# Disjunctive Programming for Multiobjective Discrete Optimisation

Tolga Bektaş

*Southampton Business School*
*and*
*Centre for Operational Research, Management Science and Information Systems (CORMSIS)*
*University of Southampton, Southampton SO17 1BJ, United Kingdom*
*e-mail: T.Bektas@soton.ac.uk*

**Abstract**

In this paper, I view and present the multiobjective discrete optimisation problem as a particular case of disjunctive programming where one seeks to identify efficient solutions from within a disjunction formed by a set of systems. The proposed approach lends itself to a simple yet effective iterative algorithm that is able to yield the set of all nondominated points, both supported and nonsupported, for a multiobjective discrete optimisation problem. Each iteration of the algorithm is a series of feasibility checks and requires only one formulation to be solved to optimality that has the same number of integer variables as that of the single objective formulation of the problem. The application of the algorithm show that it is particularly effective, and superior to the state-of-the-art, when solving constrained multiobjective discrete optimisation problem instances.

*Keywords.* multiobjective optimisation; disjunctive programming; integer programming; cutting plane.

## 1 Introduction

This paper is concerned with the solution a multiobjective discrete optimisation problem with $|K|$ objectives, for which a formulation can be given as follows:

$$\text{(MOP)} \qquad \text{Minimise} \qquad f(x) = (f_1(x), f_2(x), \dots f_{|K|}(x))$$
$$x \in X,$$

where $x$ is a vector of variables, $K$ is the index set of the objectives, $f(x)$ is a vector of conflicting objectives, element $k \in K$ of $f(x)$ corresponds to the objective function $f_k(x)$, and $X = \{Ax \leq b, x \in \mathbb{Z}\}$ is a nonempty set containing all feasible solutions. For a solution $x \in X$, the corresponding objective vector $f(x)$ is said to be a *point* in the objective space of the MOP. If there does not exist any $x' \in X$ such that $f_k(x') \leq f_k(x)$ for all $k \in K$ then $f(x)$ is said to be a *strictly nondominated* point and $x$ a *strictly efficient* solution. Similarly, if there does not exist any $x' \in X$ such that $f_k(x') < f_k(x)$ for all $k \in K$, then $f(x)$ is said to be a *weakly nondominated* point and $x$ a *weakly efficient* solution. If $x$ is an optimal solution of MOP$_\lambda$ : Minimise $\left\{\sum_{k=1}^{K} \lambda_k f_k(x) : x \in X\right\}$ for a given $\lambda = (\lambda_1, \ldots, \lambda_{|K|})$ with at least one positive element, then it is a *supported efficient solution*, otherwise it is said to be *nonsupported*.

Unlike single-objective integer programming, the solution of MOP is a set $X_E$ of efficient solutions. I assume that MOP does not admit any feasible solution that minimises all objectives simultaneously, and that the objectives are additive.

Exact algorithms to solve MOP in its general form were described as early as Bitran (1977) for the special case where $X = \{Ax \leq b, x \in \mathbb{B}\}$. Of more relevance to my work is the sequential algorithm proposed by Klein and Hannan (1982), and a variation thereof described by Sylva and Crema (2004). More recent algorithms include that of Özlen and Azizoğlu (2009) that is based on identifying objective efficiency ranges, a two-phase method described by Przybylski et al. (2000), an improvement to the method of Sylva and Crema (2004) proposed by Lokman and Köksalan (2013), an extension of the standard branch-and-cut to a multiobjective setting described by Jozefowiez et al. (2012) where special lower and upper bounding mechanisms are introduced, and, finally, a partitioning algorithm developed by Kirlik and Sayin (2014) that relies on searching the feasible space over $|K| - 1$ dimensional rectangles. The algorithms just mentioned are general in the sense that they can be used to solve MOP with any number of objectives and to generate the entire set of nondominated points. Extensive computational results presented by Kirlik and Sayin (2014) show that their algorithm is superior to the algorithms of Sylva and Crema (2004); Laumanns et al. (2006); Özlen and Azizoğlu (2009). Furthermore, Kirlik and Sayin (2014) provide results for MOP instances with up to five objectives, suggesting that their algorithm is state-of-the-art as far as solving MOP is concerned, in terms of both its speed and ability to identify set $X_E$.

Other algorithms have been described to only partially generate set $X_E$. In particular, the recursive algorithm proposed by Przybylski et al. (2010) generates all nondominated extreme points of MOP, which corresponds to a subset of the set of supported efficient solutions. Similarly, the exact algorithm of Özpeynirci and Köksalan (2013) finds all extreme supported nondominated points of multiobjective mixed integer programs. There also exist algorithms that are designed for the biobjective mixed integer programs, for example that of Stidsen et al. (2014) that is based on branch-and-bound, and those that are specifically designed to solve multiobjective versions of particular discrete optimisation problems, such as the knapsack and the assignment problem, which I will not review here, but instead will refer the reader to Ehrgott and Gandibleux (2000) and Ehrgott et al. (2016) for a review of the main properties, theoretical results and algorithms.

In this paper, I describe an iterative algorithm that is along similar lines of thought to that of Klein and Hannan (1982) in that a sequence of integer programming formulations are used to identify efficient solutions, and every efficient solution induces a set of systems to exclude the previously identified solutions from the search. However, the algorithm described here breaks away from all previous methods in that I model the sets in which efficient solutions exist (or otherwise) using a disjunction of systems, which allows the use of a so-called convex hull reformulation of the corresponding disjunctive program. This particular reformulation itself lends itself to a decomposition of the disjunction into its constituent systems. Each iteration of the algorithm is a series of feasibility checks on these systems, as is further discussed below.

## 2 Disjunctive Programming for MOP

Given a solution $x' \in X$ of MOP, I offer two questions that are of interest as far as solving MOP is concerned:

- Q1. Is $x'$ a (strictly) efficient solution of MOP? (If so, provide a certificate.)
- Q2. If $x'$ is an efficient solution of MOP, then is it the only one? (If not, provide a certificate).

