility function. That is, we focus on algorithms for generating the full set of all nondominated objective vectors. However, in Section 2 we also discuss the literature on noninteractive and exact methods where information about the utility function is known. We refer the reader to Ehrgott [2005] for a more detailed discussion on multi-objective optimisation, theory, and methodology.

The main contribution of this paper is to improve the recursive algorithm of Özlen and Azizoğlu [2009] for generating the full nondominated set. A key drawback of the former algorithm is that it does not make use of any information obtained from the already solved subproblems. Our new algorithm incorporates this valuable information, and is able to avoid solving a large set of intermediate IPs as a result.

The remainder of this paper is organised as follows. Section 2 reviews the related literature, and in Section 3 we describe the problem and explain the original recursive algorithm. In Section 4 we introduce our improved algorithm, and Section 5 offers a detailed illustration of its workings using an instance of quad-objective assignment problem (QOAP). We present the results of a computational experiment in Section 6, and discuss the savings that can be obtained using this new algorithm. We conclude and provide several future research directions in Section 7.

2 Literature

Here we survey the literature on exact noninteractive approaches to MOIP, beginning with the special case in which an explicit utility function is known, and then moving towards the most general case of MOIP.

For the special case in which there is an explicit utility function, we aim to identify a solution that optimises the given utility function. Abbas and Chaabane [2006], Jorge [2009] deal with the case of a linear utility function and propose methods to identify an optimal solution, which must be one of the extreme supported nondominated objective vectors. Ozlen et al. [2012] handle the case of a nonlinear utility function, where the solution may be any member of the nondominated set.

In a more general case of a linear but unknown utility function, it is sufficient to identify all extreme supported nondominated objective vectors, since the optimal solution must come from this set. Przybylski et al. [2010b], Özpeynirci and Köksalan [2010] propose similar algorithms to identify all extreme nondominated objective vectors for the MOIP problem. Both algorithms use a weighted single objective function and partition the weight space in order to enumerate the extreme supported nondominated set; an approach first proposed for Multi-Objective Linear Programming (MOLP) by Benson and Sun [2000, 2002].

In the most general case where there is no information available about the utility function, the aim is to generate all nondominated objective vectors, since these can optimise an arbitrary linear or nonlinear utility function. Klein and Hannan [1982] develop an approach based on the sequential solutions of the single-objective models. Their algorithm generates a subset, but not necessarily the whole set, of all nondominated objective vectors. Sylva and Crema [2004] improve the approach of Klein and Hannan [1982] by defining a weighted combination of all objectives, and their approach guarantees to generate the full nondominated set. The main drawback of their method is that with every iteration a number of binary variables and constraints are added to the subproblems, making it impractical to solve problems which require a large number of iterations. Laumanns et al. [2005, 2006] develop an adaptive version of the ϵ-constraint method to generate all nondominated objective vectors; the main handicap of their algorithm is that it needs to solve a large number of IPs to generate weak nondominated objective vectors as it progresses.

Przybylski et al. [2010a] propose a generalisation of the two-phase algorithm for MOIP, where first extreme supported nondominated objective vectors are identified, and then the remaining nondominated vectors are identified using this earlier set. For the first phase, their algorithm requires an efficient method of generating extreme supported nondominated objective vectors for MOIP. This is relatively easy for problems with unimodular constraint sets, such as assignment, transportation or minimum cost network flow problems. In these easier cases, one can use any algorithm for generating extreme nondominated objective vectors for the MOLP problem; see Burton and Ozlen [2010] for a recent discussion on this topic. In general, however, generating all extreme supported nondominated objective vectors for MOIP is hard, as discussed in Przybylski et al. [2010b], Özpeynirci and Köksalan [2010]. Likewise, the second phase of their algorithm runs well for specific well-studied problems such as the assignment problem, but for general MOIP it remains difficult to use and requires problem-specific implementation; see Przybylski et al. [2009].

An alternative and general approach for generating all nondominated objective vectors for MOIP is given by Özlen and Azizoğlu [2009], whose algorithm recursively identifies objective efficiency ranges using problems with fewer objectives. This algorithm forms the basis for this paper, and we describe it in detail in the following section.

