algorithms

MDPI

Article

An Adaptive Linear Programming Algorithm with Parameter Learning

Lin Guo [1], Anand Balu Nellippallil [2], Warren F. Smith [3], Janet K. Allen [4,*] and Farrokh Mistree [4]

[1] Department of Industrial Engineering, South Dakota School of Mines and Technology, Rapid City, SD 57701, USA; lin.guo@sdsmt.edu
[2] Department of Mechanical and Civil Engineering, Florida Institute of Technology, Melbourne, FL 32901, USA; anellippallil@fit.edu
[3] Engineering and Technology, University of New South Wales, Canberra 2600, Australia; w.smith@unsw.edu.au
[4] The Systems Realization Laboratory, The University of Oklahoma, Norman, OK 73019, USA; farrokh.mistree@ou.edu
* Correspondence: janet.allen@ou.edu; Tel.: +1-404-403-3296

Abstract: When dealing with engineering design problems, designers often encounter nonlinear and nonconvex features, multiple objectives, coupled decision making, and various levels of fidelity of subsystems. To realize the design with limited computational resources, problems with the features above need to be linearized and then solved using solution algorithms for linear programming. The adaptive linear programming (ALP) algorithm is an extension of the Sequential Linear Programming algorithm where a nonlinear compromise decision support problem (cDSP) is iteratively linearized, and the resulting linear programming problem is solved with satisficing solutions returned. The reduced move coefficient (RMC) is used to define how far away from the boundary the next linearization is to be performed, and currently, it is determined based on a heuristic. The choice of RMC significantly affects the efficacy of the linearization process and, hence, the rapidity of finding the solution. In this paper, we propose a rule-based parameter-learning procedure to vary the RMC at each iteration, thereby significantly increasing the speed of determining the ultimate solution. To demonstrate the efficacy of the ALP algorithm with parameter learning (ALPPL), we use an industry-inspired problem, namely, the integrated design of a hot-rolling process chain for the production of a steel rod. Using the proposed ALPPL, we can incorporate domain expertise to identify the most relevant criteria to evaluate the performance of the linearization algorithm, quantify the criteria as evaluation indices, and tune the RMC to return the solutions that fall into the most desired range of each evaluation index. Compared with the old ALP algorithm using the golden section search to update the RMC, the ALPPL improves the algorithm by identifying the RMC values with better linearization performance without adding computational complexity. The insensitive region of the RMC is better explored using the ALPPL—the ALP only explores the insensitive region twice, whereas the ALPPL explores four times throughout the iterations. With ALPPL, we have a more comprehensive definition of linearization performance—given multiple design scenarios, using evaluation indices (EIs) including the statistics of deviations, the numbers of binding (active) constraints and bounds, the numbers of accumulated linear constraints, and the number of iterations. The desired range of evaluation indices (DEI) is also learned during the iterations. The RMC value that brings the most EIs into the DEI is returned as the best RMC, which ensures a balance between the accuracy of the linearization and the robustness of the solutions. For our test problem, the hot-rolling process chain, the ALP returns the best RMC in twelve iterations considering only the deviation as the linearization performance index, whereas the ALPPL returns the best RMC in fourteen iterations considering multiple EIs. The complexity of both the ALP and the ALPPL is $O(n^2)$. The parameter-learning steps can be customized to improve the parameter determination of other algorithms.

Keywords: linear programming; adaptive linear programming; rule-based; parameter learning

Citation: Guo, L.; Nellippallil, A.B.; Smith, W.F.; Allen, J.K.; Mistree, F. An Adaptive Linear Programming Algorithm with Parameter Learning. *Algorithms* **2024**, *17*, 88. https://doi.org/10.3390/a17020088

Academic Editor: Binlin Zhang

Received: 24 December 2023
Revised: 8 February 2024
Accepted: 11 February 2024
Published: 19 February 2024

1. Introduction

Solution algorithms for solving engineering design problems fall into two categories: (i) an optimizing strategy or (ii) a satisficing strategy. For typical engineering design problems with nonlinear, nonconvex properties that cannot be solved in a straightforward way using simplex, Lagrange multipliers, or other gradient-based solution algorithms, when applying the optimizing strategy, designers need to solve problems approximately, for example, using pattern search methods [1], particle swarm optimization [2], memetic algorithms [3], penalty function methods [4], or metaheuristic algorithms [5]. When methods in Category (i) are used, the solutions are desired to be near-Pareto solutions, often on or close to the boundary of the feasible space (bounded by constraints) but not necessarily at a vertex of the feasible space. Whereas when methods in Category (ii) are used, such as sequential linear programming [6] or FEAST [7], the solutions are always at one or multiple vertices of the linearized (approximated) solution space. Since the solution always includes a vertex, we are able to use information of the dual problem to explore the solution space [6]. Typically, in engineering design problems the number of constraints in the primal formulation exceeds the number of variables [8]. In these cases, solving the dual formulation is quicker [6]. For optimization strategies, there are state-of-the-art surveys summarizing and classifying linearization techniques thoroughly, such as in [9]. Linearization techniques are classified into two groups—nonlinear equations replaced by an exact equivalent LP formulation or linear approximations which find the equivalent of a nonlinear function with the least deviation around the point of interest or separate straight-line segments.

In this paper, our focus is on a method in Category (ii), the satisficing strategy, namely, the compromise decision support problem (cDSP) and adaptive linear programming (ALP) algorithm [6], because it is more appropriate for engineering design problems by giving satisficing solutions and exploring solution space. Satisficing solutions are near-optimal and on the vertices of the approximated linear problems. The reasons are summarized in Sections 1.1 and 1.2.

1.1. Frame of Reference

There are special requirements for engineering design methods. Engineering design is a task that occupies multiple designers with different fields of expertise to make decisions that meet a variety of requirements [10]. Designers may define objective functions using different methods—utility theory [11], game theory [12], analytic hierarchy process (AHP) [13], Pareto-optimal methods [14], etc. For optimization problems with one or multiple objective function(s) to be minimized or maximized, one can define such problems based on maximization of expected utility. Designers accept assumptions and simplifications when using utility theory to manage a problem, sometimes without realizing it. Such assumptions include the abstraction of mathematical relationships between variables to be 100% accurate and not evolve, the levels of fidelity of different sub-models or segments of a model are the same, the design preferences among multiple objectives are fully captured, etc. [15]. For many problems, these assumptions are wrong and this may cause problems. For example, when implementing an optimal solution, a system may not give optimal output, because of the inaccurate equations used in the model, or because uncertainty breaks the equilibrium of any Karush–Kuhn–Tucker (KKT) conditions, thus destroying the optimality of the solution [16].

In addition, when the problem is nonconvex, specifically, when any nonzero linear combination of the constraints is more convex than the objective function [17], Category (i) methods cannot return a feasible solution because the sufficient KKT condition cannot be satisfied [15]. In this case, formulating the problem using the compromise decision support problem (cDSP) and solving it using the adaptive linear programming (ALP) algorithm is a way to identify satisficing solutions that meet the necessary KKT condition [15].