These questions can be answered using disjunctive programming. To see this, consider the following disjunction defined over an index set $P$,

$$\bigvee_{p \in P} I^p_{x'}, \tag{2.1}$$

where each element $p \in P$ corresponds to a system $I^p_{x'} = \{f_k(x) \leq f^p_k(x'), \forall k \in K\}$, and where the subscript indicates that the system is induced by the solution $x'$. In a more general case, I will simply drop the subscript, in which case the system corresponding to the element $p \in P$ of a given disjunction will be shown as follows, where $r^p_k$ is the right hand side coefficient of the system corresponding to objective $k \in K$.

$$I^p = \{f_k(x) \leq r^p_k, \forall k \in K\}. \tag{2.2}$$

Coming back to the disjunction in (2.1), the $|P|$ systems therein are constructed in such a way that each one includes at least one objective with a finite bound, i.e., $\exists\, k \in K$ such that $f^p_k(x) \leq f_k(x') - \epsilon$ for each $p \in P$, where $\epsilon > 0$. Let $F(x') = \left\{ x \in X | \bigvee_{p \in P} (f_k(x) \leq f^p_k(x'), \forall k \in K) \right\}$, which denotes the set of feasible solutions defined by the disjunction (2.1). Similarly, let $F(I^p_{x'})$ denote the set of feasible solutions of the set $\{x \in X | f_k(x) \leq f^p_k(x'), \forall k \in K\}$.

I now return to the two questions above, with answers.

- A1. For the first question, it suffices to consider a special system $I^{p^*}_{x'} = \{f_k(x) \leq f^{p^*}_k(x') - \epsilon$, for all $k \in K\}$. If the corresponding set $F(I^{p^*}_{x'})$ of feasible solutions is empty, then $x'$ is a (strictly) efficient solution. Similar special systems can be constructed to verify as to whether $x'$ is a weakly efficient solution.
- A2. As for the second question, if $F(x') = \emptyset$, then this implies that $F(I^p_{x'}) = \emptyset$ for all $p \in P$, meaning that there is no other solution $x \in X$ that satisfies any of the systems $I^p_{x'}$, $p \in P$,

defining the conditions for $x$ to be an efficient solution. In this case, $x'$ is the only efficient solution. On the other hand, if $F(x') \neq \emptyset$, then there exists at least one other efficient solution $x'' \in X$, which satisfies at least one of the systems $I^p_{x'}$, i.e., $\exists p' \in P$ such that $x'' \in F(I^{p'}_{x'})$.

I will use the following example to illustrate the development of the approach.

**Example 1** *The following is a $3 \times 3$ tri-objective assignment problem instance from Przybylski et al. (2010), with the following cost matrices:*

$$C^1 = \begin{pmatrix} 6 & 3 & 12 \\ 13 & 17 & 10 \\ 9 & 14 & 16 \end{pmatrix} \qquad C^2 = \begin{pmatrix} 10 & 18 & 15 \\ 19 & 7 & 12 \\ 11 & 16 & 14 \end{pmatrix} \qquad C^3 = \begin{pmatrix} 12 & 8 & 7 \\ 19 & 18 & 15 \\ 2 & 10 & 0 \end{pmatrix}.$$

*Let $x = \{x_{ij}\}$ be a solution vector, where the variable $x_{ij}$ is equal to 1 if item $i \in \{1,2,3\}$ is assigned to $j \in \{1,2,3\}$, and 0 otherwise. The set of feasible solutions to the assignment problem is denoted by $X_A = (x_{ij} : \sum_{i=1}^{3} x_{ij} = 1 \text{ for } j \in \{1,2,3\}, \sum_{j=1}^{3} x_{ij} = 1 \text{ for } i \in \{1,2,3\}, x_{ij} \in \{0,1\})$. Consider now an efficient solution $x'$ provided by Przybylski et al. (2000) with all entries equal to 0 except for $x'_{11} = x'_{23} = x'_{32} = 1$, giving rise to the point $f(x') = (f_1(x'), f_2(x'), f_3(x')) = (30, 38, 37)$. Using this point, one can construct the following $2^3 - 1 = 7$ systems, where $\epsilon = 1$ as all three cost matrices have integer entries, and $M$ is a sufficiently large number so as to render the constraint in which it is used as unbinding.*

$$I^1_{x'} = \begin{pmatrix} f_1(x) \leq 29 \\ f_2(x) \leq M \\ f_3(x) \leq M \end{pmatrix} \qquad I^2_{x'} = \begin{pmatrix} f_1(x) \leq M \\ f_2(x) \leq 37 \\ f_3(x) \leq M \end{pmatrix} \qquad I^3_{x'} = \begin{pmatrix} f_1(x) \leq M \\ f_2(x) \leq M \\ f_3(x) \leq 36 \end{pmatrix}$$

$$I^4_{x'} = \begin{pmatrix} f_1(x) \leq 29 \\ f_2(x) \leq 37 \\ f_3(x) \leq M \end{pmatrix} \qquad I^5_{x'} = \begin{pmatrix} f_1(x) \leq M \\ f_2(x) \leq 37 \\ f_3(x) \leq 36 \end{pmatrix} \qquad I^6_{x'} = \begin{pmatrix} f_1(x) \leq 29 \\ f_2(x) \leq M \\ f_3(x) \leq 36 \end{pmatrix}$$

$$I^7_{x'} = \begin{pmatrix} f_1(x) \leq 29 \\ f_2(x) \leq 37 \\ f_3(x) \leq 36 \end{pmatrix}.$$

*For this instance, the set $F(I^7_{x'})$ of feasible solutions for system $I^7_{x'}$ is empty, which indicates that $x'$ is a (strictly) efficient solution. In addition, if there is at least one $p \in \{1, \ldots, 6\}$ for which $F(I^p_{x'}) \neq \emptyset$, then $x$ cannot be the only efficient solution. Indeed, consider another solution $x'' \in X_A$ also provided by Przybylski et al. (2010) with all entries equal to 0 except for $x''_{11} = x''_{22} = x''_{33} = 1$, giving rise to the point $f(x'') = (f_1(x''), f_2(x''), f_3(x'')) = (39, 31, 30)$ satisfying systems $I^2_{x'}$, $I^3_{x'}$ and $I^5_{x'}$, indicating that $F(x') \neq \emptyset$.* ∎

Indeed, one can continue in the fashion described above by considering more and more systems for each arbitrarily chosen solution $x \in X$ and search for nonempty subsets of feasible solutions to obtain certificates as to whether $x$ is efficient or whether there are others. This may be suitable for a constraint programming approach. However, I will not pursue such an approach in this paper due to two main drawbacks: (i) the lack of a method to identify a solution $x \in X$ to use at each iteration, and, more severely, (ii) the exponentially increasing size of the disjunction given that $2^{|K|} - 1$ systems would have be added for each $x \in X$. Instead, I describe an alternative approach below that overcomes these two drawbacks.

