Chapter 2
Mathematical Models for Aggregate Planning

2.1 Modeling an Aggregate Plan

Aggregate planning can be developed based on different strategies as shown in Table 1.1. In many cases, a pure strategy does not result in a good or an optimal solution since real-world problems are not expressed by extreme conditions. For instance, the demand rate is not constant, hiring and firing decisions are followed up by many indirect effects, and a large number of issues influence the organization. Hence, we look for a mixed strategy for meeting demand and considering the trade-offs in various situations. An aggregate plan considers the trade-offs among capacity, inventory, and backlog costs. An increase in each cost results in reduction in the other costs. Finding the most profitable combination is the goal of aggregate planning [2]. The best strategy is a particular trade-off among capital investment, workforce size, working hours, inventory, and backlogs/lost sales [2].

The optimal mixed strategy or the best trade-off is typically found by considering all conditions, demand fluctuations, several parameters, constraints, and decisions. Therefore, solving a linear program (LP) which formulates aggregate planning problems, is a strong tool for finding the optimal solution(s). Next, we present the fundamental parameters and decision variables for formulating a basic aggregate planning problem.

2.1.1 *Basic Assumptions*

The cost function for hiring/firing workers is assumed to be linear. This assumption is realistic since we calculate hiring/firing costs based on the number of hired or fired workers. However, this may be unreasonable if the number of potential workers that can be hired is small [3]. Moreover, we assume that the average performance

S. A. B. Rasmi, M. Türkay, *Aggregate Planning*, SpringerBriefs in Operations Research, https://doi.org/10.1007/978-3-030-58118-3_2

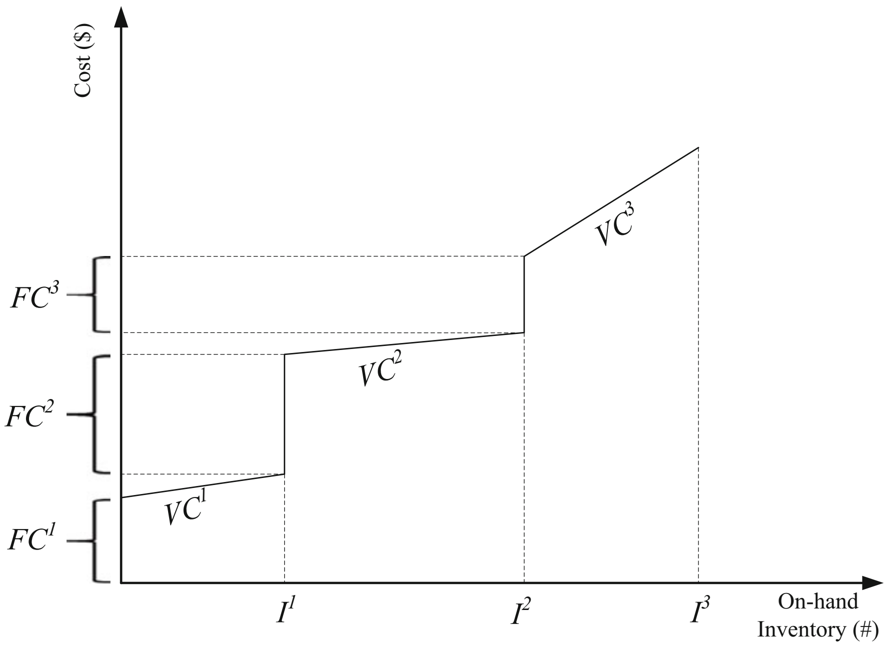

Fig. 2.1 An example for partially linear inventory holding cost function

of a newly hired worker is less than usual. In the first period or month, some training programs are provided for new workers; so, a lower performance is a logical outcome.

Inventory holding cost is another cost in aggregate planning models which is assumed to be linear. We assume that holding cost in a period is a linear function of the number of products [3]. There exist other inventory holding cost functions which are more compatible with the facts. However, they increase the complexity of the problem in terms of both modeling and the solution method. For instance, assume that a factory holds its products in a depot which has a specific capacity. This depot has a fixed cost for the factory and products are held by paying a variable cost per single product. If we require to hold more inventory than the capacity of this depot, then a second depot is required. Figure 2.1 shows an instance for such a cost function. If the factory requires to hold I^2 products in inventory, the variable cost associated with I^1 products is VC^1 at depot 1, and the fixed cost is FC^1. Moreover, $I^2 - I^1$ products are held by paying fixed cost FC^2 and variable cost VC^2 at depot 2.

The costs of regular working time, overtime, and shortage are also linear. These assumptions are very helpful to keep mathematical models easy and straightforward for the practitioners. Moreover, the solution approach will contain a lower complexity since the models are linear.

In addition, we assume that the salary of one worker in one period is independent from the number of days that s/he is working during that period. However, workers are paid additionally due to the overtime working hours. We assume that if overtime is required in a period, then the total overtime is divided by all workers of that period.

2.1.2 Parameters and Decision Variables

We define the basic parameters and the decision variables which are used in the basic aggregate planning model in the next subsection. Let T be the set of periods in the planning horizon, and t as an index denotes the tth period in our mathematical models. Then, assume that $t \in T$ and $T = \{1, 2, ..., \tau\}$. The parameters are as follows:

d_t : forecast of demand in period t.
pr_t : selling price of one unit in period t.
c_H : cost of hiring one worker.
c_F : cost of firing one worker.
c_I : holding cost of one unit in inventory for one period.
c_P : production cost of one unit.
c_{SU} : cost to subcontract one unit of production.
l_{SU} : maximum available production capacity by subcontracting in each period.
c_{SO} : cost of one unit stocked out in one period.
s_W : salary of one worker in one period.
os_W : salary of one experienced worker per 1 h overtime. Note that if a worker works with a smaller performance, then s/he is paid according to that performance during working hours.
n_t : number of production days (working days) in period t.
h: number of regular working hours in 1 day.
l_O : maximum allowed overtime per worker in one period.
k_R : number of aggregate units produced by one experienced worker in 1 h of regular time.
k_O : number of aggregate units produced by one experienced worker in 1 h of overtime. Note that $k_O \leq k_R$ logically.
ap: average performance of a newly hired worker in the first period ($0 \leq ap \leq 1$).

We also define the required decision variables for maximizing the total profit. In many studies, aggregate planning problems are modeled such that the objective function minimizes the total cost. However, we focus on the total profit. The decision variables are as follows:

W_t : workforce size in period t, $t \in T$. Moreover, W_0 is a parameter to define the initial workforce level at the start of the planning horizon.
O_t : total overtime hours in period t, $t \in T$.
H_t : number of workers hired at the beginning of period t, $t \in T$.
L_t : number of workers laid off at the beginning of period t, $t \in T$.
P_t : production level in period t, $t \in T$.

SU_t : number of produced aggregate units by subcontracting in period t, $t \in T$.
I_t : inventory level at the end of period t, $t \in T$. I_0 is the initial inventory level at the start of the planning horizon (I_0 is a parameter).
S_t : number of units sold in period t, $t \in T$.
SO_t : number of units stocked out at the end of period t, $t \in T$.

2.1.3 Basic Model

We model the objective function as the total profit. We assume that *Sales Revenue* is the only income of the production system as:

$$\text{Sales Revenue} = \sum_{t \in T} pr_t S_t.$$

This expression is equal to the sum of sold units multiplied by the price of one aggregate unit.

We continue with the other statements of the objective function given in (2.1a); next, the cost of the system. *Production Cost* refers to the cost of production within the manufacturer's or subcontracting activities for all units as follows:

$$\text{Production Cost} = \sum_{t \in T} c_P P_t + \sum_{t \in T} c_{SU} SU_t.$$

Cost of Regular time shows the total salary paid to the workers in each period for regular time working hours. Moreover, *Overtime Cost* indicates the total salary paid to the workers in the planning horizon for overtime.

$$\text{Cost of Regular time} = \sum_{t \in T} s_W W_t,$$

$$\text{Overtime Cost} = \sum_{t \in T} os_W O_t.$$

Hiring/Firing Costs represent the total cost of hiring and firing workers in the planning horizon as:

$$\text{Hiring/Firing Costs} = \sum_{t \in T} c_H H_t + \sum_{t \in T} c_F L_t.$$

Term *Inventory holding/stock out costs* indicate the total cost for holding inventory at the end of each period in the planning horizon and the costs associated with lost sales.

$$\text{Inventory holding/stock out costs} = \sum_{t \in T} c_I I_t + \sum_{t \in T} c_{SO} SO_t.$$

Next, we present the constraints of the basic model as follows:

$$W_t = W_{t-1} + H_t - L_t, \forall t \in T,$$

This is the balance constraint for workforce size; the number of workers in a period is equal to the number of workers in the previous period plus/minus the number of hired/fired workers. We balance the inventory level using

$$I_t = I_{t-1} + P_t + SU_t - S_t, \forall t \in T.$$

This constraint shows that the inventory level at the end of a period is equal to the inventory level at the end of the previous period plus/minus the number of produced/sold units. Note that a manufacturer may produce an item by outsourcing (subcontracting). Moreover, we do not desire to have stock-out at the end of planning horizon, and hence, $I_{12} = I_0$ (or $I_{|T|} = I_0$) may be added to the set of constraints. Note that it is not logical to stop all activities at the end of the planning horizon; the manufacturer may continue to work under a (different) condition.

We know that both the number of sold and stocked out units are smaller than or equal to the demand in each period. Then, the following constraint satisfies this condition.

$$S_t + SO_t = d_t, \forall t \in T$$

Assume that we are at period t. Then, the total available regular time for production is $((W_t - H_t) + apH_t)n_t h$ where $W_t - H_t$ and H_t are the number of experienced and newly hired workers, respectively. Hence, the production capacity regarding both regular time and overtime in each period is computed by

$$P_t \leq (W_t - (1 - ap)H_t)n_t hk_R + O_t k_O, \forall t \in T.$$

In addition, we restrict the total overtime and the number of units subcontracted using

$$O_t \leq l_O(W_t - (1 - ap)H_t), \forall t \in T$$

and

$$SU_t \leq l_{SU}, \forall t \in T.$$

Using the explained terms, we form model BM1 given in (2.1) for a basic aggregate planning problem.