1.2. Mechanisms to Ensure That the cDSP and ALP Find Satisficing Solutions

There are five mechanisms in the cDSP and ALP that enable designers to identify satisficing solutions to nonlinear, nonconvex engineering design problems, especially when optimization methods fail. We summarize the mechanisms in Table 1. Two assumptions in standard optimization methods make optimal solutions unavailable for some nonlinear, nonconvex problems:

Assumption 1. *Mathematical models are 100% complete and accurate abstractions of physical models, so the optimal solution to the mathematical problem is optimal for the physical problem.*

Assumption 2. *The convexity degree of at least one nonzero linear combination of all constraints is higher than the convexity degree of the objective function.*

However, for Assumption 1, designers need boundary information to deal with uncertainties, but such information is normally unavailable [18]. Using the cDSP and ALP, designers can identify satisficing solutions by removing the two assumptions.

Table 1. The mechanisms in the cDSP and ALP to find satisficing solutions [16].

Mechanisms	Advantage	Assumption Removed
Using goals and minimizing deviation variables instead of objectives	At a solution point, only the necessary KKT condition is met, whereas the sufficient KKT condition does not have to be met. Therefore, designers have a greater chance of finding a solution and a lower chance of losing a solution due to parameterizable and/or unparameterizable uncertainties.	Assumption 1
Using second-order sequential linearization	Designers can have a balance between linearization accuracy and computational complexity.	Assumption 2
Using accumulated linearization	Designers can manage nonconvex problems and deal with highly convex, nonlinear problems relatively more accurately.	Assumption 2
Combining interior-point search and vertex search	Designers can avoid getting trapped in local optima to some extent and identify satisficing solutions which are relatively insensitive when the starting points change.	Assumption 1
Allowing some violations of soft requirements, such as the bounds of deviation variables	Designers can manage rigid requirements and soft requirements in different ways to ensure feasibility. As a result, goals and constraints with different scales can be managed.	Assumptions 1 and 2

There are differences between the satisficing construct using cDSP-ALP and optimization including its variant goal programming. Solutions are usually obtained by identifying the Pareto frontier consisting of nondominated or near-optimal solutions using optimization solution algorithms. The formulation of design problems using a satisficing strategy, namely, the compromise decision support problem (cDSP), has the key features that allow designers to identify satisficing solutions that meet the necessary KKT condition but not the sufficient condition.

The format of a nonlinear optimization problem is this: for a given objective function $f(x)$, Euler and Lagrange developed the Euler–Lagrange equation forming the second-order ordinary differential equations $\nabla^2_{xx} f(x)$ to facilitate finding the stationary solutions. The value of the variables that maximize $f(x)$ within the feasible set $\mathcal{F}$ is the solution to the optimization problem, where $\mathcal{F}$ is the set bounded by constraints and bounds. The format of an optimization problem $\mathbb{O}$ can be represented as follows. x is the vector of decision variables as real numbers. $g_i(x)$ is the i^{th} inequality constraint. $h_j(x)$ is the i^{th} equality constraint. Any point x that is a local extremum of the set mapped by multiplying active

equations with a nonnegative vector is a local optimum of $\mathbb{O}$ [19], denoted as x^*. The elements of such a nonnegative vector are Lagrange multipliers, μ and λ.

The format of an optimization problem $\mathbb{O}$:

Given

$$f:\ \mathbb{R}^n \to \mathbb{R}, \mathcal{F} \subseteq \mathbb{R}^n$$
$$\mathcal{F} = \left\{x \in \mathbb{R}^n \middle| g_i(x) \geq 0,\ i = 1, \ldots, m, h_j(x) = 0,\ j = 1, \ldots, \ell\right\}$$

find

$$x^*:\ f(x^*) \succcurlyeq f(x),\ \forall x \in \mathcal{F}$$

One variant of optimization is goal programming. The format of a goal programming problem $\mathbb{O}^{goal}$ is represented as follows. A target value T is predefined for the objective function $f(x)$ as the right-hand side value, so the objective becomes an equation, and we call it a goal. d^- and d^+ are deviation variables measuring the underachievement and overachievement of the goal towards its target. The problem is solved by minimizing the deviation variables, which is minimizing the difference between $f(x)$ and T. In other words, goal programming is aimed at finding T's closest projection on $\mathcal{F}$.

The format of a goal programming problem $\mathbb{O}^{goal}$:

Given

$$f:\ \mathbb{R}^n \to \mathbb{R}, \mathcal{F} \subseteq \mathbb{R}^n$$
$$\mathcal{F} = \left\{x \in \mathbb{R}^n \middle| g_i(x) \geq 0,\ i = 1, \ldots, m, h_j(x) = 0,\ j = 1, \ldots, \ell, d^- \cdot d^+ = 0,\ 0 \leq d^{\mp} \leq 1, Goal:\ f(x) + d^- - d^+ = T\right\}$$

find

$$x^*: \mathbb{P}_{x \in \mathcal{F}}(Goal:\ f(x) = T)$$

In the cDSP, elements of mathematical programming and goal programming are combined. A cDSP $\mathbb{C}$ is represented as follows. For a nonlinear cDSP, we first linearize the nonlinear equations, including nonlinear constraints and nonlinear goal. Therefore, the nonlinear cDSP first becomes a linear problem with a linear goal $Goal^{linear}$, a linear feasible space $\mathcal{F}^{li}$ bounded by linear constraints $g(x)^{li} \geq 0$ and $h(x)^{li} = 0$. Thus, using a cDSP, we seek the closest projection from the linear goal set onto a linear feasible set. We define the solution as a satisficing solution, and we use x^s to denote it.

The format of a cDSP $\mathbb{C}$:

Given

$$f:\ \mathbb{R}^n \to \mathbb{R}, \mathcal{F} \subseteq \mathbb{R}^n$$
$$\mathcal{F}^{li} = \left\{x \in \mathbb{R}^n \middle| g_i(x)^{li} \geq 0,\ i = 1, \ldots, m, h_j(x)^{li} = 0,\ j = 1, \ldots, \ell, d^- \cdot d^+ = 0,\ 0 \leq d^{\mp} \leq 1, Goal^{li}: \frac{f(x)^{li}}{T} + d^- - d^+ = 1\right\}$$

find

$$x^s: \mathbb{P}_{x \in \mathcal{F}^{li}}\left(Goal^{li}: f(x)^{li} = T\right)$$

The difference between x^* and x^s is that x^* conforms to both the necessary (first-order) and sufficient (second-order) KKT conditions, whereas x^s conforms to the necessary KKT condition but may not conform to the sufficient KKT condition. This is because the second derivative of the linear equations, $Goal^{li}$, $g_i(x)^{li} \geq 0$, and $h_j(x)^{li} = 0$, degenerates—as a result, no uncertainty may affect the feasibility of x^s because there is no uncertainty to break the equilibrium of the second-order Lagrange equation. In addition, when the convexity of $\mathcal{F}$ is greater than the convexity of the $f(x)$, x^* may not be identified as the second-order Lagrange equation has no solution, but x^s is obtainable because the second-order KKT condition is irrelevant. The relationship among optimal solutions, satisficing solutions, and near-optimal solutions is illustrated in Figure 1.