## 2.1 Integer linear programming

Consider the following formulation that incorporates a disjunction defined with respect to an index set $P$, where the $|K|$ objectives have been combined into a single objective function.

$$\text{MOP}(P) \qquad \text{Minimise} \sum_{k \in K} f_k(x) \text{ subject to } x \in X \cap \left\{ \bigcup_{p \in P} F(I^p) \right\}.$$

MOP($P$) is an augmented version of MOP$_\lambda$, where $\lambda_k = 1$ for all $k \in K$, by the disjunction $\bigvee_{p \in P} I^p$ formed by the systems $I^p$, $p \in P$. It is well known that an optimal solution $x^*$ of MOP($\emptyset$) is a *supported* efficient solution of one of the objectives for the weighted sum single-objective problem (Przybylski et al., 2010). In other words, at least one of the objectives will attain its minimal value at $x^*$. Formulation MOP($P$) then suggests, in its crude form, an iterative algorithm where one would start with MOP($\emptyset$), use a resulting optimal solution to construct a disjunction $P$, solve MOP($P$), a formulation that would effectively cut solution $x^*$ off, and which would either identify another efficient solution or return as infeasible indicating that no other efficient solution exists. This approach would address the first drawback described above.

In relation to the second drawback, I make the following observation. Each of the systems in the disjunction (2.1) plays a dual role by partitioning the search space. In particular, each system $I^p_{x'}$ either (i) returns an efficient solution (which I call a *certificate of efficiency*), or (ii) returns an infeasibility proving that that no efficient solution is contained in $F(I^p_{x'})$ (which I name *certificate of infeasibility*). For this reason, one simply cannot discard an infeasible system from a disjunction as otherwise the certificate of infeasibility will be lost. However, one can take advantage of formulation MOP($P$) to discard some of the systems without loosing information on the certificate of efficiency or infeasibility, as shown in the following proposition.

**Proposition 1** *Let $P$ be an index set of systems defining a disjunction and let $I^p$ and $I^q$ be two systems defined as (2.2) such that $p, q \in P$. If $r^q_k \leq r^p_k$ for all $k \in K$, then $I^q$ can be discarded.*

**Proof** First, I observe that $F(I^q) \subset F(I^p)$. There are two cases to consider:

1. If $F(I^p) = \emptyset$, then $F(I^q) = \emptyset$. In this case, one can remove the system $I^q$ from the disjunction without affecting the certificate of infeasibility.

2. If $F(I^p) \neq \emptyset$, then MOP($P$) will yield the optimal point $f(x)$ with a corresponding efficient solution $x$. I now show that $f(x)$ is also the optimal point of MOP($P \setminus \{q\}$) using the two sub-cases below:

   (a) $F(I^q) = \emptyset$, then $F(I^p \vee I^q) = F(I^p)$. Consequently, MOP($P$) = MOP($P \setminus \{q\}$), indicating that $f(x)$ is also optimal for MOP($P \setminus \{q\}$).

   (b) $F(I^q) \neq \emptyset$, then $x \in F(I^q) \subset F(I^p)$. In this case, $f(x)$ must be the optimal point of MOP($P \setminus \{q\}$) as otherwise there would have to be another point $f(\bar{x})$ with $\bar{x} \in F(I^p)$, $\bar{x} \neq x$ with at least one $k \in K$ such that $f_k(\bar{x}) < f_k(x)$, contradicting the optimality of point $f(x)$. ∎

The result of Proposition 1 suggests that, under the minimising objective function of MOP($P$), it suffices to use $|K|$ systems to construct a disjunction for a given efficient solution $x$, using the systems $I_x^k = \{f_k(x) \leq r_k^k, f_{k'}(x) \leq M, k' \in K \setminus \{k\}\}$, $\forall k \in K$, where $M$ is a sufficiently large value.

The development presented above suggests a sequential procedure to generate all efficient points for MOP, and is reminiscent of idea that has already been put forward, originally by Klein and Hannan (1982) and subsequently by Sylva and Crema (2004). However, the implementation is not straightforward. In Klein and Hannan (1982), the logical constraints (which correspond to the disjunctions here) are built within a branch-and-bound algorithm. In the work of Sylva and Crema (2004), an iterative procedure has been described where the disjunctions are modelled using the standard "big-M" constraints, requiring the addition of $|K|$ binary variables into MOP$_\lambda$ at each iteration, one for each disjunction. The number of additional variables and constraints then becomes prohibitively large and increases the difficulty of solving MOP$_\lambda$ to optimality, which was also empirically observed by Lokman and Köksalan (2013).

It is at this point where I break away from the direction of research that the two references above have pursued. In the following section, and in contrast to Sylva and Crema (2004), I will show that it is possible to embed the disjunctive constraints into MOP($P$) without the need to use additional binary variables. I will then describe an iterative algorithm where MOP($P$) will initially be constructed using a disjunction defined by an index set $P$ of systems, and $P$ will be iteratively expanded with each new efficient solution identified. This is achieved by using conjunctions of disjunctions and the convex hull representation of MOP($P$), both of which are explained below.

## 2.2 Intermingling disjunctions and conjunctions

Let $x'$ and $x'' \neq x'$ be two efficient solutions of MOP. There exists another efficient solution $x$ such that $x \neq x'$ and $x \neq x''$ if and only if $x \in F(x')$ and $x \in F(x'')$, or, alternatively, the following set is nonempty,

$$\left\{x \in X | \bigcup_{p \in P_1} F(I^p_{x'})\right\} \bigcap \left\{x \in X | \bigcup_{p \in P_2} F(I^p_{x''})\right\}, \tag{2.3}$$

where $P_1$ and $P_2$ are the index sets on which the two disjunctions are constructed using solutions $x'$ and $x''$, respectively. The set (2.3) of solutions correspond to the following conjunction,

$$Conj(I^p_{x'}, I^p_{x''}) = \left\{\bigvee_{p \in P_1} \Big(f_k(x) \leq f^p_k(x'), \forall k \in K\Big)\right\} \bigwedge \left\{\bigvee_{p \in P_2} \Big(f_k(x) \leq f^p_k(x''), \forall k \in K\Big)\right\}, \tag{2.4}$$

which, by using the well-known distributivity operator on disjunctions $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$, can be expressed in terms of the following expanded disjunction defined on an augmented index set $P$ of systems,

$$\left\{\bigvee_{p \in P} \Big((f_k(x) \leq f^p_k(x')) \wedge (f_k(x) \leq f^p_k(x'')), \forall k \in K\Big)\right\}, \tag{2.5}$$

where $P = P_1 \cup P_2$. It is easy to see that a pair of inequalities $f_k(x) \leq f^{p_1}_k(x')$ and $f_k(x) \leq f^{p_2}_k(x'')$, for a given $p_1 \in P_1$, $p_2 \in P_2$ and $k \in K$, under an "and" operator can be expressed as $f_k(x) \leq r^p_k = \min\{f^p_k(x'), f^p_k(x'')\}$, which can be used to rewrite (2.4) as follows:

$$Conj(I^p_{x'}, I^p_{x''}) = \left\{x \in X | \bigvee_{p \in P} \Big(f_k(x) \leq r^p_k, \forall k \in K\Big)\right\}. \tag{2.6}$$