3 The problem and the recursive algorithm

In its most general form the MOIP problem is defined as:

Min $f_1(x), f_2(x), \ldots f_{k-1}(x), f_k(x)$
s.t $x \in X$
where X is the set of feasible points defined by $Ax = b, x_j \geq 0$ and $x_j \in \mathbb{Z}$ for all $j \in \{1, 2, \ldots, n\}$.

The individual objectives are defined as $f_1(x) = \sum_{j=1}^{n} c_{1j}x_j$, $f_2(x) = \sum_{j=1}^{n} c_{2j}x_j$, $\ldots$, and $f_k(x) = \sum_{j=1}^{n} c_{kj}x_j$, where $c_{ij} \in \mathbb{Z}$ for all $i \in \{1, 2, \ldots, k\}$ and $j \in \{1, 2, \ldots, n\}$.

A point $x' \in X$ is called *k-objective efficient* if and only if there is no $x \in X$ such that $f_i(x) \leq f_i(x')$ for each $i \in \{1, 2, \ldots, k\}$ and $f_i(x) < f_i(x')$ for at least one i. The resulting objective vector $(f_1(x'), f_2(x'), \ldots, f_k(x'))$ is said to be *k-objective nondominated*. There may be many efficient points in the decision space that correspond to the same nondominated objective vector, and so it can be extremely costly to generate all efficient points. Therefore the focus of this paper (as with most other papers of this type) is to generate the smaller set of nondominated vectors in objective space.

Özlen and Azizoğlu [2009] describe a recursive algorithm to generate the full set of nondominated objective vectors for the MOIP problem. A key tool in their algorithm is Constrained Lexicographic Multi Objective Integer Programming (CLMOIP), which is formulated as:

$$
\begin{array}{l}
\text{Lexicographic objective 1: Min } f_1(x), f_2(x), \ldots f_{k-1}(x) \\
\text{Lexicographic objective 2: Min } f_k(x) \\
\text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (*) \\
f_k(x) \leq l_k \\
x \in X.
\end{array}
$$

For any value of the bound l_k, each solution to this CLMOIP problem yields a nondominated objective vector for our original MOIP problem. The algorithm of Özlen and Azizoğlu [2009] essentially operates by repeatedly solving this CLMOIP problem, storing any solutions that are found, and then shrinking the bound l_k in order to generate new solutions (eventually terminating when the bound l_k is so small as to render the constraints infeasible). The recursion arises because the $(k-1)$-objective version of this same algorithm is used to minimise the first lexicographic objective above. Algorithm 1 describes the full process in pseudocode; see Özlen and Azizoğlu [2009] for details and proofs. The set ND_k returned by Algorithm 1 contains all k-objective non-dominated objective vectors for the original MOIP problem.

Lemma 1. *If M is the maximum value of f_k amongst all CLMOIP solutions, then after solving this CLMOIP problem we have identified all solutions to our original MOIP problem with $M = f_k \leq l_k$ (and typically several with $f_k < M$ also) and M provides an upper bound on the f_k values of all nondominated objective vectors satisfying the constraint $f_k(x) \leq l_k$.*

Proof. We only summarise the proof, see Özlen and Azizoğlu [2009] for a more detailed version of this proof.

i) For any MOIP problem where we minimise k objectives, any nondominated objective vector providing an upper bound on f_k values of all nondominated objective vectors should be nondominated with respect to first $k-1$ objectives otherwise this point would not be nondominated.

ii) Any MOIP problem with the additional constraint, $f_k(x) \leq l_k$, still is a MOIP problem so i) holds, thus any solution providing an upper bound on f_k should be nondominated with respect to first $k-1$ objectives for the constrained MOIP problem.

iii) Any CLMOIP problem defined above generates all nondominated objective vectors with respect to first $k-1$ objectives and at the same time minimises the k^{th} objective to make sure all the objective vectors generated are nondominated for k^{th} objective and thus nondominated with respect to all k objectives.

iv) Following i), ii) and iii), if M is the maximum value of f_k amongst all CLMOIP solutions, there cannot be any nondominated objective vector with $M < f_k \leq l_k$ and M provides an upper bound on the f_k values of all nondominated objective vectors satisfying the constraint on f_k. □

The minimum decrement of the objective function is 1, due to integer objective coefficients, which allows us to replace the bound l_k with $M-1$ in subsequent CLMOIP runs, as seen in Step 2 of Algorithm 1.

Remark 2. Algorithm 1 iterates by solving tighter versions of the CLMOIP problem due to the decreasing values of l_k that restrict the feasible region.