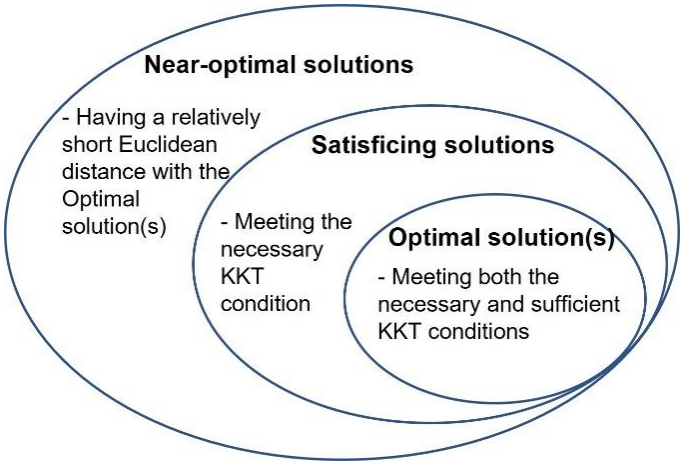

Figure 1. The relationship between the optimal, satisficing, and near-optimal solutions.

In Chapter 2 of [15], the author gives five examples to demonstrate how the mechanisms work (we have a demonstration of the comparison of five examples in this video: https://www.youtube.com/watch?v=7apDZO-9A74, accessed on 21 November 2020, and a tutorial of using DSIDES to formulate and solve a problem in this video: https://www.youtube.com/watch?v=tUpVC97Y1L8, accessed on 25 November 2020. In this paper, since our focus is to fill a gap in the ALP, we give only one example in Section 1.3 to show one of the advantages versus the optimization method. Information such as applications using cDSP and ALP to obtain satisficing solutions can be found in [20].

1.3. An Example and Explanation Using KKT Conditions

We use a simple example with multiple goals (objectives), nonlinear and nonconvex equations, and the goal targets with various degrees of achievability. We formulate the problem using optimization and the cDSP in Table 2.

Table 2. The optimization model and compromise DSP of the example.

Optimizing	Satisficing
	Given
	$x_1,\ x_2,\ {d_1}^{\pm},\ {d_2}^{\pm}$
	$f_1(x) = \cos\left({x_1}^2 + {x_2}^3\right)$
	$f_2(x) = 25{\cdot}(x_1 - 2)^3 + 50{\cdot}(x_2 - 2)^3 + 50{\cdot}x_1{\cdot}{x_2}^2$
Objective Functions	Find
	$x_1,\ x_2,\ {d_1}^{\mp}, {d_2}^{\mp}$
$f_1(x) = \cos\left({x_1}^2 + {x_2}^3\right)$	Satisfy
$f_2(x) = 25{\cdot}(x_1 - 2)^3 + 50{\cdot}(x_2 - 2)^3 + 50{\cdot}x_1{\cdot}{x_2}^2$	Goals:
Constraints and Bounds	$\frac{f_1(x)}{1.2} + {d_1}^- - {d_1}^+ = 1$
$\text{s.t.} \begin{cases} x_1{\cdot}x_2 \le 1 \\ f_1(x) \ge 0 \\ f_2(x) \ge 0 \\ 0 \le x_1 \le 2 \\ 0 \le x_2 \le 2 \end{cases}$	$\frac{f_2(x)}{400} + {d_2}^- - {d_2}^+ = 1$
	Constraints:
	$x_1{\cdot}x_2 \le 1$
	$f_1(x) \ge 0$
	$f_2(x) \ge 0$
	${d_i}^-{\cdot}{d_i}^+ = 0, i = 1,\ 2$
Combination of Objective Functions	Bounds:
$\text{Max } \sum_{i=1}^{2} w_i{\cdot}f_i(x)$	$0 \le x_1,\ x_2 \le 2$
	$0 \le {d_1}^{\pm},\ {d_2}^{\pm} \le 1$
	Minimize
	Merit function $Z = \sum_{i=1}^{2} w_i{\cdot}\left({d_i}^- + {d_i}^+\right)$

The methods used in the optimizing and satisficing strategies are listed in Table 3. For optimizing methods, we used the "SciPy.optimize" package (https://docs.scipy.org/doc/scipy/reference/optimize.html, accessed on 1 January 2008). There are ten algorithms in the package. We use three of them to solve the example problem because they are the only relevant ones. We do not use the Nelder–Mead, Powell (Powell's conjugate direction method) [21], or conjugate gradient (CG) methods [22] or the Broyden–Fletcher–Coldfarb–Shannon (BFGS) algorithm [23] because they cannot easily manage problems with constraints. The Newton conjugate gradient (Newton-CG) method [24,25], L-BFGS-B (an extension BFGS for large-scale, bounded problems) [26], and TNC (truncated Newton method or Hessian-free optimization) [26] either cannot deal with problems without Jacobians (when using Newton-CG, even setting the Jacobian as false, and the algorithm may not work without a provided Jacobian partially because the default temporary memory of Jacobian cannot be cleared; see: https://stackoverflow.com/questions/33926357/jacobian-is-required-for-newton-cg-method-when-doing-a-approximation-to-a-jaco, accessed on 15 May 2019) or return infeasible solutions without recognizing that they are infeasible. Therefore, in SciPy we use the constrained optimization by linear approximation (COBYLA) algorithm [27,28], Trust-constr (https://docs.scipy.org/doc/scipy/reference/optimize.minimize-trustconstr.html, accessed on 1 January 2008), and sequential least squares programming (SLSQP) (https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html, accessed on 1 January 2008) to solve the example problem.

Table 3. Methods for comparison of the two strategies.

Item \ Strategy	Optimizing	Satisficing
Model formulation construct	Mathematical programming Goal programming	Compromise decision support problem
Solution algorithm	Constrained optimization by linear approximation (COBYLA) algorithm	Adaptive linear programing (ALP) algorithm
	Trust-region constrained (trust-constr) algorithm	
	Sequential least squares programming (SLSQP) algorithm	
	Nondominated sorting generation algorithm II/III (NSGA II)	
Solver	Python SciPy.optimize	DSIDES [6]

We select nondominated sorting genetic algorithm II (NSGA II) [29] as a verification method to compare and evaluate the performance of our selected optimization methods and the satisficing method in Table 3. We choose NSGA II as the verification method because it can solve problems with the complexities of the example problem in Table 2—nonlinearity, nonconvexity, multiple objectives or goals, and various achievability of the goals. We use NSGA II in MATLAB. Some readers may wonder, since NSGA II can solve the complexities often incorporated in engineering design problems, why do we study a satisficing algorithm, the cDSP, and ALP to manage engineering design problems? The reason is that we observe that NSGA II has the following drawbacks that may prevent designers from acquiring insight to improve the design formulation and exploring the solutions space:

First, NSGA II cannot give designers information to improve the model, such as the bottlenecks in the model, the sensitivity of each segment of the model, the rationality of the dimensions of the model, etc.

Second, the performance of NSGA II, including convergence speed, optimality of solutions, and diversity of solutions, is sensitive to hyperparameter settings. Hyperparameters, such as the population size and generation number, must be predefined. However, usually designers only assume that a larger population size or a larger number of generations returns better solutions, but they may not know how large is "good enough". Designers

need to tune those hyperparameters in NSGA II, but it requires much higher computational power than the cDSP and ALP do.