**Example 2** *For the tri-objective assignment problem described in Example 1, consider the two efficient points $x' = (30, 38, 37)$ and $x'' = (39, 31, 30)$, each of which gives rise to the three sets of inequalities shown below.*

$$I^1_{x'} = \begin{pmatrix} f_1(x) \leq 29 \\ f_2(x) \leq M \\ f_3(x) \leq M \end{pmatrix} \qquad I^2_{x'} = \begin{pmatrix} f_1(x) \leq M \\ f_2(x) \leq 37 \\ f_3(x) \leq M \end{pmatrix} \qquad I^3_{x'} = \begin{pmatrix} f_1(x) \leq M \\ f_2(x) \leq M \\ f_3(x) \leq 36 \end{pmatrix}$$

$$I^1_{x''} = \begin{pmatrix} f_1(x) \leq 38 \\ f_2(x) \leq M \\ f_3(x) \leq M \end{pmatrix} \qquad I^2_{x''} = \begin{pmatrix} f_1(x) \leq M \\ f_2(x) \leq 30 \\ f_3(x) \leq M \end{pmatrix} \qquad I^3_{x''} = \begin{pmatrix} f_1(x) \leq M \\ f_2(x) \leq M \\ f_3(x) \leq 29 \end{pmatrix}$$

*The disjunction associated with solution $x'$ is $\bigvee_{p=1}^{3} I^p_{x'}$. Similarly, the disjunction associated with solution $x''$ is $\bigvee_{p=1}^{3} I^p_{x''}$. The conjunction of $I^1_{x'}$ with $I^1_{x''}$ results in $I^1_{x'}$, whereas the conjunction of $I^1_{x'}$ with $I^2_{x''}$ results in a new disjunction $p$ with $r^p_1 = 29$, $r^p_1 = 30$ and $r^p_3 = M$. Continuing in a similar way, the right hand side coefficients $r^p_k$ of the complete set of systems arising from the conjunction* *of the two disjunctions are obtained as below.*

| $(r_k^p)$ | $p=1$ | $p=2$ | $p=3$ | $p=4$ | $p=5$ | $p=6$ | $p=7$ | $p=8$ | $p=9$ |
|---|---|---|---|---|---|---|---|---|---|
| $k=1$ | 29 | 29 | 29 | 38 | $M$ | $M$ | 38 | $M$ | $M$ |
| $k=2$ | $M$ | 30 | $M$ | 37 | 30 | 37 | $M$ | 30 | $M$ |
| $k=3$ | $M$ | $M$ | 29 | $M$ | $M$ | 29 | 36 | 36 | 29 |

*By invoking the dominance criterion described in Proposition 1, one can reduce the nine systems shown above to the four below.*

| $(r_k^p)$ | $p=1$ | $p=2$ | $p=3$ | $p=4$ |
|---|---|---|---|---|
| $k=1$ | 29 | 38 | $M$ | 38 |
| $k=2$ | $M$ | 37 | 30 | $M$ |
| $k=3$ | $M$ | $M$ | $M$ | 36 |

*Indeed, the two remaining nondominated points reported by Przybylski et al. (2010) for this particular instance,* $(22, 41, 25)$ *and* $(38, 33, 27)$ *are feasible with respect to the four-system disjunction above, where the first satisfies either* $p = 1$ *or* $p = 4$*, and the second satisfies* $p = 2$ *or* $p = 4$*. In fact, this particular MOP instance can be solved using a total of nine systems in total to identify the four nondominated points, as opposed to the* $3^4 = 81$ *systems which would otherwise have been needed in the absence of Proposition 1.* $\blacksquare$

As the example above illustrates, even though the size of the disjunction increases exponentially by a factor of $|K|$ at each iteration, one can check the resulting set of systems in polynomial time by performing pairwise comparisons, to identify and subsequently discard any dominated system. I denote this procedure by $Dom(I)$ as applied to a given set $I$ of systems.

## 2.3 Convex hull reformulation of a disjunctive program

Balas (1998) shows that a disjunctive program defined over a disjunction of a set of systems indexed by $P$ can be modelled using $|P|$ additional variables by what is referred to as a *convex hull reformulation.* The convex hull reformulation $\mathrm{C}(P)$ of the model $\mathrm{MOP}(P)$ is given as follows:

$$\text{Minimise} \sum_{k \in K} \sum_{p \in P} f_k(x_p)$$

subject to

$$
\begin{aligned}
\sum_{p \in P} x_p &= x \\
\sum_{p \in P} y_p &= 1 \\
f_k(x_p) &\leq r_k^p y_p && \forall k \in K, p \in P \\
x_p &\leq u_p y_p && \forall p \in P \\
x &\in X \\
y &\in \{0,1\}^{|P|}. && (2.7)
\end{aligned}
$$

Here, I note that $C(\emptyset)$ is the same as $MOP(\emptyset)$. According to a result given by Balas (1998), an optimal solution of the formulation above, if exists, will always identify a $p^* \in P$ such that $y_{p^*} = 1$ and $y_p = 0$ for all $p \in P \setminus \{p^*\}$. In fact, this result implies that it suffices to solve $C(P)$ where the integrality restrictions (2.7) are relaxed as $y \in [0,1]$. Unfortunately, preliminary computational tests have suggested that even the relaxed version of $C(P)$ with a reduced number of systems can be challenging with modern solvers. However, I will not necessarily rely on this integrality property in the development of the ensuing algorithm, as explained in the following section.

## 2.4 An iterative algorithm

Using the result by Balas (1998), I decompose formulation $C(P)$ into a series of smaller subproblems, where each subproblem $C_{p^*}$ corresponds to a particular $p^* \in P$, where $y_{p^*} = 1$ and $y_p = 0$ for all $p \in P \setminus \{p^*\}$. In practice, one can project the $y_p$ variables out from each subproblem, yielding the following form of $C_p$:

$$\text{Minimise} \sum_{k \in K} f_k(x)$$

subject to

$$
\begin{aligned}
f_k(x) &\leq r_k^p && \forall k \in K && (2.8) \\
x &\in X.
\end{aligned}
$$

The iterative algorithm I propose starts with identifying an efficient solution by solving $C_p$ with no disjunctions, which I denote by $C_0$, and which is identical to $C_p$ without constraints (2.8). The efficient solution is then used to construct a disjunction to populate $C(P)$, which, when decomposed into a series of subproblems $C_p$, each subproblem will either provide a certificate of infeasibility, or return an efficient solution. The algorithm will iterate in this manner. There are three techniques I describe here to reduce the computational effort spent at each iteration:

1. For a given disjunction with $P \neq \emptyset$, one need not solve $\mathrm{C}_p$ for all $p \in P$; in fact it suffices to stop as soon as an efficient solution is identified (i.e., stop after the first feasible $\mathrm{C}_p$).