BM1

$$\max z = \sum_{t\in T} pr_t S_t - \sum_{t\in T} c_P P_t - \sum_{t\in T} c_{SU} SU_t - \sum_{t\in T} s_W W_t - \sum_{t\in T} os_W O_t$$
$$- \sum_{t\in T} c_H H_t - \sum_{t\in T} c_F L_t - \sum_{t\in T} c_I I_t - \sum_{t\in T} c_{SO} SO_t \tag{2.1a}$$

$$\text{s.t.} \quad W_t = W_{t-1} + H_t - L_t, \forall t \in T, \tag{2.1b}$$

$$I_t = I_{t-1} + P_t + SU_t - S_t, \ I_{|T|} = I_0, \forall t \in T, \tag{2.1c}$$

$$S_t + SO_t = d_t, \forall t \in T, \tag{2.1d}$$

$$P_t \leq (W_t - (1 - ap)H_t)n_t h k_R + O_t k_O, \forall t \in T, \tag{2.1e}$$

$$O_t \leq l_O(W_t - (1 - ap)H_t), \forall t \in T, \tag{2.1f}$$

$$SU_t \leq l_{SU}, \forall t \in T, \tag{2.1g}$$

$$W_t, H_t, L_t, P_t, SU_t, I_t, S_t, SO_t, O_t \geq 0, \tag{2.1h}$$

where (2.1a) shows the objective function and (2.1b)–(2.1g) define the constraints of the model. Equation (2.1h) displays the nonnegativity constraints. Note that W_t, H_t, L_t, P_t, SU_t, I_t, S_t, and SO_t are theoretically nonnegative integer values ($\in \mathbb{Z}^+ \cup \{0\}$). Due to the high complexity of solving integer and mixed-integer linear programming problems, we assume that these decision variables are continuous. Moreover, if we round the solution to the integer values, the result will be highly reliable in terms of feasibility and optimality.

Regarding the number of hired and fired workers in each period, it does not make sense to have a positive number for both of them. As a result we would need to add this constraint, $H_t F_t = 0$, $\forall t \in T$. However, since the optimal solution of linear programs occurs only on an extreme points, adding that constraint is not required [3].

2.2 Modeling Aggregate Plans with Additional Practical Modifications

We have presented a basic model for aggregate planning problem in Sect. 2.1.3. That model and approach includes fundamental assumptions and considerations for an aggregate plan. However, decision makers may need to consider more practical assumptions and remodel the problem. We establish model BM2 given in (2.2) to add values to BM1 and improve its practicality.

2.2.1 Aggregate Plan with Production Setup Cost, Backlog, and Promotion Programs

A fixed cost must be considered when the manufacturer produces at least one aggregate unit in a time period. Moreover, in case of occurring stock-out, the sale will be backlogged. Thus, the firm satisfies the backorders in the next period. Note that backlogging results in cost in the objective function. In addition, we assume that a number of promotion strategies (let n promotion programs exist) are available for the company. Then, at most one of these strategies can be selected by the company, and they result in cost. They also result in an increase in demand. In order to satisfy these assumptions, we define some new parameters and decision variables. The following parameters are added to the aggregate planning problem:

c_U : fixed cost of production in each period.
c_B : backlogging cost per unit in one period.
c_ρ^i: cost of using promotion program i, $i \in \{1, ..., n\}$.
d_{it}^+: demand increase rate in period t if promotion program i is used, $i \in \{1, ..., n\}$ and $t \in T$.
M: a sufficiently large positive number (big M).

The following decision variables are also used in the model:

U_t : is a binary decision variable. It is equal to 1 if the company produces in period t. Otherwise, it is equal to zero ($t \in T$).
ρ_i : is a binary decision variable. It is equal to 1 if promotion program i is used. Otherwise, it is equal to zero ($i \in \{1, ..., n\}$).
D_t : actual demand in period t, $t \in T$. Note that this decision variable will be computed by d_t, d_{it}^+, and ρ_i (see (2.2h)).

In problem BM2, I_t, the inventory level at the end of period t, will be unrestricted and takes a real value (see constraint (2.2n)). When $I_t > 0$, we have some inventory on hand. If it is negative, then backlogging has occurred. Hence, we define two new nonnegative decision variables to track on-hand and backlog inventory levels:

I_t^+: on-hand inventory level at the end of period t where $t \in T$.

I_t^-: the number of backorders at the end of period t, $t \in T$ ($I_t = I_t^+ - I_t^-$, see constraint (2.2j)). Note that in some cases, I_t^+ and I_t^- can accept positive values in the same time because a manufacturer may prefer to hold inventory, accept backlogging, and sale the product later because of its larger price.

In problem BM2 given in (2.2), we formulate an aggregate plan with a setup cost for production, backorders, and n available promotion programs.

$$\begin{aligned}
&\text{BM2} \\
&\max z = \sum_{t\in T} pr_t S_t - \sum_{t\in T} c_P P_t - \sum_{t\in T} c_{SU} SU_t - \sum_{t\in T} s_W W_t \\
&\qquad - \sum_{t\in T} os_W O_t - \sum_{t\in T} c_H H_t - \sum_{t\in T} c_F L_t - \sum_{t\in T} c_I I_t^+ \\
&\qquad - \sum_{t\in T} c_B I_t^- - \sum_{t\in T} c_U U_t - \sum_{i=1}^{n} c_\rho^i \rho_i
\end{aligned} \tag{2.2a}$$

$$\text{s.t.} \quad W_t = W_{t-1} + H_t - L_t, \forall t \in T, \tag{2.2b}$$

$$O_t \le l_O(W_t - (1 - ap)H_t), \forall t \in T, \tag{2.2c}$$

$$SU_t \le l_{SU}, \forall t \in T, \tag{2.2d}$$

$$P_t \le (W_t - (1 - ap)H_t)n_t hk_R + O_t k_O, \forall t \in T, \tag{2.2e}$$

$$I_t = I_{t-1} + P_t + SU_t - S_t, \forall t \in T, \tag{2.2f}$$

$$P_t \le MU_t, \forall t \in T, \tag{2.2g}$$

$$D_t = d_t(1 + \sum_{i=1}^{n} d_{it}^+ \rho_i), \forall t \in T, \tag{2.2h}$$

$$\sum_{i=1}^{n} \rho_i \le 1, \tag{2.2i}$$

$$I_t = I_t^+ - I_t^-, \ I_{|T|} = I_0 \forall t \in T, \tag{2.2j}$$

$$S_t \leq D_t + I_{t-1}^-, \forall t \in T,\ I_0^- = 0, \tag{2.2k}$$

$$S_t \leq P_t + SU_t + I_{t-1}^+, \forall t \in T,\ I_0^+ = I_0, \tag{2.2l}$$

$$W_t, H_t, L_t, P_t, D_t, SU_t, S_t, I_t^+, I_t^-, O_t \geq 0, \tag{2.2m}$$

$$I_t \in \mathbb{R}, \tag{2.2n}$$

$$U_t, \rho_i \in \{0, 1\}, \forall t \in T, i \in \{1, ..., n\}. \tag{2.2o}$$

The objective function is given in (2.2a). The last three terms of the objective function are new compared to BM1. These terms are related to the costs of backlogging, setup, and applying promotions. Constraints (2.2b)–(2.2f) similarly present the same constraints described in BM1 given in (2.1). Constraints (2.2g) guarantee that U_t is equal to one when the company produces at least one aggregate unit in period t. Constraints (2.2h) add demand increments to d_t thanks to the selected promotion program. Note that $D_t = d_t$ if no promotion program is used. Equation (2.2i) is used to have at most one promotion program for the planning horizon. Constraints (2.2j) indicate the connection of I_t, I_t^-, and I_t^+. Note that our objective is to finish the planning horizon with the inventory level at which we has started (backlog is not possible at the end of horizon). In terms of the number of sales in each period, problem BM1 shows that $S_t \leq d_t$ (see (2.1d)). In BM2, however, the firm may satisfy the backorders from previous periods as well. Then, (2.2k) and (2.2l) guarantee that the number of sales is less than or equal to the actual demand plus backorders and total production capacity plus on-hand inventory level, respectively. Equations (2.2m) and (2.2o) are constraints for nonnegative and binary decision variables, respectively.

2.2.2 *Aggregate Plan with Sustainability Considerations*

Addressing sustainability considerations in supply chain models has become an important goal in the recent years. Aggregate planning is a tool of supply chain management which has potentiality to address many sustainability considerations. We address three main pillars with several sub-pillars for a sustainable aggregate plan. Then, we propose a model for including those pillars and sub-pillars.

2.2.2.1 Including Sustainability in Aggregate Planning

In some countries, the government establishes specific rules for the industries that emit high amounts of greenhouse gases (GHG) or consume non-renewable energy sources. This approach forces these industries to implement manufacturing technologies or strategies which result in less GHG emission and energy usage. We assume that GHG are emitted and energy is consumed because of three main activities: production, inventory holding, and subcontracting.

When a number of innovative projects are available for a decision maker, the projects that improve production capacity and/or make the manufacturer more environmentally-friendly are given priority. A wide range of projects can be considered for this purpose; for example, optimizing and changing the assembly line, implementing new and more efficient equipment, using productive 4.0 methods, etc. Some of these projects require large investments; the decision maker, however, may decide to invest in some of them due to the advantages. We consider two main advantages for these kind of innovative projects: reducing GHG emissions and electricity consumption. These two benefits are considered as the sub-pillars of the environmental pillar and may provide significant benefits in terms of sustainability.

Regarding the social pillar, a number of sub-pillars can be considered in sustainable aggregate planning. The total overtime working hours is one of the sub-pillars that gives flexibility to manufacturers to increase their production capacities. This indicator, however, is significant since the probability of an accident increases during overtime hours. Thus, the safety of employees is improved by decreasing overtime based on the correlation explained. Moreover, work-family conflicts decreases when workers have more time to spend with their families [1].

When a worker is fired, the probability that they would experience emotional, mental, and physical injuries increases. The well-being of that worker decreases as the life expectancy 1.5 years reduces in average. Hence, the number of layoffs is a considerable sub-pillar that needs to be minimized in the social objective function.