Therefore, we choose NSGA II in order to assess the optimality and diversity of the solutions returned by the tested methods (in Table 3), but we still recommend designers use the satisficing method to manage engineering design problems.

Through applying the three chosen optimizing methods, the satisficing method, and the verification method, NSGA II, we obtain the results and summarize them in Table 4. Since there are two objectives (goals), we use different weights to combine them linearly, so we show the results for each weight. We also use different starting points for the solution searching to show whether a solution algorithm is sensitive to the starting point. We visualize the solutions in objective space and x-f(x) space in Figures 2 and 3. The three optimization algorithms cannot return any feasible solutions. Due to the difference in the scale of the two objectives, one cannot use optimizing algorithms to solve the example problem by linearly combining them, because (i) the objective with a large scale dominates the other objective(s) and (ii) a linearized function of the weighted sum objective in a local area can be singular.

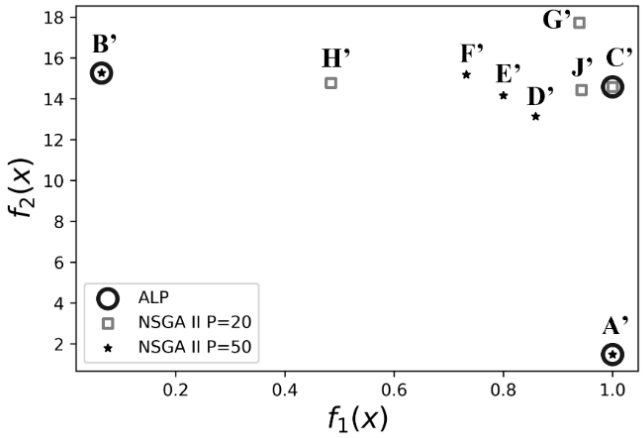

Figure 2. The solution points to the example problem in objective space using two algorithms—solutions returned by NSGA II are more diverse but sensitive to parameter settings and increasing the population and iterations does not always produce better results.

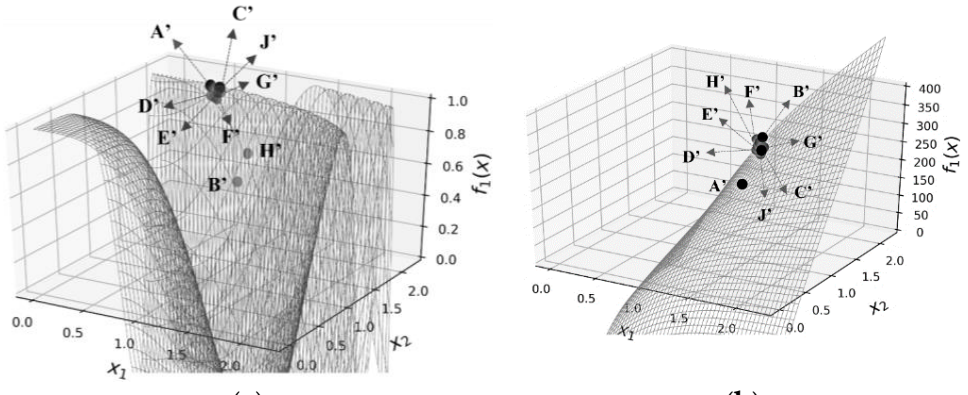

Figure 3. The solution points in the x-f(x) space. Subfigure (**a**) is the x-f_1(x) space and Subfigure (**b**) is the x-f_2(x) space. All solutions' f_1(x) values are close to 1.0 except B′ and H′; all solutions' f_2(x) values are close to 400 except A′ [15].

Table 4. Solutions to the example problem—dominated solutions for each weight are in *italics*; for each method, the best solution of each design scenario is marked using a capital letter (A′, B′, C′, D′, E′, F′, G′, H′, and J′) [15].

Weight	Starting Point	COBYLA		Trust-Constr		SLSQP		ALP		NSGA II/III—Population (P) = 20/50	
		Solution	$\sum_{i=1}^{2} w_i \cdot f_i(x)$	Solution	$\sum_{i=1}^{2} w_i \cdot f_i(x)$	Solution	$\sum_{i=1}^{2} w_i \cdot f_i(x)$	Solution	$\sum_{i=1}^{2} w_i \cdot f_i(x)$	Solution	$\sum_{i=1}^{2} w_i \cdot f_i(x)$
(1, 0)	(0.5, 1)	Cannot manage nonconvex equations with bounds		All solutions violate one or more constraints		All solutions violate one or more constraints		A′ (0.51,1.82)	1	P = 20: G′ (0.55, 1.85) P = 50: A′ (0.51, 1.82)	P = 20: 0.99 P = 50: 1
	(0, 0)										
	(2, 0.5)										
(0, 1)	(0.5, 1)							B′ (0.51, 1.96)	15.27	P = 20: H′ (0.52, 1.92) P = 50: B′ (0.51, 1.96)	P = 20: 14.86 P = 50: 15.36
	(0, 0)										
	(2, 0.5)										
(0.5, 0.5)	(0.5, 1)							*C′ (0.55, 1.82)*	*7.5*	P = 20: H′ (0.52, 1.92) P = 50: D′ (0.53, 1.87)	P = 20: 7.69 P = 50: 7.78
	(0, 0)										
	(2, 0.5)										
(0.7, 0.3)	(0.5, 1)							*C′ (0.55, 1.82)*	*4.9*	P = 20: C′ (0.55, 1.82) P = 50: E′ (0.53, 1.88)	P = 20: 4.9 P = 50: 5.01
	(0, 0)										
	(2, 0.5)										
(0.3, 0.7)	(0.5, 1)							*C′ (0.55, 1.82)*	*10.01*	P = 20: J′ (0.54, 1.85) P = 50: F′ (0.53, 1.89)	P = 20: 10.49 P = 50: 10.6
	(0, 0)										
	(2, 0.5)										

For a multi-objective (multi-goal) problem with nonconvex functions, when the scale of the objectives varies largely, optimizing algorithms cannot return feasible solutions, whereas a satisficing strategy may allow designers to identify adequate solutions. This is explained using the KKT conditions hereafter.