2. The conjunction of two systems $I^p$ and $I^q$ in a given iteration does not necessarily produce a new system. Assume without loss of generality that $Conj(I^p, I^q) = I^p$. The previous iteration will already have solved $\mathrm{C}_p$ and identified whether there exists a feasible solution or not. By building a *memory* feature to retain such information, spending additional computational time to test the feasibility of $\mathrm{C}_p$ in later iterations can be avoided.

3. Let $b_k = \text{Minimise}\,\{f_k(x) : x \in X\}$. If at any iteration, there exists a system $p \in P$ for which $r_k^p < b_k$ for any $k \in K$, then the corresponding system $p$ can be marked as being infeasible without requiring a further feasibility check. The calculation of $b_k$ for all $k \in K$ is done only once, and prior to the start of the algorithm.

A pseudocode of the proposed algorithm is given in Algorithm 1.

---

**Algorithm 1** An iterative algorithm to solve MOP

1: $I \leftarrow \emptyset$, $X_E \leftarrow \emptyset$, $P \leftarrow \{0\}$,
2: Label($p$) $\leftarrow$ feasible, for all $p \in P$
3: **repeat**
4:     Choose an unexamined element $p \in P$
5:     **if** Label($p$) $\neq$ infeasible **then**
6:         Solve $\mathrm{C}_p$
7:         **if** $\mathrm{C}_p$ is infeasible **then**
8:             Label($p$) $\leftarrow$ infeasible
9:         **if** $\mathrm{C}_p$ is feasible **then**
10:             Let $x'$ be an optimal solution of $\mathrm{C}_p$
11:             $X_E \leftarrow X_E \cup \{x'\}$
12:             $I' \leftarrow Conj(I, \bigvee_{k \in K} I_{x'}^k)$
13:             $I \leftarrow Dom(I')$
14:             Update the index set $P$ of the disjunction defined by set $I$
15:             Label($p$) $\leftarrow$ feasible, for all newly formed $p \in P$
16: **until** Label($p$) $=$ infeasible for all $p \in P$
17: Stop. $X_E$ is the set of efficient solutions.

---

The algorithm starts by initialising three sets, in particular a set $I$ of systems, a set $P$ of indices, one for each system defining a disjunction, and a set $X_E$ of efficient solutions. The algorithm then enters a loop between lines 3 and 16 to solve subproblems $C_p$, and exits the loop as soon as a feasible $C_p$ is found which yields an efficient solution $x'$. Any ordering of elements in set $P$ can be used for this purpose. The system $I_{x'}^k$ induced by this solution is then added to the set $I$. The algorithm maintains only a single disjunction at each iteration, comprising a set of systems, and one which is gradually enlarged in Steps 12 and 13. In particular, a conjunction operator is applied to the existing set of systems $I$ and the new system $I_{x'}^k$ in Step 12. In Step 13, the set $I'$ is checked to discard any dominated sets of inequalities. As explained above, $Dom(I')$ is the operator that performs pairwise checks for all systems using the dominance criterion in Proposition 1. The algorithm continues in this manner until the loop 3–16 fails to identify any feasible subproblem. It is at this point that the algorithm stops, indicating that there are no other efficient solutions and returns $X_E$ as the set of efficient solutions.

# 3 Computational Experiments

In this section, I present some computational experience with Algorithm 1 and comparison results. The algorithm is compared with that of Kirlik and Sayin (2014), available for public use at `http://home.ku.edu.tr/~gkirlik/research.html`, for the very reason that it is shown to outperform three other general purpose algorithms for MOP described by Sylva and Crema (2004); Laumanns et al. (2006); Özlen and Azizoğlu (2009). A common time limit is imposed for both algorithms, which is one hour for MOAP and MOKP instances, and three hours for the MOTSP.

Both algorithms are run on a laptop computer running on an 2.2 GHz Intel Core i7 with 16GB memory. Algorithm 1 has been coded in C. All subproblems within the two algorithms have been solved using CPLEX 12.6 through the use of the callable libraries. For Algorithm 1, I do not use the default parameter settings that come with CPLEX. In particular, the switch that controls the trade-offs between speed, feasibility, optimality, and moving bounds in solving mixed-integer programming formulations has been set to place emphasis on moving best bound (`CPX_PARAM_MIPEMPHASIS` set to 3). Furthermore, the presolve feature has been switched off by setting `CPX_PARAM_MIPEMPHASIS` to 0, and all automatic cuts are disabled by setting `CPX_PARAM_CUTSFACTOR` to 0, as I have found these features to slow down the detection of infeasibility in the subproblems. All other parameters remain at their default setting. The results are presented for three different types of MOP, namely the multiobjective assignment problem (MOAP), the multiobjective knapsack problem (MOKP) and the multiobjective travelling salesman problem (MOTSP), in the following sections.

## 3.1 Results on the MOAP

The MOAP instances tested here are those described in Kirlik and Sayin (2014) and are available at `http://home.ku.edu.tr/~gkirlik/research.html`. The size $n$ of the instances range from five to 15, where the number $|K|$ of objectives is either three or four. The objective function coefficients of these instances have been randomly drawn from the interval [1, 20]. Table 1 presents the results, where the figures shown on each line are averaged over 10 instances. For the two algorithms, the column titled "Time" shows the total time needed, in seconds, to identify the entire set of nondominated points. The results under the heading "Disjunctive Programming" pertain to those obtained by Algorithm 1, where the column titled "No. Sol." shows the average number of efficient solutions, and column titled "No. Disj." presents the total number of systems generated.