As can be seen in the numerical example presented in Section 5, this algorithm at the lowest level can end up solving a large number of IPs that generate the same nondominated objective vectors again and

Algorithm 1 Özlen and Azizoğlu [2009]'s recursive algorithm for the original MOIP problem

Step 0. Set $l_k = \infty$ and initialise ND_k to the empty set.

Step 1. Solve the CLMOIP problem $(*)$, using Algorithm 1 to optimise the first $k-1$ objectives.
If the problem is infeasible then,
STOP.
Let the $(k-1)$-objective nondominated set be ND^*_{k-1}

Step 2. $ND_k = ND_k \cup ND^*_{k-1}$.
$l_k = \max\{f_k \mid f \in ND^*_{k-1}\} - 1$.
Go to Step 1.

again. The main reason behind this is that the algorithm does not make use of already solved subproblems or their solutions.

4 An improved recursive algorithm

A major drawback of the recursive algorithm described above is that it does not store or utilise information on subproblems that have already been solved. With this in mind, we propose an improved algorithm that uses such information to avoid solving a large number of low-level IPs. The main idea is, when solving a new CLMOIP problem, to search for a *relaxation* of this problem that has been solved before, enabling us to skip the new CLMOIP problem entirely. We only allow relaxation of the constraints on individual objectives, thus avoid any issues that may arise from allowing a linear relaxation.

The following two lemmas show how a relaxation to a CLMOIP problem can be used to avoid solving it. Both results are straightforward, and so we omit the proofs.

Lemma 3. *Let $\mathcal{P}$ be a CLMOIP problem, and let $\mathcal{R}$ be a relaxation of $\mathcal{P}$. If $\mathcal{R}$ is infeasible, then $\mathcal{P}$ is also infeasible.*

Lemma 4. *Let $\mathcal{P}$ be a CLMOIP problem, and let $\mathcal{R}$ be a relaxation of $\mathcal{P}$. If every nondominated objective vector for $\mathcal{R}$ is also feasible for $\mathcal{P}$, then the set of all nondominated objective vectors for $\mathcal{P}$ is precisely the set of all nondominated objective vectors for $\mathcal{R}$.*

Remark 5. If $\mathcal{R}$ has even a single nondominated objective vector that is not feasible for $\mathcal{P}$, then the solution to $\mathcal{R}$ cannot be used to avoid solving $\mathcal{P}$.

As we recurse down through Algorithm 1, we accumulate constraints of the form $f_i \leq l_i$. In general, each intermediate CLMOIP problem that we solve is of the form:

Lexicographic objective 1: Min $f_1(x), f_2(x), \ldots f_q(x)$
Lexicographic objective 2: Min $f_{q+1}(x)$
s.t.
$f_{q+1}(x) \leq l_{q+1}$, $f_{q+2}(x) \leq l_{q+2}$, …, $f_k(x) \leq l_k$
$x \in X$.

We denote such a problem using the notation $(q, l_{q+1}, l_{q+2}, \ldots, l_k)$. It is straightforward to identify relaxations using this notation:

Lemma 6. *The CLMOIP problem $(q, l'_{q+1}, l'_{q+2}, \ldots, l'_k)$ is a relaxation of $(q, l_{q+1}, l_{q+2} \ldots, l_k)$ if $l'_i \geq l_i$ for all $i = q+1, \ldots, k$ and if $l'_i > l_i$ for some $i = q+1, \ldots, k$.*

Remark 7. Since Algorithm 1 iterates by incrementally lowering the bounds $l_2, l_3, \ldots, l_k$, as the algorithm progresses it becomes highly likely that we can find a relaxation of the current CLMOIP amongst our set of already solved problems.

Remark 8. There could be many relaxations to a given CLMOIP, each with different nondominated sets—all of the relaxations should be examined until one is found that allows us to avoid solving the current CLMOIP.

Using these ideas, Algorithm 2 improves the earlier Algorithm 1 by making use of already solved CLMOIP problems and their solution sets.

Algorithm 2 Improved recursive algorithm to generate nondominated set of MOIP

Step 0. Set $l_k = \infty$.

Step 1. Repeat:
Check the list of previously solved CLMOIPs to find a relaxation to the current CLMOIP problem $(*)$.
If all nondominated objective vectors for the relaxation are feasible for the current CLMOIP then,
Let that nondominated set be ND^*_{k-1} and go to Step 3.
If the relaxation is infeasible then,
STOP
Until there are no other relaxations to the current CLMOIP.