Employee gender equity is a sub-pillar that can be included in aggregate planning. In order to satisfy this factor, we minimize the absolute value of the difference between male and female workers. Hence, a large difference between the number of male and female workers is not sustainable in terms of social concerns. We also consider the average productivity of two workers of different genders may be different (note that it may not be applicable in some cases). This consideration makes the problem more realistic. If the average productivity of female workers is less than that of the male workers in a particular industry, the human resources would hire preferably male workers in many cases when there is no difference in the salaries of male/female workers. This sub-pillar, however, supports hiring female workers which is dismissed in many organizations. Countries that have a higher labor force participation rate for women have higher standards of living.

The last sub-pillar by which we address a social concern is related to customer satisfaction level (CSL). One of the main issues in CSL is the availability of a product. Therefore, we minimize the number of stock-outs in the social objective function. Moreover, in order to keep satisfaction level larger than a specific ratio, we restrict the ratio of stock-outs.

2.2.2.2 A Tri-Objective Sustainable Aggregate Planning

We establish a sustainable aggregate planning (SAP) model to consider the concerns discussed in the previous section. The sustainability considerations discussed are addressed within three objective functions representing three pillars mentioned. Note that in order to have a model with single objective function, a decision maker must take those considerations into account by adding additional constraints. For example, instead of minimizing total GHG emissions in the environmental objective function, one may restrict it by an upper bound in the feasible region. This approach is helpful in terms of the solution method. In this case, however, a comprehensive analysis of the trade-offs between different solutions will not be easily obtainable.

We define the following parameters to formulate electricity consumption and GHG emissions due to manufacturing, inventory holding, and subcontracting activities for one item in one period:

$\tilde{g}^P$: GHG emissions due to manufacturing one item in one period.
$\tilde{g}^I$: GHG emissions due to holding one item in inventory for one period.
$\tilde{g}^{SU}$: GHG emissions due to subcontracting one item in one period.
$\tilde{e}^P$: electricity consumption due to manufacturing one item in one period.
$\tilde{e}^I$: electricity consumption due to holding one item in inventory for one period.
$\tilde{e}^{SU}$: electricity consumption due to subcontracting one item in one period.
α: Cycle Service Level (CSL) such that the proportion of stock-outs can not be greater than $1-\alpha$.

We assume that a set of innovative projects is available for the manufacturer, denoted by K. Y_k, $k \in K$ is a binary decision variable that denotes whether project k is implemented (i.e. $Y_k = 1$) or not. We define parameters associated with project k as follows:

c_{IN}^k: the cost of implementing project k.
g_k^I: the total decrease in the amount of GHG emissions due to holding one less item in the inventory in one period if project k is implemented.
g_k^P: the total decrease in the amount of GHG emissions due to manufacturing one less item in one period if project k is implemented.
e_k^I: the decrease in the electricity consumption due to holding one less item in the inventory in one period if project k is implemented.
e_k^P: the decrease in the electricity consumption due to manufacturing one less item in one period if project k is implemented.
δ_{tk} : the production capacity increase in terms of the number of items in period t if project k is implemented ($k \in K$ and $t \in T$).

We also show the following decision variables to connect some parameters and variables.

$\tilde{GI}_k$: the total decrease in GHG emissions due to the inventory holding activities in the planning horizon if project k is implemented ($k \in K$).

$\tilde{GP}_k$: the total decrease in GHG emissions due to the manufacturing activities in the planning horizon if project k is implemented ($k \in K$).

$\tilde{EI}_k$: the total decrease in electricity consumption due to the inventory holding activities in the planning horizon if project k is implemented ($k \in K$).

$\tilde{EP}_k$: the total decrease in electricity consumption due to the manufacturing activities in the planning horizon if project k is implemented ($k \in K$).

Note that $\tilde{GI}_k = Y_k(g_k^I \sum_{t \in T} I_t)$. This constraint is nonlinear, and the same issue exists for $\tilde{GP}_k$, $\tilde{EI}_k$, and $\tilde{EP}_k$. In order to keep the model linear, e.g., for $\tilde{GI}_k = Y_k(g_k^I \sum_{t \in T} I_t)$ where $k \in K$, we propose the following constraints to be added to the model that is presented as SAP.

$$\begin{aligned}
&\tilde{GI}_k \leq MY_k, \\
&\tilde{GI}_k \leq g_k^I \sum_{t \in T} I_t, \\
&\tilde{GI}_k \geq g_k^I \sum_{t \in T} I_t - (1 - Y_k)M, \\
&\tilde{GI}_k \geq 0.
\end{aligned} \tag{2.3}$$

We follow the same approach to linearize the constraints related to $\tilde{GP}_k$, $\tilde{EI}_k$, and $\tilde{EP}_k$. Let *LGHGE* denote the set of constraints that linearize the constraints related to $\tilde{GI}_k = Y_k(g_k^I \sum_{t \in T} I_t)$, $\tilde{GP}_k = Y_k(g_k^P \sum_{t \in T} P_t)$, $\tilde{EI}_k = Y_k(e_k^I \sum_{t \in T} I_t)$, and $\tilde{EP}_k = Y_k(e_k^P \sum_{t \in T} P_t)$, $\forall k \in K$.

We also define the following decision variables and parameters to consider the gender of employees in the model:

Decision Variables:

MW_t : the number of male workers in period t, $t \in T$.

FW_t : the number of female workers in period t, $t \in T$.

MH_t : the number of male workers hired in period t, $t \in T$.

FH_t : the number of female workers hired in period t, $t \in T$.

MF_t : the number of male workers fired in period t, $t \in T$.

FF_t : the number of female workers fired in period t, $t \in T$.

DW_t : the absolute value of the difference between the number of female and male workers in period t, $t \in T$.

Parameters:

MW_0 : the initial number of male workers.

FW_0 : the initial number of female workers.

cw: a constant that shows the maximum and minimum allowed percentage changes in the number of workers comparing to the initial number of workers.

ap_W : the average productivity of a female worker comparing to a male worker.

In the SAP problem given in (2.4), we formulate an aggregate plan with sustainability considerations. We address the three pillars of sustainability as economy, environment, and society. In the environmental and social pillars, we explicitly address two and four sub-pillars, respectively.

SAP

$$\begin{aligned}\max z_{eco} = &\sum_{t\in T} pr_t S_t - \sum_{t\in T} c_P P_t - \sum_{t\in T} c_{SU} SU_t - \sum_{t\in T} s_W (MW_t + FW_t)\\ &- \sum_{t\in T} os_W O_t - \sum_{t\in T} c_H (MH_t + FH_t) - \sum_{t\in T} c_F (MF_t + FF_t) - \sum_{t\in T} c_I I_t\\ &- \sum_{t\in T} c_{SO} SO_t - \sum_{t\in T} c_U U_t - \sum_{i=1}^{n} c_\rho^i \rho_i - \sum_{k\in K} c_{IN}^k Y_k \end{aligned}\tag{2.4a}$$

$$\begin{aligned}\min z_{env} = \ & coef_{env}^1 \Big((\tilde{g}^I \sum_{t\in T} I_t - \sum_{k\in K} \tilde{GI}_k) + \tilde{g}^{SU} \sum_{t\in T} SU_t\\ &+ (\tilde{g}^P \sum_{t\in T} P_t - \sum_{k\in K} \tilde{GP}_k) \Big)\\ &+ coef_{env}^2 \Big((\tilde{e}^I \sum_{t\in T} I_t - \sum_{k\in K} \tilde{EI}_k) + \tilde{e}^{SU} \sum_{t\in T} SU_t\\ &+ (\tilde{e}^P \sum_{t\in T} P_t - \sum_{k\in K} \tilde{EP}_k) \Big)\end{aligned}\tag{2.4b}$$

$$\begin{aligned}\min z_{soc} = \ & coef_{soc}^1 \sum_{t\in T} O_t + coef_{soc}^2 \sum_{t\in T} (FF_t + MF_t)\\ &+ coef_{soc}^3 \sum_{t\in T} DW_t + coef_{soc}^4 \sum_{t\in T} SO_t\end{aligned}\tag{2.4c}$$

$$\text{s.t. } MW_t = MW_{t-1} + MH_t - MF_t, \forall t \in T, \tag{2.4d}$$

$$FW_t = FW_{t-1} + FH_t - FF_t, \forall t \in T, \tag{2.4e}$$

$$FW_t - MW_t \le DW_t, \ MW_t - FW_t \le DW_t, \forall t \in T, \tag{2.4f}$$

$$\begin{aligned}&(1 - cw)(MW_0 + FW_0) \le MW_t + FW_t, \forall t \in T,\\ &MW_t + FW_t \le (1 + cw)(MW_0 + FW_0), \forall t \in T,\end{aligned}\tag{2.4g}$$

$$\begin{aligned}O_t \le l_O \Big(&(MW_t - (1 - ap)MH_t)\\ &+ ap_W (FW_t - (1 - ap)FH_t) \Big), \forall t \in T,\end{aligned}\tag{2.4h}$$

$$SU_t \leq l_{SU}, \forall t \in T, \tag{2.4i}$$

$$P_t \leq \Big((MW_t - (1-ap)MH_t) + ap_W(FW_t - (1-ap)FH_t)\Big)n_t hk_R \\ +O_t k_O + \sum_{k \in K} \delta_{tk} Y_k, \forall t \in T, \tag{2.4j}$$

$$I_t = I_{t-1} + P_t + SU_t - S_t, \forall t \in T,\ I_{12} = I_0, \tag{2.4k}$$

$$P_t \leq MU_t, \forall t \in T, \tag{2.4l}$$

$$S_t + SO_t = D_t, \forall t \in T, \tag{2.4m}$$

$$D_t = d_t(1 + \sum_{i=1}^{n} d_{it}^{+} \rho_i), \forall t \in T, \tag{2.4n}$$

$$SO_t \leq (1-\alpha)D_t, \forall t \in T, \tag{2.4o}$$

$$\sum_{i=1}^{n} \rho_i \leq 1, \tag{2.4p}$$

$$LGHGE, \tag{2.4q}$$

$$MW_t, MH_t, MF_t, FW_t, FH_t, FF_t, DW_t \geq 0, \forall t \in T, \\ D_t, P_t, SU_t, S_t, I_t, SO_t, O_t \geq 0, \forall t \in T, \\ \tilde{GI}_k, \tilde{GP}_k, \tilde{EI}_k, \tilde{EP}_k \geq 0, \forall k \in K, \tag{2.4r}$$

$$U_t, \rho_i, Y_k \in \{0,1\}, \forall t \in T, i \in \{1, ..., n\}, k \in K. \tag{2.4s}$$