Table 1: A summary of comparison results for the MOAP instances

| | | Kirlik and Sayin (2014) | Disjunctive Programming | | |
|---|---|---|---|---|---|
| $\|K\|$ | $n$ | Time (s) | No. Sol. | No. Disj. | Time (s) |
| 3 | 5 | **0.08** | 14.10 | 26.30 | 0.30 |
| 3 | 10 | **7.44** | 176.80 | 268.50 | 8.18 |
| 3 | 15 | 64.44 | 674.90 | 967.60 | **55.99** |
| 4 | 5 | **0.53** | 34.00 | 123.70 | 1.57 |
| 4 | 10 | **199.95** | 895.20 | 2928.20 | 382.87 |

Table 1 shows that the disjunctive programming algorithm is competitive with the algorithm of Kirlik and Sayin (2014) for $|K| = 3$ in terms of the total time required, but is slower for $|K| = 4$. The main reason behind this is the number of efficient solutions that grows significantly as the size of the problem increases, which, in turn, requires the disjunctive programming algorithm to iterate for as many times as the number of efficient solutions of the instance.

### 3.2 Results on the MOKP

I now present results on MOKP instances, the sizes of which range from 20 to 40 items, and with three, four and five objectives. The instances for $|K| = 3$ or $|K| = 4$ are the same as those used in Kirlik and Sayin (2014), whereas those with $|K| = 5$ are generated in the same way in the latter reference as they are not made available. In particular, the weight and the profits of each item are randomly drawn integers from the interval [1, 1000], and the capacity of the knapsack is calculated as half of the total weight of all the items, rounded up where appropriate. The results are presented in a similar fashion as in Table 2. For sets that contain instances that could not be solved within the time limit, the average computational time has been calculated with respect to those instances for which the entire set of nondominated points has been found by both algorithms. These sets are $(|K| = 4, n = 40)$, $(|K| = 5, n = 20)$ and $(|K| = 5, n = 25)$, for which detailed results are given in Tables 3–5. In cases where the time limit is exceeded, the number of solutions and the number of systems reported are those obtained at the time of premature termination.

The results in Table 2 show that the proposed algorithm is dominated by that of Kirlik and Sayin (2014) in terms of total solution time for instances with $|K| = 3$. However, the situation is quite the opposite for when the number of objectives increases. In particular, the disjunctive programming algorithm shows a significant decrease in the time required to generate the set of efficient solutions for $|K| = 4$ and $|K| = 5$, and is able to solve more instances than Kirlik and Sayin (2014).

The results presented in this section for the MOKP suggest that the effectiveness of the disjunctive programming algorithm increases with the number of objectives, and when the size of the set of nondominated solutions is not so large. The stark contrast between the results reported for the

Table 2: A summary of comparison results for the MOKP instances

| | | Kirlik and Sayin (2014) | Disjunctive Programming | | |
|---|---|---|---|---|---|
| $\|K\|$ | $n$ | Time (s) | No. Sol. | No. Disj. | Time (s) |
| 3 | 30 | **3.70** | 115.80 | 231.90 | 8.83 |
| 3 | 40 | **16.05** | 311.40 | 617.80 | 47.45 |
| 3 | 50 | **27.93** | 444.20 | 876.00 | 84.15 |
| 4 | 20 | 23.60 | 136.80 | 659.30 | **17.89** |
| 4 | 30 | 441.52 | 397.60 | 1988.80 | **168.31** |
| 4 | 40 | 1085.50† | 676.25 | 3407.75 | **513.63** |
| 5 | 15 | 58.85 | 57.70 | 551.10 | **9.12** |
| 5 | 20 | 522.37‡ | 104.14 | 983.29 | **23.97** |
| 5 | 25 | 948.56§ | 137.50 | 1373.50 | **36.41** |

†Solved four instances out of 10.
‡Solved seven instances out of 10.
§Solved two instances out of 10.

Table 3: Results for MOKP instances with $|K| = 4$ and $n = 40$

| | Kirlik and Sayin (2014) | Disjunctive Programming | | |
|---|---|---|---|---|
| Instance number | Time (s) | No. Sol. | No. Disj. | Time (s) |
| 1 | 1766.00 | 901 | 4516 | **906.28** |
| 2 | 3600 | 1427 | 7949 | 3600 |
| 3 | 352.52 | 508 | 2349 | **213.3** |
| 4 | 3600 | 1248 | 6139 | **2139.9** |
| 5 | 3600 | 1391 | 7945 | 3600 |
| 6 | 3600 | 1357 | 7944 | 3600 |
| 7 | 3600 | 1390 | 7769 | 3600 |
| 8 | 1671.44 | 741 | 3981 | **640.96** |
| 9 | 3600 | 1486 | 7683 | 3600 |
| 10 | 552.02 | 555 | 2785 | **293.96** |
| Total number of instances solved | 4/10 | 5/10 | | |

Table 4: Results for MOKP instances with $|K| = 5$ and $n = 20$

| | Kirlik and Sayin (2014) | Disjunctive Programming | | |
|---|---|---|---|---|
| Instance number | Time (s) | No. Sol. | No. Disj. | Time (s) |
| 1 | 3600 | 220 | 2277 | **98.68** |
| 2 | 134.59 | 95 | 801 | **15.12** |
| 3 | 47.26 | 76 | 603 | **9.62** |
| 4 | 127.41 | 89 | 1007 | **18.04** |
| 5 | 456.34 | 110 | 1018 | **19.06** |
| 6 | 71.15 | 87 | 706 | **11.72** |
| 7 | 23.61 | 61 | 531 | **8.1** |
| 8 | 2796.22 | 211 | 2217 | **86.16** |
| 9 | 3600 | 237 | 3504 | **177.59** |
| 10 | 3600 | 426 | 5993 | **889.9** |
| Total number of instances solved | 7/10 | 10/10 | | |

Table 5: Results for MOKP instances with $|K| = 5$ and $n = 25$

| | Kirlik and Sayin (2014) | Disjunctive Programming | | |
|---|---|---|---|---|
| Instance number | Time (s) | No. Sol. | No. Disj. | Time (s) |
| 1 | 3600 | 679 | 9958 | 3600 |
| 2 | 3600 | 687 | 5482 | 3600 |
| 3 | 3600 | 540 | 10690 | 3600 |
| 4 | 1624.33 | 172 | 1814 | **55.03** |
| 5 | 3600 | 309 | 3469 | **262.01** |
| 6 | 3600 | 612 | 8711 | 3600 |
| 7 | 3600 | 448 | 5568 | **920.71** |
| 8 | 272.78 | 103 | 933 | **17.78** |
| 9 | 3600 | 689 | 9607 | 3600 |
| 10 | 3600 | 452 | 5881 | **985.93** |
| Total number of instances solved | 2/10 | 5/10 | | |

MOKP and the MOAP suggest that the algorithm works much better on constrained problems. I will provide further evidence on this in the following section.