Step 2. Solve the CLMOIP problem $(*)$, using Algorithm 2 to optimise the first $k-1$ objectives.
If the problem is infeasible then,
STOP.
Let the $(k-1)$-objective nondominated set be ND^*_{k-1}

Step 3. $ND_k = ND_k \cup ND^*_{k-1}$.
$l_k = \max\{f_k \mid f \in ND^*_{k-1}\} - 1$.
Go to Step 1.

Set ND_k returned by Algorithm 2 resides all k-objective non-dominated objective vectors, stated formally:

Theorem 9. *Algorithm 2 generates all nondominated objective vectors for the original MOIP problem.*

Proof. Özlen and Azizoğlu [2009] show that Algorithm 1 generates all nondominated objective vectors. Algorithm 2 only differs in Step 1, and it is clear from Lemma 3 and Lemma 4 that these changes to Step 1 do not change the subsequent results. □

5 Numerical example

In this section we illustrate our approach on a numerical example. We use a randomly generated 4 objective assignment problem with the objective function coefficients listed in Table 1. This problem has 14 nondominated objective vectors that can be identified using Algorithm 1, as presented in Table 2 and its continuation Table 3. The rows of these tables show the various CLMOIP and IP problems as they are recursively solved.

The first column of these tables shows the CLMOIP problems with $q = 3$ that are solved at the highest level of the recursion. For instance, rows 1–20 follow the CLMOIP problem $(3, \infty)$; that is:

Lexicographic objective 1: Min $f_1(x), f_2(x), f_3(x)$
Lexicographic objective 2: Min $f_4(x)$
s.t.
$f_4(x) \leq \infty$
$x \in X$.

c_1						c_2						c_3						c_4					
	3	2	4	4	3		2	1	3	2	2		2	5	5	4	4		2	5	4	5	2
	5	4	3	4	3		3	5	5	1	1		4	2	4	2	5		2	4	1	2	2
	3	1	5	3	4		4	3	4	1	5		4	1	2	4	3		3	3	2	4	1
	5	5	4	5	3		4	5	5	2	4		4	4	1	3	1		1	3	4	3	5
	1	1	1	1	3		3	4	1	5	5		4	3	5	4	1		2	2	1	3	1

Tab. 1: Objective function coefficients for the example 4 objective assignment problem

The second column shows the CLMOIP problems with $q = 2$ that appear at the first level of recursion. For instance, rows 6–9 follow the problem $(2, 22, \infty)$:

Lexicographic objective 1: Min $f_1(x), f_2(x)$

Lexicographic objective 2: Min $f_3(x)$

s.t.

$f_3(x) \leq 22, f_4(x) \leq \infty$

$x \in X$.

The third column shows the CLMOIP problems with $q = 1$ at the deepest level of recursion. For instance, row 7 describes the problem $(1, 18, 22, \infty)$:

Lexicographic objective 1: Min $f_1(x)$

Lexicographic objective 2: Min $f_2(x)$

s.t.

$f_2(x) \leq 18,\ f_3(x) \leq 22, f_4(x) \leq \infty$

$x \in X$.

Each of these deepest problems (i.e., each table row) yields a new IP that Algorithm 1 must solve. The resulting nondominated objective vectors (obtained by minimising the remaining lexicographic objectives up the recursion stack) are given in the columns labelled $f_1(x), \ldots, f_4(x)$. Each IP that yields a *new* nondominated objective vector is marked with an asterisk $(*)$.

The final "relaxation" column shows the improvements that we gain with the new Algorithm 2: each entry in this column lists a previously-solved subproblem that allows us to avoid solving the current CLMOIP. For instance, the problem $(2, 10, 13)$ can be avoided by using the relaxation $(2, 10, \infty)$, and the problem $(1, \infty, 14, 12)$ can be avoided by using the relaxation $(1, \infty, 15, 12)$. Illustrating Remark 5 we cannot use $(2, \infty, \infty)$ to avoid solving $(2, \infty, 13)$ although it is a previously solved relaxation, since one of the solutions of $(2, \infty, \infty)$, namely $(11, 19, 12, 14)$, is not feasible for $(2, \infty, 13)$.