Equation (2.4a) is the economic objective function that shows the total profit. Equation (2.4a) has the following terms that are different from (2.2a): 1- $-\sum_{t \in T} c_{SO} SO_t$ since we assume that backorder is not possible, 2- $-\sum_{t \in T} c_I I_t$ due to the same reason, and 3- $-\sum_{k \in K} c_{IN}^k Y_k$ because of the implementation cost of projects. In addition, labor, hiring, and firing costs are updated based on the gender of the workers. Equation (2.4b) shows the environmental objective function. The first line of this equation refers to the amount of GHG emissions, the second line to the total amount of electricity consumed. Note that $coef_{env}^1$ and $coef_{env}^2$ are coefficients for normalizing the GHG emissions and total electricity consumed because their units are different (the coefficients in (2.4c) are present for the same reason). The social objective function is defined in (2.4c) where we minimize the normalized sum of the total overtime, the number of layoffs, the difference between the number of male and female workers, and the number of stock-outs. In the constraints, we set the balance equation for the number of workers by (2.4d) and (2.4e). Equation (2.4f) specifies the difference between the number of female and male workers. Equation (2.4g) forces the number of workers to be within a specific range in all periods. In (2.4j), we set an upper bound for the number of productions in a period such that the average productivity of female workers is taken into account, and the additional capacity originated from innovative projects is calculated. Constraints (2.4o) show that the number of stock-outs must not exceed a specific value to have a certain level of the customer satisfaction. Equation (2.4q) is the set of constraints that are discussed in (2.3). The remaining constraints, (2.4h), (2.4i), (2.4k), (2.4l), (2.4m), (2.4n), (2.4p), (2.4r), and (2.4s) are discussed in (2.2c), (2.2d), (2.2f), (2.2g), (2.1d), (2.2h), (2.2i), (2.2m), and (2.2o).

References

1. Anvari S, Türkay M (2017) The facility location problem from the perspective of triple bottom line accounting of sustainability. Int J Prod Res 55:6266–6287
2. Chopra S, Meindl P (2007) Supply chain management. Strategy, planning & operation. In: Das summa summarum des management. Springer, Berlin, pp 265–275
3. Nahmias S, Cheng Y (2009) Production and operations analysis, vol 6. McGraw-Hill, New York

Chapter 3
Solution Methods for Aggregate Planning Problems Using Python

3.1 Solving a Generic Mathematical Program Using Python

Python is a programming language that has been widely being used by researchers and practitioners. This programming language is open source and has interfaces with many commercial/free-license packages in a variety pf scientific fields. An important functionality of Python is the ability to solve mathematical programs using optimization packages; Cplex and Gurobi are two popular commercial optimization packages for the researchers. These two optimization packages provide either the optimal solution or high-quality solutions for a wide range of mathematical programming problems in a reasonable time. Sometimes these solvers result in a poor solution due to the complexity of a problem and/or limited solution time. Many researchers—particularly in industry—however, use open-source solvers to find solutions to their challenging mathematical programs. One of these packages is *Pulp* (visit https://pypi.org/project/PuLP/ for more information and online tutorial).

In the next sections of this chapter, we provide sample pieces of code in Python to solve the mathematical models presented in the last chapter. In this regard, we need to first establish each model, then, use Pulp as a solver.

3.2 Solving Model BM1

We use the following code to solve model BM1 that is given in (2.1a)–(2.1h). The parameters are set based on [5].

S. A. B. Rasmi, M. Türkay, *Aggregate Planning*, SpringerBriefs in Operations Research, https://doi.org/10.1007/978-3-030-58118-3_3

```
# We define the sets and parameters:
T=12; d=[217823, 217316, 260104, 256002, 317527, 329603, 312316, 383955, 310242, 267525, 245584, 195383]
pr=[1400, 1407, 1414, 1421, 1428, 1435, 1443, 1450, 1457, 1464, 1472, 1479]
n=[20, 20, 27, 25, 24, 20, 18, 25, 23, 25, 26, 20]
cH=1000; cF=3500; cI=9.86; cP=1142; cSU=1220; lSU=40000; cSO=950; sW=3050
osW=28; hr=8; lO=20; kR=0.6; kO=0.45; ap=0.6; W0=1900; I0=85000

# We import the packages required
import pulp # Open source optimizer

# We define the model
model = pulp.LpProblem('BM1', pulp.LpMaximize)

# We define decision variables (2.1h)
Wt={}; Ht={}; Lt={}; Pt={}; SUt={}; It={}; St={}; SOt={}; Ot={}
for t in range(T):
    Wt[t] = pulp.LpVariable('W_'+str(t+1), lowBound=0, cat='Continuous')
    Ht[t] = pulp.LpVariable('H_'+str(t+1), lowBound=0, cat='Continuous')
    Lt[t] = pulp.LpVariable('L_'+str(t+1), lowBound=0, cat='Continuous')
    Pt[t] = pulp.LpVariable('P_'+str(t+1), lowBound=0, cat='Continuous')
    SUt[t] = pulp.LpVariable('SU_'+str(t+1), lowBound=0, cat='Continuous')
    It[t] = pulp.LpVariable('I_'+str(t+1), lowBound=0, cat='Continuous')
    St[t] = pulp.LpVariable('S_'+str(t+1), lowBound=0, cat='Continuous')
    SOt[t] = pulp.LpVariable('SO_'+str(t+1), lowBound=0, cat='Continuous')
    Ot[t] = pulp.LpVariable('O_'+str(t+1), lowBound=0, cat='Continuous')

#2.1a
```

```
model += pulp.lpSum([pr[t]*St[t] - cP*Pt[t] - cSU*SUt[t] - sW*Wt[t] - osW*Ot[t] - cH*Ht[t] - cF*Lt[t] - cI*It[t] - cSO*SOt[t] for t in range(T)])

#2.1b
model += pulp.lpSum([W0+Ht[0] - Lt[0] - Wt[0]]) == 0
for t in range(1,T):
    model += pulp.lpSum([Wt[t-1] + Ht[t] - Lt[t] - Wt[t]]) == 0

#2.1c
model += pulp.lpSum([I0 + Pt[0] + SUt[0] - St[0] - It[0]]) == 0
model += pulp.lpSum([It[11]]) == I0
for t in range(1,T):
    model += pulp.lpSum([It[t-1] + Pt[t] + SUt[t] - St[t] - It[t]]) == 0

#2.1d
for t in range(T):
    model += pulp.lpSum([St[t] + SOt[t]]) == d[t]

#2.1e
for t in range(T):
    model += pulp.lpSum([Pt[t] - n[t]*hr*kR*(Wt[t] - (1 - ap)*Ht[t]) - Ot[t]*kO]) <= 0

#2.1f
for t in range(T):
    model += pulp.lpSum([Ot[t] - lO*(Wt[t] - (1 - ap)*Ht[t])]) <= 0

#2.1g
for t in range(T):
    model += pulp.lpSum(SUt[t]) <= lSU

# We solve the problem
model.solve()

# If an optimal solution exists for the problem, we find it. We also can extract the optimal values of the decision variables.
print("Status:", pulp.LpStatus[model.status])
```

```
print("Optimal objective function value = ", pulp.value(model.objective))
if pulp.LpStatus[model.status]=='Optimal':
    for v in model.variables():
        # We print decision variables with non-zero values.
        if v.varValue!=0:
            print(v.name, "=", v.varValue)
```

This optimization problem results in an optimal objective function value equal to 884113102. By compiling the code, it is possible to find the optimal value of the decision variables.