### 3.3 Results on the MOTSP

The choice of this particular multiobjective problem is deliberate, as it is a constrained version of MOAP, and where the aim is to see the effect of further constraining the set of feasible solutions and therefore the set of nondominated points on the performance of the algorithm. Consequently, the model used to solve the MOTSP is a restricted version of the MOAP, in that it is an assignment based formulation augmented with a set of subtour breaking constraints in the spirit of Gavish and Graves (1978). The MOTSP instances tested here have three objectives, with sizes ranging from 10 to 20, for which the costs have been generated in the same way as the MOAP instances. Whilst the sizes of the instances tested may seem small, they are comparable with those in Özpeynirci and Köksalan (2013), particularly as the latter reference describes an algorithm that identifies only a subset of the set of efficient solutions. The results are presented in Table 6.

Table 6: A summary of comparison results for the MOTSP instances

| | | Kirlik and Sayin (2014) | Disjunctive Programming | | |
|---|---|---|---|---|---|
| $\|K\|$ | $n$ | Time (s) | No. Sol. | No. Disj. | Time (s) |
| 3 | 10 | 56.08 | 126.00 | 205.00 | **14.46** |
| 3 | 15 | 1162.49 | 567.20 | 836.20 | **161.38** |
| 3 | 20 | 4125.90 | 1292.60 | 1805.10 | **900.20** |

The results shown in Table 6 show a clear-cut superiority of the disjunctive programming algorithm in terms of the computational time required. The reduction in the average number of solutions from MOAP to MOTSP is evident when the results are compared with those presented in Table 1, which is a factor that contributes to the efficiency of the proposed algorithm. In addition to the three-objective instances, I have also solved 10 instances of a four-objective TSP with $n = 10$. For these instances, the average computational time required by the algorithm of Kirlik and Sayin (2014) was 995.16 seconds, whereas the same figure for Algorithm 1 was 309.80 seconds. Both algorithms solved all 10 instances. The average number of efficient solutions was 636.70.

## 4 Identifying a Well-Dispersed Subset of NonDominated Solutions

A relevant question for multiobjective discrete optimisation, particularly when the size of the non-dominated set of points is undesirably large, is to find a well-dispersed subset of such points. The first phase of the two phase method, to a certain extent, addresses this question, but it is not straightforward to extend this method to problems with three or more objectives (Przybylski et al., 2000). In this section, I show that this can be done in a relatively simple way using disjunctive programming, through a judicious selection of the systems contained within a disjunction during the course of the iterative algorithm.

For a given MOP, let $P$ be a nonempty set of indices of systems forming a disjunction and $X_E$ a set of efficient solutions already identified. The question of finding a well-dispersed subset can be rephrased as finding a system $p \in P$ such that $x^* \in F(I^p)$ maximises a given distance metric between $x^*$ and all other $x' \in X_E$. For the purposes of this paper, I will use the following metric:

$$D(x^*, x') = \sum_{k \in K} \left( f_k(x^*) - f_k(x') \right)^2. \tag{4.1}$$

The above question is now tantamount to finding a $x^* \in X = \underset{x \in F(I^p), p \in P}{\operatorname{argmax}} \sum_{x' \in X_E} D(x, x')$. In this section, I will additionally assume that $f_k(x) \geq 0$ for any $x \in X$ for all $k \in K$. Consider, now, a $x \in F(I^p)$ for a given $p \in P$, for which the total distance from all other solutions in $X_E$, by using the definition (4.1), can be calculated as follows:

$$\begin{aligned} \sum_{x' \in X_E} D(x, x') &= \sum_{x' \in X_E} \left( \sum_{k \in K} \left( f_k(x) - f_k(x') \right)^2 \right) \\ &\leq \sum_{x' \in X_E} \sum_{k \in K} (f_k(x))^2 + \sum_{x' \in X_E} \sum_{k \in K} (f_k(x'))^2 \\ &\leq \sum_{x' \in X_E} \sum_{k \in K} (r_k^p)^2 + \sum_{x' \in X_E} \sum_{k \in K} (f_k(x'))^2. \end{aligned} \tag{4.2}$$

As the last component of (4.2) is a constant for a given set $X_E$, an upper bound on the maximum distance is given by the first component, which implies that choosing a system $p^* \in P$ satisfying the following condition,

$$p^* = \underset{p \in P}{\operatorname{argmax}} \sum_{k \in K} r_k^p, \tag{4.3}$$

is the one most likely to yield a solution $x^*$ that has the largest cumulative distance from all other solutions $x' \in X$. This observation requires searching through all the systems in $P$ to identify the one satisfying the condition (4.3), which is not impossible. However, a more practical approach would be to limit the search from within the set of systems to those having the least amount of finite bounds imposed across the $|K|$ objective functions (i.e., those with the largest number of "big-M" right hand side coefficients). For a three-objective MOP, for example, one can discard all systems with two or more finite bounds on the individual objective function components.

To illustrate the outcome of the proposed strategy, I consider two tri-objective TSP instances, one with 15 for which the results are shown in Figures 1 and 2, and the other with 20 nodes for which the results are presented in Figures 3 and 4.

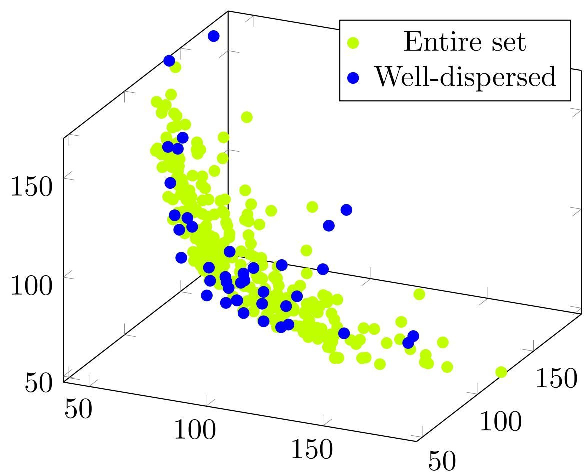


Figure 1: Well-dispersed subset of nondominated points for the tri-objective 15-node TSP instance

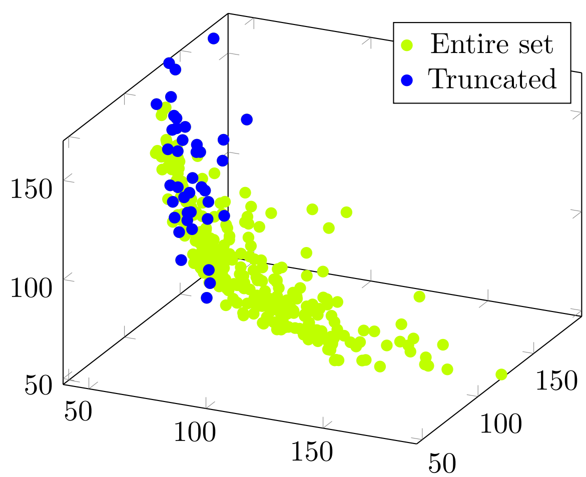


Figure 2: Truncated subset of nondominated points for the tri-objective 15-node TSP instance