The results are extremely pleasing: by reusing problems that have already been solved, Algorithm 2 is able to generate the full nondominated set by solving only 40 IPs (shown by the 40 rows with no relaxation entry), in contrast to the 79 IPs required by Algorithm 1 (corresponding to all 79 rows of the tables).

6 Computational experiment

Przybylski et al. [2010a] perform computational experiments using 3-objective assignment problems and compare CPU times of various non-dominated set generation algorithms including Sylva and Crema [2004] and Laumanns et al. [2006]. Their results identify Laumanns et al. [2006] as the best general algorithm with other algorithms failing to generate the set, for even small problem instances, within a reasonable amount of time. As such, we compare an improved version of Laumanns et al. [2006] as discussed in Laumanns et al. [2005] with Özlen and Azizoğlu [2009] and the improved recursive algorithm using the 3-objective 3-dimensional Knapsack (3O3DKP) problem instances from Laumanns et al. [2006] in Table 4. Özlen and Azizoğlu [2009] is significantly faster compared to Laumanns et al. [2005] in solving 303DKPs and this behaviour becomes more significant with the growing problem size. The improved algorithm introduced in this paper is significantly faster compared to Özlen and Azizoğlu [2009] for all problem sizes while solving 3O3DKPs.

3-obj	2-obj	1-obj		$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	Relaxation
$(3,\infty)$	$(2,\infty,\infty)$	$(1,\infty,\infty,\infty)$	*	11	19	12	14	
		$(1,18,\infty,\infty)$	*	12	11	11	13	
		$(1,10,\infty,\infty)$	*	13	9	16	11	
		$(1,8,\infty,\infty)$	*	14	8	23	13	
		$(1,7,\infty,\infty)$		inf				
	$(2,22,\infty)$	$(1,\infty,22,\infty)$		11	19	12	14	$(1,\infty,\infty,\infty)$
		$(1,18,22,\infty)$		12	11	11	13	$(1,18,\infty,\infty)$
		$(1,10,22,\infty)$		13	9	16	11	$(1,10,\infty,\infty)$
		$(1,8,22,\infty)$		inf				
	$(2,15,\infty)$	$(1,\infty,15,\infty)$		11	19	12	14	$(1,\infty,\infty,\infty)$
		$(1,18,15,\infty)$		12	11	11	13	$(1,18,\infty,\infty)$
		$(1,10,15,\infty)$		inf				
	$(2,11,\infty)$	$(1,\infty,11,\infty)$		12	11	11	13	
		$(1,10,11,\infty)$		inf				$(1,10,15,\infty)$
	$(2,10,\infty)$	$(1,\infty,10,\infty)$	*	15	16	7	12	
		$(1,15,10,\infty)$	*	16	15	10	13	
		$(1,14,10,\infty)$		inf				
	$(2,9,\infty)$	$(1,\infty,9,\infty)$		15	16	7	12	$(1,\infty,10,\infty)$
		$(1,15,9,\infty)$		inf				
	$(2,6,\infty)$	$(1,\infty,6,\infty)$		inf				
$(3,13)$	$(2,\infty,13)$	$(1,\infty,\infty,13)$		12	11	11	13	
		$(1,10,\infty,13)$		13	9	16	11	$(1,10,\infty,\infty)$
		$(1,8,\infty,13)$		14	8	23	13	$(1,8,\infty,\infty)$
		$(1,7,\infty,13)$		inf				$(1,7,\infty,\infty)$
	$(2,22,13)$	$(1,\infty,22,13)$		12	11	11	13	$(1,\infty,\infty,13)$
		$(1,10,22,13)$		13	9	16	11	$(1,10,\infty,\infty)$
		$(1,8,22,13)$		inf				$(1,8,22,\infty)$
	$(2,15,13)$	$(1,\infty,15,13)$		12	11	11	13	$(1,\infty,\infty,13)$
		$(1,10,15,13)$		inf				$(1,10,15,\infty)$
	$(2,10,13)$	$(1,\infty,10,13)$		15	16	7	12	$(2,10,\infty)$
		$(1,15,10,13)$		16	15	10	13	$(2,10,\infty)$
		$(1,14,10,13)$		inf				$(2,10,\infty)$
	$(2,9,13)$	$(1,\infty,9,13)$		15	16	7	12	$(2,9,\infty)$
		$(1,15,9,13)$		inf				$(2,9,\infty)$
	$(2,6,13)$	$(1,\infty,6,13)$		inf				$(2,6,\infty)$
$(3,12)$	$(2,\infty,12)$	$(1,\infty,\infty,12)$		13	9	16	11	
		$(1,8,\infty,12)$		inf				
	$(2,15,12)$	$(1,\infty,15,12)$		15	16	7	12	
		$(1,15,15,12)$	*	17	13	15	11	
		$(1,12,15,12)$		inf				
	$(2,14,12)$	$(1,\infty,14,12)$		15	16	7	12	$(1,\infty,15,12)$
		$(1,15,14,12)$	*	19	15	14	11	
		$(1,14,14,12)$		inf				
	$(2,13,12)$	$(1,\infty,13,12)$		15	16	7	12	$(1,\infty,15,12)$
		$(1,15,13,12)$		inf				
	$(2,6,12)$	$(1,\infty,6,12)$		inf				$(2,6,\infty)$