3.3 Solving Model BM2

We use the following code to solve model BM2, given in (2.2a)–(2.2o). The parameters are set based on [5].

```
 # We define the sets and parameters:
T=12; N=3
d=[217823, 217316, 260104, 256002, 317527, 329603, 312316, 383955, 310242, 267525, 245584, 195383]
pr=[1400, 1407, 1414, 1421, 1428, 1435, 1443, 1450, 1457, 1464, 1472, 1479]
n=[20, 20, 27, 25, 24, 20, 18, 25, 23, 25, 26, 20]
crho=[4800000, 4900000, 4200000]
cH=1000; cF=3500; cI=9.86; cP=1142; cSU=1220; lSU=40000; sW=3050
osW=28; hr=8; lO=20; kR=0.6; kO=0.45; ap=0.6; W0=1900; I0=85000
Iplus0=I0
Iminus0=0; cU=30000000; cB=300; M=1000000
ditP=[[0.014, 0.014, 0.014, 0.014, 0.014, 0.014, 0.017, 0.017, 0.017, 0.017, 0.017, 0.017], [0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02], [0.021, 0.021, 0.018, 0.018, 0.015, 0.014, 0.015, 0.013, 0.015, 0.017, 0.019, 0.024]]

# We import the packages required
import pulp # Open source optimizer

# We define the model
```

```python
model = pulp.LpProblem('BM2', pulp.LpMaximize)

# We define decision variables (2.2m, 2.2n, 2.2o)

Wt={}; Ht={}; Lt={}; Pt={}; Dt={}; SUt={}; Itplus={}; Itminus={}; St={}; Ot={}
It={}
Ut={}; rhoi={}
for t in range(T):
    Wt[t] = pulp.LpVariable('W_'+str(t+1), lowBound=0, cat='Continuous')
    Ht[t] = pulp.LpVariable('H_'+str(t+1), lowBound=0, cat='Continuous')
    Lt[t] = pulp.LpVariable('L_'+str(t+1), lowBound=0, cat='Continuous')
    Pt[t] = pulp.LpVariable('P_'+str(t+1), lowBound=0, cat='Continuous')
    Dt[t] = pulp.LpVariable('D_'+str(t+1), lowBound=0, cat='Continuous')
    SUt[t] = pulp.LpVariable('SU_'+str(t+1), lowBound=0, cat='Continuous')
    Itplus[t] = pulp.LpVariable('Iplus_'+str(t+1), lowBound=0, cat='Continuous')
    Itminus[t] = pulp.LpVariable('Iminus_'+str(t+1), lowBound=0, cat='Continuous')
    St[t] = pulp.LpVariable('S_'+str(t+1), lowBound=0, cat='Continuous')
    Ot[t] = pulp.LpVariable('O_'+str(t+1), lowBound=0, cat='Continuous')
    It[t] = pulp.LpVariable('I_'+str(t+1), cat='Continuous')
    Ut[t] = pulp.LpVariable('U_'+str(t+1), cat='Binary')

for i in range(N):
    rhoi[i] = pulp.LpVariable('rho_'+str(i+1), cat='Binary')

#2.2a
model += pulp.lpSum([pr[t]*St[t] - cP*Pt[t] - cSU*SUt[t] - sW*Wt[t] - osW*Ot[t] - cH*Ht[t] - cF*Lt[t] - cI*Itplus[t] - cB*Itminus[t] - cU*Ut[t] for t in range(T)]) - pulp.lpSum([crho[i]*rhoi[i] for i in range(N)])
```

```python

#2.2b
model += pulp.lpSum([W0 + Ht[0] - Lt[0] - Wt[0]]) == 0
for t in range(1,T):
    model += pulp.lpSum([Wt[t-1] + Ht[t] - Lt[t] - Wt[t]]) == 0

#2.2c
for t in range(T):
    model += pulp.lpSum([Ot[t] - lO*(Wt[t] - (1 - ap)*Ht[t])]) <= 0

#2.2d
for t in range(T):
    model += pulp.lpSum(SUt[t]) <= lSU

#2.2e
for t in range(T):
    model += pulp.lpSum([Pt[t] - n[t]*hr*kR*(Wt[t] - (1 - ap)*Ht[t]) - Ot[t]*kO]) <= 0

#2.2f

model += pulp.lpSum([I0 + Pt[0] + SUt[0] - St[0] - It[0]]) == 0
model += pulp.lpSum([It[11]]) == I0
for t in range(1,T):
    model += pulp.lpSum([It[t-1] + Pt[t] + SUt[t] - St[t] - It[t]]) == 0

#2.2g
for t in range(T):
    model += pulp.lpSum([Pt[t]]) <= M*Ut[t]

#2.2h
for t in range(T):
    model += pulp.lpSum([d[t]*(1 + sum(rhoi[i]*ditP[i][t] for i in range(N)))]) == Dt[t]

#2.2i
model += pulp.lpSum([rhoi[i] for i in range(N)]) <= 1

#2.2j
```

```
for t in range(T):
    model += pulp.lpSum([Itplus[t] - Itminus[t]]) == It[t]

#2.2k
model += pulp.lpSum([Dt[0] - Iminus0]) >= St[0]

for t in range(1,T):
    model += pulp.lpSum([Dt[t] + Itminus[t-1]]) >= St[t]

#2.2l
model += pulp.lpSum([Pt[0] + SUt[0]+Iplus0]) >= St[0]
for t in range(1,T):
    model += pulp.lpSum([Pt[t] + SUt[t] + Itplus[t-1]]) >= St[t]

# We solve the problem
model.solve()

# If an optimal solution exists for the problem, we find it. We also can extract the optimal values of the decision variables.
print("Status:", pulp.LpStatus[model.status])
print("Optimal objective function value = ", pulp.value(model.objective))
if pulp.LpStatus[model.status]=='Optimal':
    for v in model.variables():
        # We print decision variables with non-zero values.
        if v.varValue!=0:
            print(v.name, "=", v.varValue)
```

This model results in an optimal objective function value equal to 631804202. By compiling the code, it is possible to find the values of decision variables at the optimal solution. Note that the optimal objective function value is lower than the optimal objective function of BM1. This is due to the fact that we have more restrictive constraints in BM2.

3.4 Solving Model SAP

The SAP problem given in (2.4a)–(2.4s) is a tri-objective mixed-integer linear program (TOMILP). Most of the time, in the presence of more than one objective function, we are interested in finding a set of non-dominated (ND) points and/or efficient solutions. A generic form of a multi-objective program is as follows:

$$\begin{aligned} \max \ & z(x) = Cx \\ \text{s.t.} \ & x \in S, \end{aligned} \tag{3.1}$$

where, S is a feasible region of (3.1) defined by a set of constraints. $z(x) = (z_1(x), \ldots, z_k(x))$ (s.t. $k > 1$), x is the n-vector of decision variables, and C is a $k \times n$ matrix.

Assume that $\hat{x} \in S$ exists such that there is no vector in the set $\{Cx \mid x \in S, z_j(x) \geq z_j(\hat{x}), j = 1, \ldots, k, \ Cx \neq z(\hat{x})\}$. Then, $\hat{x}$ is an efficient solution and $z(\hat{x})$ is a ND point. Based on the mathematical definition of a ND point, we present the following problem that is a weighted-sum of (3.1).

$$\begin{aligned} \max \ & \bar{z}(x) = \sum_{j=1}^{k} \lambda_j z_j(x) \\ \text{s.t.} \ & x \in S, \end{aligned} \tag{3.2}$$

where λ_j's are positive real values. The optimal solution of (3.2) is an efficient solution for (3.1). If the positivity condition is replaced by the nonnegativity condition for the weights, there is a potential for the solution to be weakly efficient in some cases where the feasible region is non-convex [1].

Our objective is to generate a number of ND points or efficient solutions for the SAP model given in (2.4a)–(2.4s). Hence, we use the concept of weighted-sum method for this task. On the other hand, due to the fact that there are different measurement units for the terms in the objective functions and numerical complexities, we normalize the objective functions in the following Python code. In this regard, we first minimize and maximize each term in the objective functions to find their ranges. Then, we divide the current coefficients of objective functions by the corresponding ranges. In the obtained objective functions, the scale of the terms in z_{env} and z_{soc} become consistent. Therefore, we can sum two terms of z_{env}, and four terms of z_{soc} to construct these objective functions.

```
# We define the sets and parameters:
T=12; N=3; K=3
d=[217823, 217316, 260104, 256002, 317527, 329603,
 ↪  312316, 383955, 310242, 267525, 245584, 195383]
pr=[1400, 1407, 1414, 1421, 1428, 1435, 1443, 1450,
 ↪  1457, 1464, 1472, 1479]
n=[20, 20, 27, 25, 24, 20, 18, 25, 23, 25, 26, 20]
crho=[4800000, 4900000, 4200000]
```

```
ditP=[[0.014, 0.014, 0.014, 0.014, 0.014, 0.014, 0.017, 0.017, 0.017, 0.017, 0.017, 0.017], [0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02], [0.021, 0.021, 0.018, 0.018, 0.015, 0.014, 0.015, 0.013, 0.015, 0.017, 0.019, 0.024]]
cH=1000; cF=3500; cI=9.86; cP=1142; cSU=1220; lSU=40000
cSO=950; sW=3050; osW=28; hr=8; lO=20; kR=0.6; kO=0.45
ap=0.6; apW=0.9; cw=0.4; MW0=1600; FW0=300; I0=85000
cU=30000000; M=1000000;
gtildeP=0.0379; gtildeI=0.0048; gtildeSU=0.04
etildeP=0.1346; etildeI=0.0057; etildeSU=0.14
alpha=0.975; prop=[0.18, 0.15, 0.25]
ckIN=[58500000, 39000000, 78000000]

gIk=[val*gtildeI for val in prop]
gPk=[val*gtildeP for val in prop]
eIk=[val*etildeI for val in prop]
ePk=[val*etildeP for val in prop]
deltatk=[[12000, 7000, 15000], [12000, 7000, 15800], [12000, 7000, 19000], [12000, 7000, 18500], [12000, 7000, 23000], [12000, 7000, 24000], [12000, 7000, 22500], [12000, 7000, 28000], [12000, 7000, 22500], [12000, 7000, 19500], [12000, 7000, 18000], [12000, 7000, 13500]]

# We import the packages required
import pulp # Open source optimizer

# We define the model
model = pulp.LpProblem('SAP', pulp.LpMaximize)

# We define decision variables (2.4r, 2.4s)
MWt={}; MHt={}; MFt={}; FWt={}; FHt={}; FFt={}
DWt={}
Pt={}; Dt={}; SUt={}; It={}; SOt={}; St={}; Ot={}
GtildeIk={}; GtildePk={}; EtildeIk={}; EtildePk={}
Ut={}; rhoi={}; Yk={}

for t in range(T):
    MWt[t] = pulp.LpVariable('MW_'+str(t+1), lowBound=0, cat='Continuous')
```

```
    MHt[t] = pulp.LpVariable('MH_'+str(t+1), lowBound=0, cat='Continuous')
    MFt[t] = pulp.LpVariable('MF_'+str(t+1), lowBound=0, cat='Continuous')
    FWt[t] = pulp.LpVariable('FW_'+str(t+1), lowBound=0, cat='Continuous')
    FHt[t] = pulp.LpVariable('FH_'+str(t+1), lowBound=0, cat='Continuous')
    FFt[t] = pulp.LpVariable('FF_'+str(t+1), lowBound=0, cat='Continuous')
    DWt[t] = pulp.LpVariable('DW_'+str(t+1), lowBound=0, cat='Continuous')
    Pt[t] = pulp.LpVariable('P_'+str(t+1), lowBound=0, cat='Continuous')
    Dt[t] = pulp.LpVariable('D_'+str(t+1), lowBound=0, cat='Continuous')
    SUt[t] = pulp.LpVariable('SU_'+str(t+1), lowBound=0, cat='Continuous')
    It[t] = pulp.LpVariable('I_'+str(t+1), lowBound=0, cat='Continuous')
    SOt[t] = pulp.LpVariable('SO_'+str(t+1), lowBound=0, cat='Continuous')
    St[t] = pulp.LpVariable('S_'+str(t+1), lowBound=0, cat='Continuous')
    Ot[t] = pulp.LpVariable('O_'+str(t+1), lowBound=0, cat='Continuous')
    Ut[t] = pulp.LpVariable('U_'+str(t+1), cat='Binary')

for i in range(N):
    rhoi[i] = pulp.LpVariable('rho_'+str(i+1), cat='Binary')
for k in range(K):
    GtildeIk[k] = pulp.LpVariable('GtildeI_'+str(k+1), cat='Binary')
    GtildePk[k] = pulp.LpVariable('GtildeP_'+str(k+1), cat='Binary')
    EtildeIk[k] = pulp.LpVariable('EtildeI_'+str(k+1), cat='Binary')
    EtildePk[k] = pulp.LpVariable('EtildeP_'+str(k+1), cat='Binary')
    Yk[k] = pulp.LpVariable('Y_'+str(k+1), cat='Binary')