When using optimizing algorithms to solve optimization problems, the first-order derivative of the Lagrange equation with respect to decision variable x, which is a function of the parameters $\mathcal{P}$ of the model (the coefficients in objectives and constraints), decision variables x (if any objective or constraint is nonlinear), Lagrange multipliers μ and λ, and weights $\mathscr{P}$, combining the multiple goals, is shown in Equation (1).

$$\nabla_x L(x, \mu, \lambda) = \mathscr{Y}(\mathcal{P}, x, \mu, \lambda, \mathscr{P}) \tag{1}$$

For a satisficing strategy, the first-order derivative of the Lagrange equation contains only the coefficients of the deviation variables in the objective, since only deviation variables $d^{\mp}$ constitute the objective (not the decision variables, x). For a κ-goal cDSP with m inequality constraints g(x) and ℓ equality constraints h(x), if we use weights to combine the κ goals G(x,d), i.e., using the Archimedean strategy to manage a multi-goal cDSP, then the coefficients in the first-order Lagrange equation are only the weights and τ—τ is the Lagrange multiplier of the goal functions; see Equation (2).

$$\begin{aligned}\nabla_d L(x^s, d, \mu, \lambda, \tau) &= \nabla_d z(d) + \sum_{i=1}^{m} \mu_i \nabla_d g_i(x^s) - \sum_{j=1}^{\ell} \lambda_j \nabla_d h_j(x^s) - \sum_{k=1}^{\kappa} \tau_k \nabla_d G(x^s, d) \\ &= \mathscr{Y}(\mathscr{P}, \tau)\end{aligned} \tag{2}$$

when using optimization, the second-order Lagrange equation may still have parameters and decision variables due to nonlinearity; see Equation (3). For satisficing, the second-order Lagrange equation with respect to deviation variables degenerates to zero because the objective of a cDSP is a linear combination of deviation variables; see Equation (4). That is why satisficing solutions do not need to meet the second-order KKT conditions.

$$\nabla^2_{xx} L(x, \mu, \lambda) = \nabla_x \mathscr{Y}(\mathcal{P}, x, \mu, \lambda, \mathscr{P}) = \mathscr{Y}'(\mathcal{P}', x) \tag{3}$$

$$\nabla^2_{dd} L(x^s, d, \mu, \lambda, \tau) = \nabla_d (\mathscr{Y}(\mathscr{P}, \tau)) \equiv 0 \tag{4}$$

Although both optimal solutions x^* and satisficing solutions x^s meet the first-order KKT conditions, the chance of maintaining the first-order KKT conditions for the two strategies under uncertainties varies. If any uncertainty with probability P takes place to an item $\Im$ in the first-order equation that destroys its equilibrium, we denote it as $\Pr\left(\widetilde{\Im} \,|P\right)$. For an N-dimension, Q-parameter (here, we define a coefficient or an intercept of a constraint or an objective as a parameter. A parameter has a given value (either a constant value or a stochastic value) and the value does not depend on any decision variables), and κ-goal problem, using optimizing strategy, the source of $\widetilde{\Im}$ can be decision variables $\widetilde{x_n}$, Lagrange multipliers $\widetilde{\mu_i}$ and $\widetilde{\lambda_j}$, and weights $\widetilde{\mathscr{P}_k}$; for satisficing strategy, the source of $\widetilde{\Im}$ can only be the weights $\widetilde{\mathscr{P}_k}$ and the Lagrange multipliers for the goals $\widetilde{\tau_k}$. If and only if none of the items under the uncertainty breaks the equilibrium of the first-order equation, then the optimal/satisficing solution is still optimal/satisficing under this uncertainty. For an N-dimension, Q-parameter, and κ-goal problem, the probabilities of maintaining an optimal solution and a satisficing solution under Uncertainty P are given in Equations (5) and (6), respectively.

$$\Pr(x^* \,|P) \approx \prod_{q=1}^{Q}\left[1 - \Pr\left(\widetilde{\mathcal{P}_q} \,|P\right)\right] \prod_{n=1}^{N}[1 - \Pr(\widetilde{x_n} \,|P)] \prod_{i=1}^{m}[1 - \Pr(\widetilde{\mu_i}|P)] \prod_{j=1}^{\ell}\left[1 - \Pr(\widetilde{\lambda_j}\middle|P)\right] \prod_{k=1}^{\kappa}\left[1 - \Pr(\widetilde{\mathscr{P}_k}\middle|P)\right] \tag{5}$$

$$\Pr(x^s|P) \approx \prod_{k=1}^{\kappa}\left[1 - \Pr(\widetilde{\mathscr{P}_k}\middle|P)\right] \prod_{k=1}^{\kappa}[1 - \Pr(\widetilde{\tau_k}|P)] \tag{6}$$

As the value of any probability is in the range of [0, 1], the more items on the right-hand side we multiply (the more items the probability depends on), the lower the probability becomes. The items in Equation (6) are fewer than those of Equation (5). Hence, the chance

of maintaining an optimal solution under Uncertainty P is often smaller than the chance of maintaining a satisficing solution with the same uncertainty; Equation (7).

$$\Pr(x^* | P) \leq \Pr(x^s | P) \tag{7}$$

In summary, using the satisficing strategy, we are less likely to lose a solution due to uncertainty for nonconvex problems with multiple objectives that have various scales; see Equations (8) and (9). Using satisficing, designers can deal with nonconvex, multi-objective problems that may be incomplete or inaccurate and with uncertainties, which helps remove Assumptions 1 and 2.

$$\left[\Pr\left(\mathcal{S} \neq \varnothing | \left\{\begin{array}{c}\text{Nonconvexity}\\ \text{objectives with various scales}\end{array}\right\}\right)\right]_{\text{Satisficing}} \geq$$

$$\left[\Pr\left(\mathcal{S} \neq \varnothing | \left\{\begin{array}{c}\text{Nonconvexity}\\ \text{objectives with various scales}\end{array}\right\}\right)\right]_{\text{Optimizing}} \tag{8}$$

$$[\Pr(x^s|P)]_{\text{Satisficing}} \geq [\Pr(x^*|P)]_{\text{Optimizing}} \tag{9}$$

1.4. A Limitation in the ALP to Be Improved

Just like NSGA II, the ALP also has a parameter, the reduced move coefficient, or RMC, that may impact the solution space, especially for multi-goal problems. The problems as simple as the example problem in Section 1.3 is not sensitive to RMC setting, but more complicated problems are. We use a test problem of steel rod manufacturing to illustrate it in Section 4.

The RMC is a fractional step size [6] defining the starting point for the next iteration. In the ALP, the RMC has been set at 0.5 as a default value. In fact, in the ALP, a designer has no knowledge of the connection between the RMC and solution quality or the opportunity for improving solution efficiency by controlling the RMC. Improving the RMC determination is discussed in [30]. In this paper, we give more explanations about the motivation, the advantages of the satisficing strategy using the cDSP and ALP, and the benefits of parameter learning.

We introduce the ALP and discuss the limitation of using a fixed RMC in Section 2. In Section 3, we introduce parameter learning to make the RMC adaptive for each iteration. In Section 4, we use a test problem that is sensitive to the RMC value—the steel hot rod-rolling process—to demonstrate efficacy of the augmented ALP, that is, the adaptive linear programming algorithm with parameter learning (ALPPL). In Section 5, we summarize the contributions and comment on the generalization of parameter learning for use in gradient-based optimization methods.

2. How Does the ALP Work?

2.1. The Adaptive Linear Programming (ALP) Algorithm

The ALP algorithm is implemented in DSIDES. DSIDES is used to formulate and solve engineering design problems, and it is especially efficient in dealing with nonlinear problems [6]. In DSIDES, the nonlinear problem is formulated as a compromise decision support problem (cDSP). Then, the ALP is invoked to solve the nonlinear problem. The nonlinear problem is linearized in a synthesis cycle. The resulting linear problem is solved using the revised dual simplex algorithm. The synthesis cycle is repeated until the solution satisfies a set of stopping criteria or is terminated after a fixed number of iterations.