Tab. 2: Iteration of Algorithm 1 and Algorithm 2 on a numerical example

3-obj	2-obj	1-obj		$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	Relaxation
$(3, 11)$	$(2, \infty, 11)$	$(1, \infty, \infty, 11)$		13	9	16	11	$(2, \infty, 12)$
		$(1, 8, \infty, 11)$		inf				$(2, \infty, 12)$
	$(2, 15, 11)$	$(1, \infty, 15, 11)$	*	15	17	11	10	
		$(1, 16, 15, 11)$		17	13	15	11	
		$(1, 12, 15, 11)$		inf				$(1, 12, 15, 12)$
	$(2, 14, 11)$	$(1, \infty, 14, 11)$		15	17	11	10	$(1, \infty, 15, 11)$
		$(1, 16, 14, 11)$	*	17	16	13	11	
		$(1, 15, 14, 11)$		19	15	14	11	
		$(1, 14, 14, 11)$		inf				$(1, 14, 14, 12)$
	$(2, 13, 11)$	$(1, \infty, 13, 11)$		15	17	11	10	$(1, \infty, 15, 11)$
		$(1, 16, 13, 11)$		17	16	13	11	$(1, 16, 15, 11)$
		$(1, 15, 13, 11)$		inf				$(1, 15, 13, 12)$
	$(2, 12, 11)$	$(1, \infty, 12, 11)$		15	17	11	10	$(1, \infty, 15, 11)$
		$(1, 16, 12, 11)$		inf				
	$(2, 10, 11)$	$(1, \infty, 10, 11)$		inf				
$(3, 10)$	$(2, \infty, 10)$	$(1, \infty, \infty, 10)$	*	13	19	17	10	
		$(1, 18, \infty, 10)$	*	14	11	16	9	
		$(1, 10, \infty, 10)$		inf				
	$(2, 16, 10)$	$(1, \infty, 16, 10)$		14	11	16	9	
		$(1, 10, 16, 10)$		inf				$(1, 10, \infty, 10)$
	$(2, 15, 10)$	$(1, \infty, 15, 10)$		15	17	11	10	$(1, \infty, 15, 11)$
		$(1, 16, 15, 10)$	*	18	15	15	9	
		$(1, 14, 15, 10)$		inf				
	$(2, 14, 10)$	$(1, \infty, 14, 10)$		15	17	11	10	$(1, \infty, 15, 11)$
		$(1, 16, 14, 10)$		inf				
	$(2, 10, 10)$	$(1, \infty, 10, 10)$		inf				$(2, 10, 11)$
$(3, 9)$	$(2, \infty, 9)$	$(1, \infty, \infty, 9)$		14	11	16	9	
		$(1, 10, \infty, 9)$		inf				$(1, 10, \infty, 10)$
	$(2, 15, 9)$	$(1, \infty, 15, 9)$	*	16	18	15	9	
		$(1, 17, 15, 9)$		18	15	15	9	
		$(1, 14, 15, 9)$		inf				$(1, 14, 15, 10)$
	$(2, 14, 9)$	$(1, \infty, 14, 9)$		inf				
$(3, 8)$	$(2, \infty, 8)$	$(1, \infty, \infty, 8)$		inf				

Tab. 3: Iteration of Algorithm 1 and Algorithm 2 on a numerical example (cont.)