```

```python
#zeco (2.4a)
z = [] # z is a Python list where each element is a term of one of the objective functions
z.append(pulp.lpSum([pr[t]*St[t] - cP*Pt[t] - cSU*SUt[t] - sW*(MWt[t] + FWt[t]) - osW*Ot[t] - cH*(MHt[t] + FHt[t]) - cF*(MFt[t] + FFt[t]) - cI*It[t] - cSO*SOt[t] - cU*Ut[t] for t in range(T)]) - pulp.lpSum([crho[i]*rhoi[i] for i in range(N)]) - pulp.lpSum([ckIN[k]*Yk[k] for k in range(K)]))

#zenv (2.4b)

z.append(pulp.lpSum([gtildeI*It[t] for t in range(T)]) - pulp.lpSum([GtildeIk[k] for k in range(K)]) + pulp.lpSum([gtildeSU*SUt[t] for t in range(T)]) + pulp.lpSum([gtildeP*Pt[t] for t in range(T)]) - pulp.lpSum([GtildePk[k] for k in range(K)]))
z.append(pulp.lpSum([etildeI*It[t] for t in range(T)]) - pulp.lpSum([EtildeIk[k] for k in range(K)]) + pulp.lpSum([etildeSU*SUt[t] for t in range(T)]) + pulp.lpSum([etildeP*Pt[t] for t in range(T)]) - pulp.lpSum([EtildePk[k] for k in range(K)]))

#zsoc (2.4c)
z.append(pulp.lpSum([Ot[t] for t in range(T)]))
z.append(pulp.lpSum([(FFt[t]+MFt[t]) for t in range(T)]))
z.append(pulp.lpSum([DWt[t] for t in range(T)]))
z.append(pulp.lpSum([SOt[t] for t in range(T)]))

#2.4d
model += pulp.lpSum([MW0 + MHt[0] - MFt[0] - MWt[0]]) == 0
for t in range(1,T):
    model += pulp.lpSum([MWt[t-1] + MHt[t] - MFt[t] - MWt[t]]) == 0

#2.4e

model += pulp.lpSum([FW0 + FHt[0] - FFt[0] - FWt[0]]) == 0
for t in range(1,T):
```

```python
    model += pulp.lpSum([FWt[t-1] + FHt[t] - FFt[t] - FWt[t]]) == 0

#2.4f
for t in range(T):
    model += pulp.lpSum([-FWt[t] + MWt[t]]) <= DWt[t]
    model += pulp.lpSum([FWt[t] - MWt[t]]) <= DWt[t]

#2.4g
for t in range(T):
    model += pulp.lpSum([MWt[t] + FWt[t]]) >= (1 - cw)*(MW0 + FW0)
    model += pulp.lpSum([MWt[t] + FWt[t]]) <= (1 + cw)*(MW0 + FW0)

#2.4h
for t in range(T):
    model += pulp.lpSum([lO*((MWt[t] - (1 - ap)*MHt[t]) + apW*(FWt[t] - (1 - ap)*FHt[t]))]) >= Ot[t]

#2.4i
for t in range(T):
    model += pulp.lpSum(SUt[t]) <= lSU

#2.4j
for t in range(T):
    model += pulp.lpSum([n[t]*hr*kR*((MWt[t] - (1 - ap)*MHt[t]) + apW*(FWt[t] - (1 - ap)*FHt[t])) + Ot[t]*kO + sum(Yk[k]*deltatk[t][k] for k in range(K))]) >= Pt[t]

#2.4k
model += pulp.lpSum([I0 + Pt[0] + SUt[0] - St[0] - It[0]]) == 0
model += pulp.lpSum([It[11]]) == I0
for t in range(1,T):
    model += pulp.lpSum([It[t-1] + Pt[t] + SUt[t] - St[t] - It[t]]) == 0

#2.4l
for t in range(T):
    model += pulp.lpSum([Pt[t]]) <= M*Ut[t]
```

```
#2.4m
for t in range(T):
    model += pulp.lpSum([St[t] + SOt[t]]) == Dt[t]

#2.4n
for t in range(T):
    model += pulp.lpSum([d[t]*(1 + sum(rhoi[i]*ditP[i][t] for i in range(N)))]) == Dt[t]

#2.4o
for t in range(T):
    model += pulp.lpSum([SOt[t]]) <= (1 - alpha)*Dt[t]

#2.4p
model += pulp.lpSum([rhoi[i] for i in range(N)]) <= 1

#LGHGE (2.4q)
# For GtildeIk[k]'s
for k in range(K):
    model += pulp.lpSum([GtildeIk[k]]) <= M*Yk[k]
    model += pulp.lpSum([gIk[k]*sum(It[t] for t in range(T))]) >= GtildeIk[k]
    model += pulp.lpSum([gIk[k]*sum(It[t] for t in range(T)) - (1 - Yk[k])*M]) <= GtildeIk[k]
# For GtildePk[k]'s
for k in range(K):
    model += pulp.lpSum([GtildePk[k]]) <= M*Yk[k]
    model += pulp.lpSum([gPk[k]*sum(It[t] for t in range(T))]) >= GtildePk[k]
    model += pulp.lpSum([gPk[k]*sum(It[t] for t in range(T)) - (1 - Yk[k])*M]) <= GtildePk[k]
# For EtildeIk[k]'s
for k in range(K):
    model += pulp.lpSum([EtildeIk[k]]) <= M*Yk[k]
    model += pulp.lpSum([eIk[k]*sum(It[t] for t in range(T))]) >= EtildeIk[k]
    model += pulp.lpSum([eIk[k]*sum(It[t] for t in range(T)) - (1 - Yk[k])*M]) <= EtildeIk[k]
# For EtildePk[k]'s
for k in range(K):
    model += pulp.lpSum([EtildePk[k]]) <= M*Yk[k]
    model += pulp.lpSum([ePk[k]*sum(It[t] for t in range(T))]) >= EtildePk[k]
```

```
    model += pulp.lpSum([ePk[k]*sum(It[t] for t in range(T)) - (1 - Yk[k])*M]) <= EtildePk[k]

# We normalize the objective functions by starting with their terms. Note that zenv and zsoc have two and four terms, respectively.
maxminlist=[] # A list for the maximum and minimum values of the objective functions.
for j in range(7):
    if j==5: # Note that the maximization of the sum of DWt's is unbounded. Therefore, we simply assume that its upperbound is equal to the product of the maximum number of labors and the number of periods (T). Then, we add the following constraint to the model.
        model += pulp.lpSum([DWt[t] for t in range(T)]) <= ((1 + cw)*(MW0 + FW0)*T)
    model += z[j]
    model.solve()
    fvalmax=pulp.value(model.objective)
    model += (-1)*z[j]
    model.solve()
    fvalmin=-pulp.value(model.objective)
    maxminlist.append([fvalmax,fvalmin])
# After finding the maximum and minimum values of the terms used in each objective function, we normalize these seven terms.
for j in range(7):
    z[j]=(1/(maxminlist[j][0] - maxminlist[j][1]))*z[j]
# We define "Z" as a list of three objective functions
Z=[]
# for Zeco
Z.append(z[0])
# for Zenv
Z.append(z[1] + z[2])
# for Zsoc
Z.append(z[3] + z[4] + z[5] + z[6])
# We normalize three objective functions.
maxminlist=[]
for j in range(3):
    model += Z[j]
    model.solve()
    fvalmax=pulp.value(model.objective)
```

```
    model += (-1)*Z[j]
    model.solve()
    fvalmin=-pulp.value(model.objective)
    maxminlist.append([fvalmax,fvalmin])
for j in range(3):
    Z[j]=(1/(maxminlist[j][0] -
     ↪  maxminlist[j][1]))*Z[j]
```

After normalizing the coefficients of the objective functions, we perform the following piece of code to solve the SAP model with a single objective function. The objective function is a weighted-sum of three objective functions normalized. We solve the problem several times with different weights. We assign positive values to the weights to make sure that the results of solving the problems (i.e., weighted-sum problems similar to (3.2)) will be ND point.