The ALP incorporates a local approximation algorithm [6,31], in which a secant plane of the paraboloid (with the second-order derivatives at the starting point as the coefficients) replaces the original nonlinear function. In Figure 4, we show two dimensions of a problem being approximated in two iterations (synthesis cycle). The weighted sum of the goals is $\sum_{k\in K} W_k \cdot G_k$. The starting point X_0^0 may not be in the feasible region. A random search or a

Hooke–Jeeves pattern search can be invoked to identify a point X_1^0 in the feasible area. In the i^{th} first iteration, the problem is linearized at X_1^0.

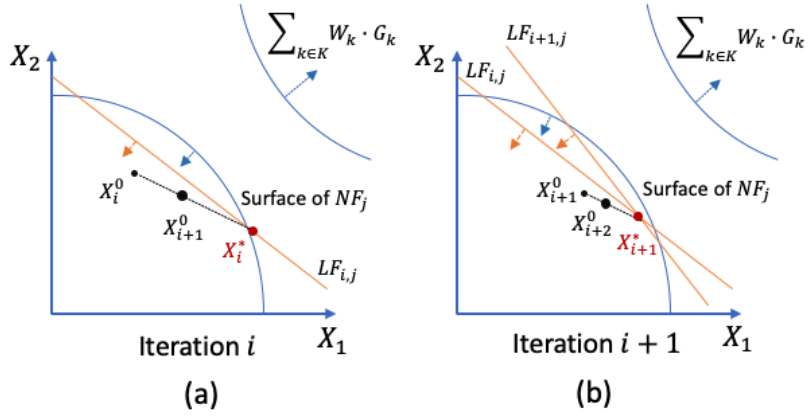

Figure 4. The approximation and solution using ALP in two iterations when RMC = 0.4. Subfigure (**a**) is Iteration i and Subfigure (**b**) is Iteration i + 1.

In iteration i, Figure 4a, a projection of a nonlinear constraint NF_j onto a two-dimensional plane, X_1-X_2, is approximated at X_i^0, so an approximated constraint $LF_{i,j}$ is obtained. Doing this for all nonlinear functions and framing a linear model, the revised simplex dual algorithm is used to obtain solution X_i^*. Using the RMC heuristic, we find the starting point of the next iteration X_{i+1}^0. In iteration i + 1, Figure 4b, the approximated linear constraints of both iterations $LF_{i,j}$ and $LF_{i+1,j}$ are accumulated, and a solution X_{i+1}^* is returned and the starting point of iteration i + 2 is again defined using the RMC, X_{i+1}^0 and X_{i+1}^*.

In Figure 5a, we illustrate the two-step linear approximation method. First, NF_j (Paraboloid ABC) is approximated to $NF_{i,j}''$ (Paraboloid AB^*C^*) with the diagonal terms of its Hessian matrix at X_i^0 as coefficients. Then, NF_j'' is approximated to a secant Plane $LF_{i,j}$ (Plane AB^*C^*). NF_j'' and $LF_{i,j}$ are computed as follows.

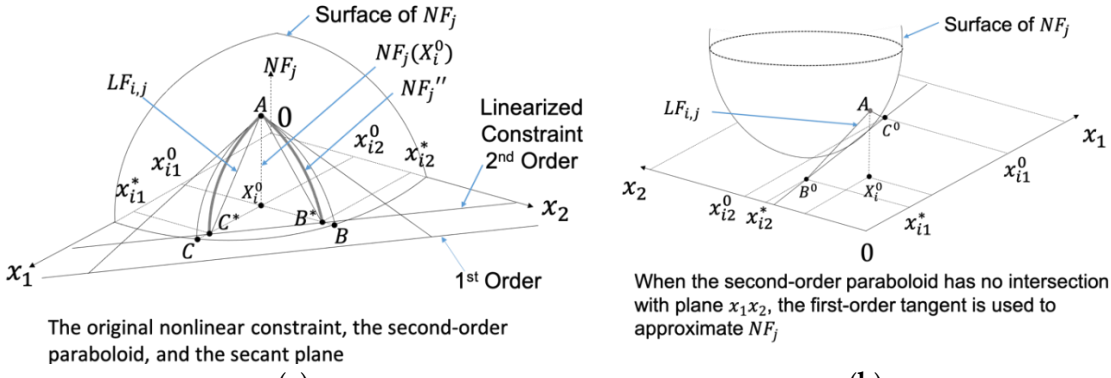

Figure 5. The two-step linear approximation methods using the ALP. Subfigure (**a**) illustrates the situation when the second-order paraboloid of the riginal nonlinear constraint NF_j has intersection with Plane x_1x_2, whereas Subfigure (**b**) illustrates the situation when the second-order paraboloid has no intersection with Plane x_1x_2 hence the first-order tangent is used to approximate NF_j.

$NF_{i,j}''$ is obtained using the second-order full derivatives at X_i^0, Equation (10), because the second-order partial derivatives have limited impact on the gradient.

$$NF_{i,j}'' = NF_j\left(X_i^0\right) + \sum_{p=1}^{n}\left(x_{ip} - x_{ip}^0\right)\left(\frac{\partial NF_j}{\partial x_{ip}}\right)_0 + \frac{1}{2}\sum_{p=1}^{n}\left(x_{ip} - x_{ip}^0\right)^2\left(\frac{\partial^2 NF_j}{\partial {x_{ip}}^2}\right)_0 \quad (10)$$

From Equation (10), for the p^{th} dimension, the quadratic to be solved to obtain $\left(x_{ip} - x_{ip}^0\right)$ is:

$$NF_j\left(X_i^0\right) + \left(x_{ip} - x_{ip}^0\right)\left(\frac{\partial NF_j}{\partial x_{ip}}\right)_0 + \frac{1}{2}\left(x_{ip} - x_{ip}^0\right)^2\left(\frac{\partial^2 NF_j}{\partial {x_{ip}}^2}\right)_0 = 0 \tag{11}$$

If Equation (11) has real roots, Figure 5a, by solving Equation (11) and selecting the root between Equations (12) and (13) with the smaller absolute value for each dimension, we obtain the intersection that is closer to the paraboloid in each dimension, such as B* and C*.

$$\left(\frac{\partial NF_j}{\partial x_{ip}}\right)^*_0 = \frac{-NF_j\left(X_i^0\right)\left(\frac{\partial^2 NF_j}{\partial {x_{ip}}^2}\right)_0}{-\left(\frac{\partial NF_j}{\partial x_{ip}}\right)_0 - \sqrt{\left(\frac{\partial NF_j}{\partial x_{ip}}\right)_0^2 - 2NF_j\left(X_i^0\right)\left(\frac{\partial^2 NF_j}{\partial {x_{ip}}^2}\right)_0}} \tag{12}$$

$$\left(\frac{\partial NF_j}{\partial x_{ip}}\right)^*_0 = \frac{-NF_j\left(X_i^0\right)\left(\frac{\partial^2 NF_j}{\partial {x_{ip}}^2}\right)_0}{-\left(\frac{\partial NF_j}{\partial x_{ip}}\right)_0 + \sqrt{\left(\frac{\partial NF_j}{\partial x_{ip}}\right)_0^2 - 2NF_j\left(X_i^0\right)\left(\frac{\partial^2 NF_j}{\partial {x_{ip}}^2}\right)_0}} \tag{13}$$