n	$\|ND\|$	Laumanns et al. [2005]	Özlen and Azizoğlu [2009]		Improved Algorithm	
		CPU time (secs)	CPU time (secs)	# IP	CPU time (secs)	# IP
10	9	0.80	0.30	58	0.22	46
20	61	20.40	10.09	1119	3.99	333
30	195	391.24	117.93	4705	35.65	1204
40	389	1046.01	416.83	13628	84.27	2357
50	1048	7081.07	2044.65	38683	422.52	6001
100	6500	403937.83	82420.58	207515	21358.38	35450

Tab. 4: Comparison of Laumanns et al. [2005], Özlen and Azizoğlu [2009] and improved recursive algorithm on 3O3DKP

n	k	$\|ND\|$	Özlen and Azizoğlu [2009]		Improved Algorithm	
			CPU time (secs)	# IPs solved	CPU time (secs)	# IPs solved
$5x5$	3	12	0.12	152	0.05	71
	4	33	4.24	4332	0.63	704
	5	34	41.59	42229	2.53	3524
$10x10$	3	221	25.61	4062	8.61	1158
	4	736	2771.80	316448	214.98	16268
$15x15$	3	483	79.07	8367	25.25	2268
	4	7855	21075.83	1405070	2580.15	122986
$20x20$	3	1942	450.72	29710	163.66	9055
	4	22837	146813.61	5568823	11808.15	323703
$25x25$	3	3750	784.80	33803	401.14	15320
$30x30$	3	5195	1765.90	55272	819.73	22410
$35x35$	3	10498	4119.68	96161	2062.49	41828
$40x40$	3	14733	5204.42	111096	3066.30	55935
$45x45$	3	23942	12069.63	190405	6455.39	91780
$50x50$	3	29193	16464.46	215528	9108.42	109142

Tab. 5: Comparison of Özlen and Azizoğlu [2009] and improved recursive algorithm on MOAP

n	k	$\|ND\|$	Özlen and Azizoğlu [2009]		Improved Algorithm	
			CPU time (secs)	# IPs solved	CPU time (secs)	# IPs solved
5	3	8	2.31	75	1.51	45
	4	10	8.00	453	3.13	141
10	3	94	1006.27	3410	183.15	605
	4	561	281300.50	1052981	11984.49	39665
15	3	560	32841.38	41702	3557.41	3662

Tab. 6: Comparison of Özlen and Azizoğlu [2009] and improved recursive algorithm on MOTSP

We also perform experimentation using the 3-objective assignment problem (3OAP) instances from Przybylski et al. [2010a] and generate additional objectives using a similar distribution to theirs to test the performance of Özlen and Azizoğlu [2009] and the improved recursive algorithm on problems with more than 3 objectives (MOAP). The results are available in Table 5. The improved algorithm is faster compared to Özlen and Azizoğlu [2009] on MOAP. CPU time improvement becomes more significant with increasing number of objectives where the improved algorithm cuts the number of IPs solved quite effectively using the relaxations. For instance, the new algorithm improves the CPU time by 90% in solving the 4-objective 20 rows AP.

In order to see the performance of the improved algorithm on harder MOIP problems, we experiment using multi-objective travelling salesman (MOTSP) instances from Özpeynirci and Köksalan [2010]. We also generate some problems with higher number of objectives using their problem generator, the results are available in Table 6. The usage of relaxations bring great CPU time improvements for the MOTSP as well, and we see over 95% cut in the CPU time for the 4-objective 10-city TSP problem.

All computations are carried out on a single core of an Intel Core i7-2600 processor on a machine with 8 GB of RAM with hyper-threading and turbo-boasting disabled. GCC 4.7 is used to compile the code with default options, and CPLEX 12.5 with a single thread and default settings is used as the integer programming solver. A general C implementation of the improved recursive algorithm to solve problems, input in an extended LP file format, with arbitrary number of objectives is available at https://bitbucket.org/melihozlen/moip_aira/.

The large and diverse set of problems in terms of their difficulty that we use to compare the original and improved algorithms show that the new algorithm brings significant improvements in terms of the CPU time. The improvements become even larger when the number of objectives increases.

7 Conclusion

We propose a significant improvement on the recursive algorithm developed by Özlen and Azizoğlu [2009], based on the systematic reuse of solutions to relaxations of intermediate CLMOIP problems. Our numerical example and computational experiments show that the CPU time required by this new algorithm is significantly smaller than the requirement of the original algorithm, and moreover this improvement becomes more pronounced with increasing number of objectives.