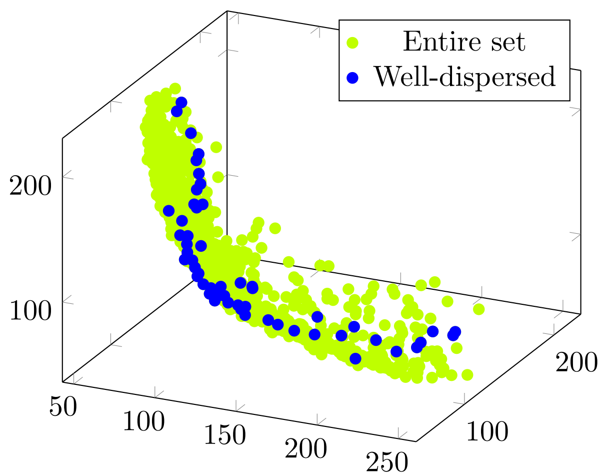


Figure 3: Well-dispersed subset of nondominated points for the tri-objective 20-node TSP instance

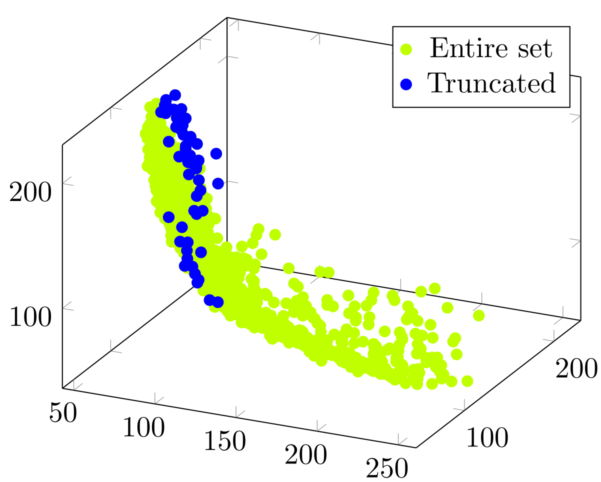


Figure 4: Truncated subset of nondominated points for the tri-objective 20-node TSP instance

Figure 1 shows a well-dispersed subset of 40 nondominated points identified through the strategy proposed above, against the entire set of 335 nondominated points of the 15-node MOTSP instance. Similarly, Figure 3 shows a well-dispersed subset of 53 nondominated points obtained with the proposed strategy for the 20-node instance, against the entire set of 1013 nondominated points. These points are contrasted with those obtained using a truncated version of Algorithm 1, terminated after finding the first 40 nondominated points for the 15-node instance, and the first 53 points for the 20-node instance, which are shown in Figures 2 and 4. There is clear indication from these figures to suggest that the simple strategy described above suffices to generate a representative sample of the set of nondominated solutions for these instances.

# 5 Conclusions

The iterative algorithm described in this paper can be applied to any multiobjective discrete optimisation problem, with any number of objectives, to generate the entire set of nondominated points, provided that the underlying subproblems can either be solved, or checked for infeasibility, using an optimiser. The algorithm is particularly effective in finding nondominated points when the size of the set of efficient solutions is relatively small. It does not seem to suffer from the increase in the number of objectives in the way as some of the other state-of-the-art methods do, such as the two-phase method. In the case that a limited subset of a possibly large set of efficient solutions is sought, the algorithm can also provide a well-dispersed subset of nondominated points by looking at a specially selected subset of systems defining a disjunction.


## Acknowledgements

I thank Gokhan Kirlik for helpful discussions and for providing guidance on the use of his code and Nicolas Jozefowiez for raising a question that resulted in the exposition given under Section 4.


# References


Balas, E. 1998. Disjunctive programming: Properties of the convex hull of feasible points. Discrete Applied Mathematics, 89, 3–44.

Bitran, E. 1977. Linear multiple objective programs with zero-one variables. Mathematical Programming, 13, 121–139.

Ehrgott, M. and Gandibleux, X. 2000. A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spektrum 22, 425–450.

Ehrgott, M., Gandibleux, X. and Przybylski, A. 2016. Exact methods for multi-objective combinatorial optimisation. In: Multiple criteria decision analysis (Greco, S., Ehrgott, M., Figueira, J.R., eds.), International Series in Operations Research & Management Science, Volume 233. Springer, US, pp. 817–850.

Gavish, B. and Graves, S.C. *The traveling salesman problem and related problems.* Working Paper OR-078-78, Operations Research Center, MIT, Cambridge, MA .

Jozefowiez, N., Laporte, G., and Semet, F. 2012. A generic branch-and-cut algorithm for multiobjective optimization problems: Application to the multilabel traveling salesman problem. INFORMS Journal on Computing 24, 554–564.

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

Klein, D. and Hannan, E. 1982. An algorithm for the multiple objective integer linear programming problem. European Journal of Operational Research 9, 378–385.

Laumanns, M, Thiele, L. and Zitzler, E. 2006. An efficient, adaptive parameter variation scheme for generating all non-dominated solutions. European Journal of Operational Research 199, 25–35.

Lokman, B. and Köksalan, M. 2013. Finding all nondominated points of multi-objective integer programs. Journal of Global Optimization 57, 347–365.

Özlen, M. and Azizoğlu, M. 2009. Multi-objective integer programming: A general approach for generating all non-dominated solutions. European Journal of Operational Research 199, 25–35.

Özpeynirci, Ö. and Köksalan, M. 2013. An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. Management Science 56, 2302–2315.

Przybylski, A., Gandibleux, X. and Ehrgott, M. 2000. A two phase method for multi-objective integer programming. Discrete Optimization 7, 149–165.

Przybylski, A., Gandibleux, X. and Ehrgott, M. 2010. A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. INFORMS Journal on Computing 22, 371–386.

Stidsen, T., Andersen, K. A. and Dammann, B. 2014. A branch and bound algorithm for a class of biobjective mixed integer programs. Management Science 60, 1009–1032.

Sylva, J. and Crema, A. 2004. A method for finding the set of non-dominated vectors for multiple objective integer linear programs. European Journal of Operational Research 158, 46–55.