If nonnegative values are assigned to the weights, we require to exploit a remedy based on lexicographic optimization that avoids generating weakly ND points. In lexicographic optimization, we optimize one objective function and fix the value of that objective function to the found value as a constraint in the next problems. For example, when three objective functions exist, we may optimize z_1 subject to the feasible region and denote the optimal objective function value as z_1^*. Then, we optimize z_2 subject to the feasible region and constraint $z_1(x) = z_1^*$. Let z_2^* denote the optimal objective function value of the second optimization problem. Then, we optimize z_3 subject to the feasible region, $z_1(x) = z_1^*$, and $z_2(x) = z_2^*$. Such a way consumes more calculations and may result in finding a larger number of ND points.

```
 import numpy as np

# A parameter to control the change of weights
 ↪  (lambda_j).
stepsize = 0.01

# A parameter that must be set into a small positive
 ↪  value to store only one ND point from the ND
 ↪  points which are very close to each other.
smallval = 0.01

# A function that returns 0 if ND1 and ND2 are not the
 ↪  same considering "smallval".
def ALL_thesame(ND1,ND2,smallval):
    for j in range(3):
        if abs(ND1[j] - ND2[j])>=smallval:
            return 0
```

```
    # If this function has not returned a value, it means that |ND1[j] - ND2[j]|<=smallval for all j = 0, 1, 2. In other words, these two points are almost the same in terms of objective function values.
    return 1

# A function to see whether two strategies are the same. This function returns 1 if str1 and str2 are the same.
def ALL_thesame_strategies(str1,str2,T,K,N):
    for i in range(T + K + N):
        if str1[i] != str2[i]:
            return 0
    # If this function has not returned a value, it means that two strategies are the same.
    return 1

# We store some ND points and the strategies which are related to the solution of binary variables. Note that binary variables represent more important (strategic) decision variables.
NDpoints=[]; strategies=[]

for l1 in np.arange(0.01,0.98,stepsize):
    for l2 in np.arange(0.01,1-l1-0.01,stepsize):
        l3 = 1 - l1 - l2
        Z_weighted_sum = l1*Z[0] - l2*Z[1] - l3*Z[2]
        model += pulp.lpSum([Z_weighted_sum])
        model.solve()
        if pulp.LpStatus[model.status]=='Optimal':
            # We generate the ND point that corresponds to the found solution by pulp.
            NDpoint=[]
            for j in range(3):
                NDpoint.append(pulp.value(pulp.lpSum([Z[j]])))
            # We check if any ND point that has been found previously is close to "NDpoint" or not?
            ExistBefore=0
            for ND in NDpoints:
```

```python
                thesame = ALL_thesame(ND, NDpoint, smallval)# See how this function works in "def ALL_thesame(ND1,ND2,smallval)".
                if thesame == 1:
                    ExistBefore = 1
                    break
            if ExistBefore == 0:
                NDpoints.append(NDpoint)
            # We generate the strategy that corresponds to the found solution by pulp.
            Strategy=[]
            for t in range(T):
                Strategy.append( round( pulp.value(pulp.lpSum([Ut[t]]))))
            for i in range(N):
                Strategy.append( round( pulp.value(pulp.lpSum([rhoi[i]]))))
            for k in range(K):
                Strategy.append( round( pulp.value(pulp.lpSum([Yk[k]]))))
            # We check whether this strategy provides a new strategy or not?
            ExistBefore=0
            for str1 in strategies:
                thesame = ALL_thesame_strategies(str1, Strategy, T, K, N) # See how this function works in "def ALL_thesame_strategies(str1, str2, T, K, N)".
                if thesame==1:
                    ExistBefore=1
                    break
            if ExistBefore==0:
                strategies.append(Strategy)

print(strategies)
print(NDpoints)
```

The solution of multi-objective mixed-integer optimization problems has been an attractive research area. The readers are referred to the papers by Fattahi and Turkay [2] for bi-objective MILPs (BOMILP), Rasmi et al. [4] for tri-objective MILPs (TOMILP), and Rasmi and Turkay [3] for multi-objective MILPs (MOMILP).

References

1. Ehrgott M (2005) Multicriteria optimization, vol 491. Springer, Berlin
2. Fattahi A, Türkay M (2018) A one direction search method to find the exact nondominated frontier of biobjective mixed-binary linear programming problems. Eur J Oper Res 266:415–425
3. Rasmi SAB, Türkay M (2019) Gondef: an exact method to generate all non-dominated points of multi-objective mixed-integer linear programs. Optim Eng 20(1):89–117
4. Rasmi SAB, Fattahi A, Türkay M (2021) SASS: slicing with adaptive steps search method for finding the non-dominated points of tri-objective mixed-integer linear programming problems. Ann Oper Res 296:841–876
5. Rasmi SAB, Kazan C, Türkay M (2019) A multi-criteria decision analysis to include environmental, social, and cultural issues in the sustainable aggregate production plans. Comput Ind Eng 132:348–360

Chapter 4
Analysis and Conclusions

4.1 Analysis of the ND Points of the SAP Model

We generate 228 ND points which are not equal to each other (this is guaranteed by using parameters *stepsize* = 0.01 and *smallval* = 0.01). Note that we may decide to generate a larger number of ND points by choosing smaller positive values for *stepsize* and *smallval*. Let $z^1 = (z^1_{eco}, z^1_{env}, z^1_{soc})$, $z^2 = (z^2_{eco}, z^2_{env}, z^2_{soc})$, ..., and $z^P = (z^P_{eco}, z^P_{env}, z^P_{soc})$ be the generated ND points. We next use the approach used by Rasmi et al. [5] to change objective function values that correspond to the ND points to some values between 0 and 100. Those numbers state the performance of the ND points in each criterion/pillar. For example, we are interested in calculating the performance of ND point $\hat{z}$ (denoted by $p(\hat{z}) := (p(\hat{z}_1), p(\hat{z}_2), p(\hat{z}_3))$ where $0 \leq p(\hat{z}_j) \leq 100$, for $j = 1, 2, 3$). Note that $\hat{z}$ belongs to the set $\{z^1, \ldots, z^P\}$. Its performance in the jth pillar is equal to:

- $\frac{max\{z^i_j | i \in \{1,\ldots,P\}\} - \hat{z}_j}{max\{z^i_j | i \in \{1,\ldots,P\}\} - min\{z^i_j | i \in \{1,\ldots,P\}\}} \times 100$ if the corresponding objective function is minimization,
- $\frac{\hat{z}_j - min\{z^i_j | i \in \{1,\ldots,P\}\}}{max\{z^i_j | i \in \{1,\ldots,P\}\} - min\{z^i_j | i \in \{1,\ldots,P\}\}} \times 100$ if the corresponding objective function is maximization.

When we analyze the generated ND points, the binary variables deserve more attention due to their role in the optimization problem. The binary variables, U_t, ρ_i, and Y_k, indicate a set of strategies that result in a ND point. We also note that the ND points generated correspond to 18 unique sets of strategies or binary solutions. This is a crucial information for the decision maker to let them know that a win-win strategy is obtainable among those 18 strategies.

The solution method we presented is a simple method that generates a large number of the ND points. In other words, the output is a subset of the ND points set.

S. A. B. Rasmi, M. Türkay, *Aggregate Planning*, SpringerBriefs in Operations Research, https://doi.org/10.1007/978-3-030-58118-3_4

There are studies, however, that generate the exact set of the ND points of BOMILPs (see [1]), TOMILPs (see [4]) and a superset of the ND points set for MOMILPs (see [3]). They also guarantee that all binary solutions which result in at least one ND point, are generated. In other words, they generate all ND strategies. Note that due to the nature of those algorithms to find all ND points, their time performance may be seriously affected in solving large instances.

Some ND points result in a high performance in some pillars; they are, however, not balanced. For example, (100, 29.43, 32.34), (6.59, 100, 4.5), and (67.61, 60.65, 100) are the performance of the some of the ND points generated. In spite of the fact that they result in a high performance in the economic, environmental, and social pillars, respectively, they are not considered as win-win solutions. In order to obtain a solution that results in a high performance for all pillars, we use four simple functions as follows:

$F1$: We calculate the Euclidean distance of the performance of a ND point from the ideal goal where the ideal goal is to reach the performance of 100 in all pillars. Therefore, $F1(p(\hat{z})) = \sqrt{\sum_{j=1}^{3}(100 - p(\hat{z}_j))^2}$. A smaller value of $F1$ is more preferable.

$F2$: This function calculates the Manhattan distance of a ND point from (100, 100, 100). Hence, $F2(p(\hat{z})) = \sum_{j=1}^{3}(100 - p(\hat{z}_j))$, and we aim to find the points that minimize $F2$.

$F3$: This function calculates the variance of the performances of a ND point in three pillars. Let $p(\hat{\bar{z}})$ be equal to $\frac{\sum_{j=1}^{3} p(\hat{z}_j)}{3}$. Then, $F3(p(\hat{z})) = \frac{\sum_{j=1}^{3}(p(\hat{z}_j) - p(\hat{\bar{z}}))^2}{3}$. A smaller value of this function for a ND point shows that the ND point gives a more balanced result in all pillars. Therefore, minimizing this function is desirable.

$F4(p(\hat{z})) = \prod_{j=1}^{3} p(\hat{z}_j)$. This function is motivated by the calculation of the Nash bargaining solution proposed by Nash [2]. A larger value of this function for a ND point is more preferable for a decision maker.

In Table 4.1, we compare the performance of a number of ND points. We aim to show that although some ND points result in the performance of 100 in one criterion, their performance in other pillars are quite low, and hence, we do not consider them as win-win solutions. Assume that a decision maker is only concerned with the monetary issues or the economic pillar. Therefore, she recommends the solution that results in (100, 29.43, 32.34) to her organization. However, this solution does not provide a sustainable result. On the other hand, (71.46, 64.61, 89.2) is the performance of a ND point by which rank 1 (based on applying $F1$), 9 (based on applying $F2$), 27 (based on applying $F3$), and 6 (based on applying $F4$) are achievable. This ND point gives objective function values such that it is 8% worse than the corresponding economic objective function value of (100, 29.43, 32.34), 4% better than the environmental objective function value of that point, and 83% better than the social objective function value of (100, 29.43, 32.34). It shows that if the decision maker accepts to have a slightly smaller profit, this decision results in significant improvements in terms of sustainability. A similar story happens if the

Table 4.1 The performance of some ND points and their ranks based on $F1$, $F2$, $F3$, and $F4$ functions

			Rank among 228 ND points based on			
Applied function	$p(\hat{z})$	$F\#(p(\hat{z}))$	$F1$	$F2$	$F3$	$F4$
$F1$	(71.46, 64.61, 89.2)	46.72	1	9	27	6
	(69.88, 66.81, 86.65)	46.76	2	12	17	10
	(69.52, 65.28, 92.12)	46.86	3	6	35	3
	(100, 29.43, 32.34)	97.76	175	177	165	161
	(6.59, 100, 4.5)	133.59	225	225	225	217
	(67.61, 60.65, 100)	50.97	16	4	77	7
$F2$	(69.91, 60.47, 98.77)	70.85	12	1	69	4
	(69.93, 62.19, 96.97)	70.91	10	2	59	1
	(70.31, 62.92, 95.32)	71.45	6	3	52	2
	(100, 29.43, 32.34)	138.23	175	177	165	161
	(6.59, 100, 4.5)	188.91	225	225	225	217
	(67.61, 60.65, 100)	71.74	16	4	77	7
$F3$	(73.03, 68.3, 68.91)	4.41	22	46	1	32
	(73.3, 68.77, 65.47)	10.3	31	52	2	37
	(64.33, 72.67, 67.37)	11.88	3 8	58	3	47
	(100, 29.43, 32.34)	1062.9	175	177	165	161
	(6.59, 100, 4.5)	1983.3	225	225	225	217
	(67.61, 60.65, 100)	294	16	4	77	7
$F4$	(69.93, 62.19, 96.97)	421720.75	10	2	59	1
	(70.31, 62.92, 95.32)	421705.6	6	3	52	2
	(69.52, 65.28, 92.12)	418123.09	3	6	35	3
	(100, 29.43, 32.34)	95185.03	175	177	165	161
	(6.59, 100, 4.5)	2966.06	225	225	225	217
	(67.61, 60.65, 100)	410048.96	16	4	77	7

ND points that provide high performances based on $F2$, $F3$, and $F4$ are selected. In Table 4.1, each time, we take either of functions $F1$, $F2$, $F3$, and $F4$. Then, we present the ND points that are ranked as top-three based on applying that function. It shows how significant improvements in terms of selecting balanced ND points are obtainable by applying those functions on the generated ND points.