One important feature of the improved recursive algorithm (which it inherits from the original) is that it may be used to generate only a subset of the nondominated objective vectors. This feature is important in cases where there are known restrictions on the individual objective function values, or where an explicitly known utility function is provided by the decision maker.

One area for future research is the development of domain-specific approaches to Multi Objective Combinatorial Optimisation (MOCO) problems that make similar improvements to avoid solving a large number of subproblems.

Another area might be to develop algorithms to solve MOIP which can take advantage of the highly parallel computing resources as most of the existing algorithms are limited to using a single core or thread.

Acknowledgements

We are thankful to anonymous reviewers for their constructive comments that helped us improve this paper substantially. Dr. Marco Laumanns kindly sent us the code and problem instances of Laumanns et al. [2005]. Dr. Özgur Özpeynirci kindly sent us their problem generator and instances from Özpeynirci and Köksalan [2010]. Dr. Anthony Przybylski kindly sent us their problem instances from Przybylski et al. [2010a]. The second author is supported by the Australian Research Council under the Discovery Projects funding scheme (project DP1094516).

References

M. Abbas and D. Chaabane. Optimizing a linear function over an integer efficient set. *European J. Oper. Res.*, 174(2):1140–1161, 2006.

H. P. Benson and E. Sun. Outcome space partition of the weight set in multiobjective linear programming. *J. Optim. Theory Appl.*, 105(1):17–36, 2000.

H. P. Benson and E. Sun. A weight set decomposition algorithm for finding all efficient extreme points in the outcome set of a multiple objective linear program. *European J. Oper. Res.*, 139(1):26–41, 2002.

B.A. Burton and M. Ozlen. Projective geometry and the outer approximation algorithm for multiobjective linear programming. arXiv:1006.3085, June 2010.

M. Ehrgott. *Multicriteria Optimization*. Springer, 2005.

J.M. Jorge. An algorithm for optimizing a linear function over an integer efficient set. *European J. Oper. Res.*, 195(1):98–103, 2009.

D. Klein and E. Hannan. An algorithm for the multiple objective integer linear programming problem. *European J. Oper. Res.*, 9(4):378–385, 1982.

M. Laumanns, L. Thiele, and E. Zitzler. An adaptive scheme to generate the pareto front based on the epsilon-constraint method. In J. Branke, K. Deb, K. Miettinen, and R. E. Steuer, editors, *Practical Approaches to Multi-Objective Optimization*, number 04461 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

M. Laumanns, L. Thiele, and E. Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European J. Oper. Res.*, 169(3):932–942, 2006.

M. Özlen and M. Azizoğlu. Multi-objective integer programming: a general approach for generating all non-dominated solutions. *European J. Oper. Res.*, 199(1):25–35, 2009.

M. Ozlen, M. Azizoğlu, and B. A. Burton. Optimising a nonlinear utility function in multi-objective integer programming. *Journal of Global Optimization*, pages 1–10, 2012. doi: 10.1007/s10898-012-9921-4.

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

A. Przybylski, X. Gandibleux, and M. Ehrgott. Computational results for four exact methods to solve the three-objective assignment problem. *Lecture Notes in Economics and Mathematical Systems*, 618: 79–88, 2009.

A. Przybylski, X. Gandibleux, and M. Ehrgott. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optim.*, 7(3): 149–165, 2010a.

A. Przybylski, X. Gandibleux, and M. Ehrgott. A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *INFORMS J. Comput.*, 22(3):371–386, 2010b.

J. Sylva and A. Crema. A method for finding the set of non-dominated vectors for multiple objective integer linear programs. *European J. Oper. Res.*, 158(1):46–55, 2004.

Melih Ozlen
School of Mathematical and Geospatial Sciences, RMIT University
GPO Box 2476V, Melbourne VIC 3001, Australia
(melih.ozlen@rmit.edu.au)

Benjamin A. Burton
School of Mathematics and Physics, The University of Queensland
Brisbane QLD 4072, Australia
(bab@maths.uq.edu.au)

Cameron A. G. MacRae
School of Mathematical and Geospatial Sciences, RMIT University
GPO Box 2476V, Melbourne VIC 3001, Australia
(cameron.macrae@rmit.edu.au)