If Equation (11) has no real roots, Figure 5b, NF_i does not intersect Plane x, then the first-order derivative at X_i^0 is used, Equation (14).

$$\left(\frac{\partial NF_j}{\partial x_{ip}}\right)^*_0 = \left(\frac{\partial NF_j}{\partial x_{ip}}\right)_0 \tag{14}$$

Based on the intersections in each dimension, such as B* and C*, we obtain LF_i.

$$LF_{i,j} = \sum_{p=1}^{n} x_{ip}\left(\frac{\partial NF_j}{\partial x_{ip}}\right)^*_0 - \left(\sum_{p=1}^{n} x_{ip}^0\left(\frac{\partial NF_j}{\partial x_{ip}}\right)^*_0 - NF_j\left(X_i^0\right)\right) \tag{15}$$

Algorithm 1 summarizes the constraint accumulation algorithm. If the degree of convexity of NF_j is positive or slightly negative (greater than -0.015) at the starting point of the i^{th} iteration, and if the constraint is active in the $(i-1)^{th}$ iteration, that is, X_{i-1}^* is on the surface of NF_j, then the accumulated constraints replace NF_j, Equation (16); otherwise, the single linear constraint in the i^{th} iteration is NF_j.

Algorithm 1. Constraint Accumulation Algorithm

In the i^{th} iteration,
for every j in J
if $\frac{1}{n}\sum_{p=1}^{n}\frac{\partial^2 NF_j}{\partial {x_{ip}}^2} \le -0.015$
and $NF_j\left(X_{i-1}^*\right) = 0$

$$LF_{i,j,} = LF_{i\text{-}1,j} \cup LF_{i,j} \tag{16}$$

Then, the revised simplex dual algorithm is invoked to solve the linear problem P_i^L, so a solution X_i^* is obtained. A point X_{i+1}^0, between the starting point X_i^0 and the solution X_i^*, becomes the starting point of the next iteration. The RMC is used to determine X_{i+1}^0, Equation (17).

$$X_{i+1}^0 = X_i^0 + RMC \cdot \left(X_i^* - X_i^0\right) \tag{17}$$

2.2. The Reduced Move Coefficient

The RMC is in the range [0, 1] and it defaults to 0.5 based on experimental observations [6], but 0.5 may not be the best for every problem. An optional golden section search algorithm is added to progressively narrow the range of the RMC by cutting off the sub-range with large deviations or larger violations of the constraints until the range is too small to reduce, Figure 6. The golden section search algorithm is given as follows – visualized in Figure 6 and summarized a Algorithm 2.

Algorithm 2. Golden section search for updating the RMC

```
#Define Performance function using RMC to linearize the model and obtain the merit function value
FUNCTION Performance (Model, RMC)
Linearize Model using ALP with RMC into Linear_Model
Solve Linear_Model using Dual Simplex
RETURN Z

#Define Golden Section Search function
FUNCTION GoldenSectionSearchForRMC (Rmin, Rmax, Th):
RMCa = Rmin + (0.382)*(Rmax − Rmin)
RMCb = Rmin + (1 − 0.382)*(Rmax − Rmin)
WHILE (RMCb − RMCa > Th):
        #Compare the performance of using RMCa versus RMCb
IF Performance (Original_Model, RMCa) < Performance (Original_Model, RMCb):
RMC = RMCa
Rmax = RMCb
ELSE:
RMC = RMCb
Rmin = RMCa
RETURN RMC

#Initialize parameters
Rmin = 0
Rmax = 1
Th = 0.0001

#Call the Golden Section Search Function
GoldenSectionSearchForRMC (Rmin, Rmax, Th)
```

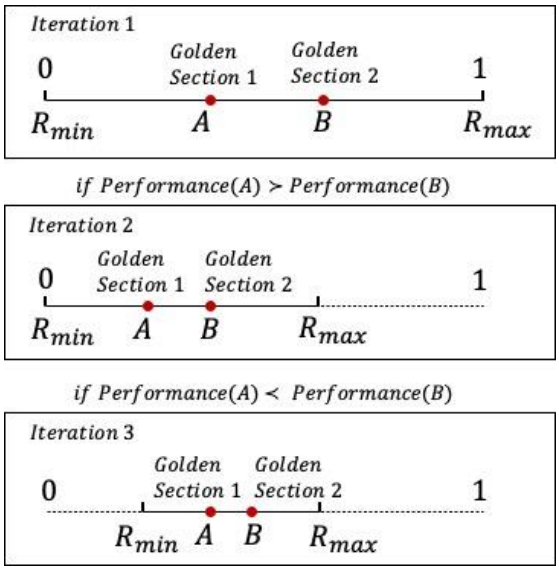

Figure 6. The golden section search for the RMCs in the ALP.

With the golden section search approach, the desired sub-range of the RMC may be missed if the performance oscillates in the range of the RMC. The criterion to evaluate the approximation performance is oversimplified—the deviations and the constraint violations. Given these limitations, in this paper, we propose to use parameter learning to determine the relation between the RMC value and solution quality with more evaluation criteria. Machine learning techniques are used to improve algorithms [32,33], so we leverage this idea to improve the ALP.

We hypothesize that by incorporating parameter learning in the ALP, we can update the RMC based on richer performance criteria. There are three steps to verify this hypothesis. In Section 3, we propose ALPPL to realize the three steps.

- Step 1. Identifying the criteria—to evaluate approximation performance.
- Step 2. Developing evaluation indices (EIs)—to quantify the approximation performance with RMC values.
- Step 3. Learning the desired range of each EI (DEI)—to tune the RMC.

In the next section, we discuss how parameter learning is used to dynamically change the value of the RMC.

3. Parameter Learning to Dynamically Change the RMC

In this section, the adaptive linear programming algorithm with parameter learning (ALPPL) is proposed, Figure 7. We add parameter updating, results extraction, performance evaluation, and feedback on parameter updating to the ALP.

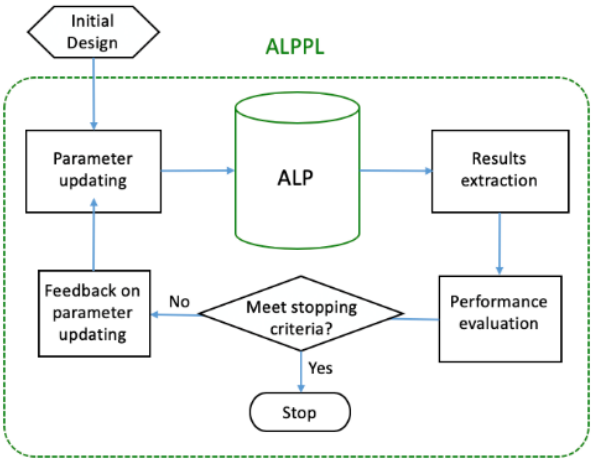

Figure 7. The concept of adaptive linear programming algorithm with parameter learning (ALPPL).