One of suitable graph types for comparing several solutions at the same time in terms of sustainability considerations is radar graph. To deliver that comparison, we display Fig. 4.1. We have mentioned the performance of three ND points that are only focused on economic, environmental, or social concerns (i.e., (100, 29.43, 32.34), (6.59, 100, 4.5), and (67.61, 60.65, 100)). Figure 4.1a represents a comparison on those three points (denoted by red, grey, and green) and the ND point found based on $F1$ colored in blue. The ND points found based on $F2-F4$ are shown in blue in Fig. 4.1b–d, respectively. A decision maker finds out practical information at one look using Fig. 4.1 to convince his/her manager on the positive obtainable outcomes of applying sustainability-oriented solutions. Note that

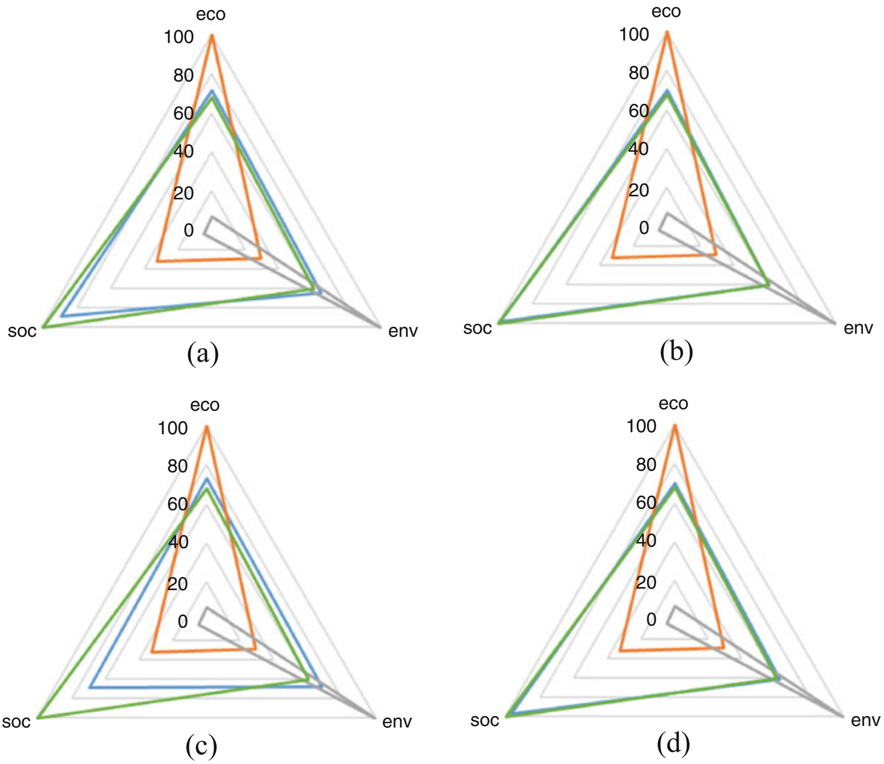

Fig. 4.1 The best ND points based on $F1$, $F2$, $F3$, and $F4$ in comparison with only economic/environmental/social-oriented points

in Fig. 4.1b, d where $F2$ and $F4$ are considered, (67.61, 60.65, 100) also results in a good performance close to the blue lines. Apparently, the final decision will be made by top-level decision makers with respect to the company's principles.

We have presented two parameters in the SAP model that directly influence on the social considerations: l_O (maximum allowed overtime per worker in one period) and α (Cycle Service Level). The values of the most of the parameters we have discussed about are defined based on their natures and/or technical limitations. However, both l_O and α are defined on the company's decision and its internal considerations. When we decrease the value of l_O (or increase the value of α), solutions with a higher social performance can be obtained, and then, we are interested in the sensitivity of the economic and environmental objective functions after that change. Figure 4.2 elaborates on these deviations by changing the value of α and l_O. We observe that these two objective functions react to l_O's changes almost linearly (Fig. 4.2c, d). However, their behavior is more sensitive to the changes of large values of α (Fig. 4.2a, b).

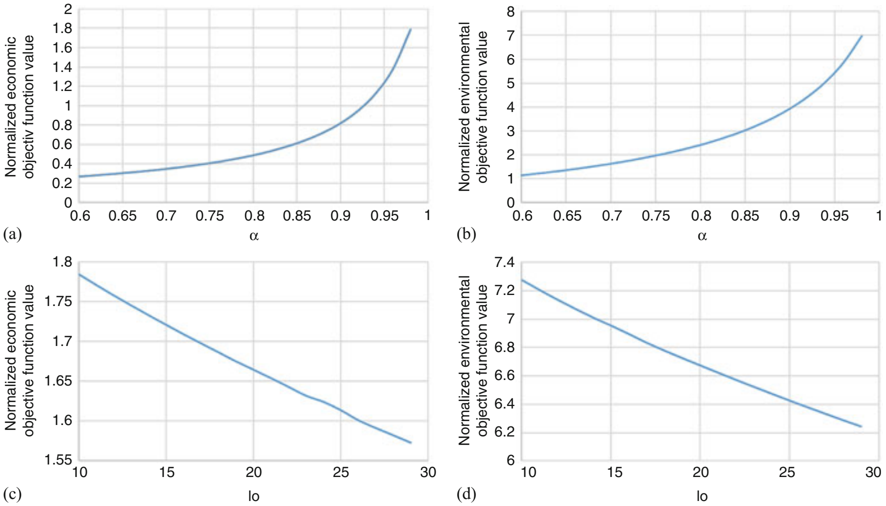

Fig. 4.2 Sensitivity analysis on the economic and environmental objective function values when either l_O or α changes

The approach provided in this section is a simple way to show that there are a large number of methods to consider multiple objective functions in a decision making process. In general, we present the following managerial insights:

- There are an infinite number of ND points in a SAP problem.
- There exist a large number of strategies that result in ND points.
- Generating these ND points and strategies is supportive for decision making process and informative for decision makers.
- As originated from the definition of a ND point, none of them dominates the others; however, there are pros and cons for every single ND point.
- Decision makers should note that focusing on the ND points which provide an unbalance performance is not a sustainable aggregate plan.
- In many cases, accepting slight negative changes in the total profit results in significant improvements and benefits in terms of sustainability.

4.2 Conclusions

Aggregate planning is a preferred mathematical programming approach that provides a solution to intermediate level decisions in the supply chains. One may customize the traditional aggregate planning problem to consider several pillars in one mathematical model which is MILP in most cases. We address training newly hired labors, production fixed cost in one period, promotion programs, innovative projects, electricity consumption, GHG emissions, etc.

When only one objective function exists, the interpretation of the optimal solution of an aggregate planning problem is straightforward. In these cases, the objective function is mainly calculated based on monetary concerns, and hence, an optimal solution that is acceptable by decision makers can be directly applied for the organization. However, a decision maker may decide to include multiple criteria in an aggregate plan. Therefore, the previous single objective MILP is converted to a MOMILP, and three major issues appear:

1. The final solution will not be a single solution; there will be a set of efficient solutions that result in ND points in the objective space.
2. The MOMILP is to be solved either entirely or partially such that all ND points or a subset of them are generated. The existing solution methods for MOMILPs are more complicated than the solution methods of single objective MILPs.
3. Finding all ND points is not sufficient for decision making. Making a decision based on a large number of efficient solutions or ND points requires comprehensive analyses.

References

1. Fattahi A, Türkay M (2018) A one direction search method to find the exact nondominated frontier of biobjective mixed-binary linear programming problems. Eur J Opers Res 266:415–425
2. Nash J (1953) Two-person cooperative games. Econometrica 21(1):128–140
3. Rasmi SAB, Türkay M (2019) GoNDEF: an exact method to generate all non-dominated points of multi-objective mixed-integer linear programs. Optim Eng 20(1):89–117
4. Rasmi SAB, Fattahi A, Türkay M (2021) SASS: slicing with adaptive steps search method for finding the non-dominated points of tri-objective mixed-integer linear programming problems. Ann Oper Res 296:841–876
5. Rasmi SAB, Kazan C, Türkay M (2019) A multi-criteria decision analysis to include environmental, social, and cultural issues in the sustainable aggregate production plans. Comput. Ind Eng 132:348–360

Correction to: Aggregate Planning

Correction to: S. A. B. Rasmi, M. Türkay, *Aggregate Planning*, SpringerBriefs in Operations Research, https://doi.org/10.1007/978-3-030-58118-3

Author's name is reflected incorrectly in the book. The correct spelling is Seyyed Amir Babak, Rasmi.
We note that the concern is on the meta PDF, where part of the given name "Amir" has been captured in "Particle" tag.
The citation has been updated as "Rasmi, S. A. B., Türkay…"

The updated version of this book can be found at
https://doi.org/10.1007/978-3-030-58118-3

S. A. B. Rasmi, M. Türkay, *Aggregate Planning*, SpringerBriefs in Operations Research, https://doi.org/10.1007/978-3-030-58118-3_5