3.1. Step 1—Identifying the Criteria

In ALPPL, we add more criteria to the two criteria considered in the golden section search—deviation and constraint violation. We use a list, EIs[best], to record all the evaluation indices.

- Criterion 1—deviation.

The deviation in every iteration is recorded and ranked among all iterations. The best deviation in all iterations is stored in EIs[best]. Through implementing multiple design scenarios (lexicographic and/or weight scenarios), we obtain the deviations for all scenarios and iterations, and the desired range DEI is updated accordingly.

- Criterion 2—robustness of solutions.

We extend the notion of "feasibility" to "robustness", which means the solutions should be feasible under multiple design scenarios and relatively insensitive to model errors and uncertainties. We assume errors and uncertainties result in changes in the boundary of the feasible region. The nonlinear surface is a part of the boundary. Therefore,

the robustness of the solution can be measured by how far it is away from the nonlinear surface. In Figure 8, we show two situations. The arrows indicate the direction of the feasible space bounded by the inequality constraints while minimizing the deviation. In Figure 8a, the solution x_i^* is not on the boundary, so it is relatively insensitive to errors and variations of the model, and the nonlinear constraint NF_j is not an active constraint. In contrast, in Figure 8b, the solution is on the boundary and NF_j is an active constraint. The RMC affects the approximation in the next iteration, so we adjust the RMC to obtain more robust solutions.

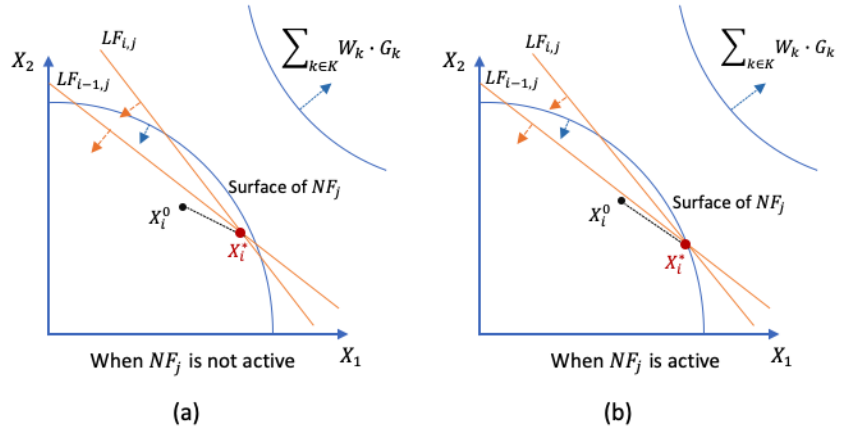

Figure 8. A relatively sensitive solution and a robust solution. Subfigure (**a**) shows the situation when the solution x_i^* is not on the boundary - the surface of NF_j, whereas Subfigure (**b**) shows the situation when the solution x_i^* is on the boundary.

- Criterion 3—approximation accuracy.

More accumulated constraints may not necessarily lead to a more accurate approximation. We want sufficient and useful accumulated constraints—Figure 9b—rather than many unnecessary accumulated constraints—Figure 9a.

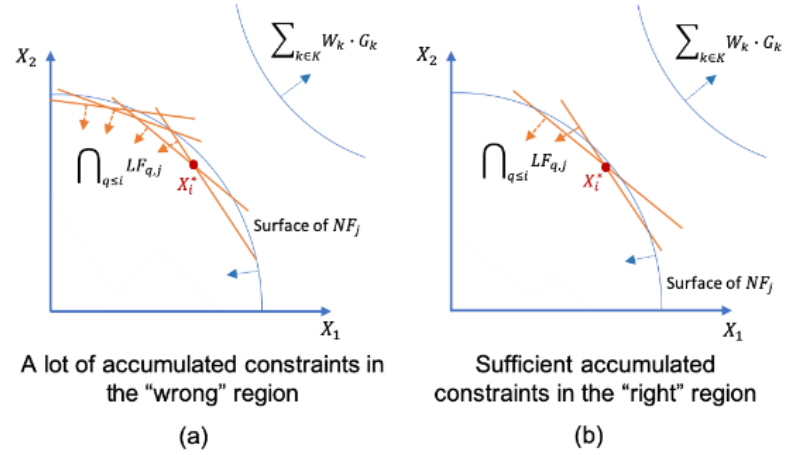

Figure 9. Unnecessary accumulated constraints versus necessary accumulated constraints. Subfigure (**a**) illustrates the situation when the accumulated constraints are in the "wrong" region, which results from the linearization points in multiple iterations being in a region that gives poor achieved values of the goal. Subfigure (**b**) illustrates the opposite situation – when the accumulated constraints are in the "right" region.

In Table 5, we summarize the criteria for approximation performance evaluation. Based on these criteria, we develop evaluation indices (EIs) in Section 3.2

Table 5. Criteria for the evaluation of approximation performance.

Criteria	Meaning and Representation
Weight-sum deviations	The unfulfilled percentage of the goals compared with their targets
Robustness	Whether a solution is away from the boundary of the system as defined by the nonlinear model
Approximation accuracy	Whether the nonlinear constraints are approximated well by a set of linear constraints in the sub-region that contains the solutions that fulfill the goals to an extent

3.2. Step 2—Developing the Evaluation Indices (EIs)

To manage different design preferences for multiple goals, we obtain solutions using multiple design scenarios. As the design scenarios are discrete and cannot enumerate all situations, we use the limited, discrete solutions to predict a satisficing solution space.

By implementing multiple design scenarios, we acquire the results of the weight-sum deviations, active constraints, accumulated constraints, etc., using which, we develop statistical-based evaluation indices (EIs) as shown in Table 6. Statistical-based quality control has been used to monitor model profiles [34], so we leverage the statistics to obtain EIs.

Our development of the EIs is based on the index for the robust concept exploration method, error margin index (EMI) [31], and design capability index (DCI) [35]. This is based on the central limit theorem that the results as samplings of each criterion follow Gaussian distributions, and the sampling statistics represent their characteristics. Therefore, we use the mean (μ) and the standard deviation (σ) as EIs and tune the RMC by minimizing the μ and σ of each EI.

Index for evaluating the weight-sum deviation—μ_Z and σ_Z. To maintain the goal fulfillment (one minus weight-sum deviation) relatively insensitive to design scenario changes, we control the center and spread of the sample solutions by minimizing μ_Z and σ_Z.

Index for evaluating the robustness—μ_{Nab}, σ_{Nab}, μ_{Naoc}, and σ_{Naoc}. Using the simplex algorithm, we obtain the vertex solution to the approximated linear problem. At a vertex solution, there is at least one active constraint (AOC) or active bound (AB) of the approximated linear problem, but we prefer fewer AOCs and ABs so the chance of losing a solution due to potential errors or variations is relatively small, so we minimize the μ_{Nab}, σ_{Nab}, μ_{Naoc}, and σ_{Naoc}.

Index for evaluating the computational complexity—μ_{Nacc}, σ_{Nacc}, μ_{Nit}, and σ_{Nit}. As the approximation accuracy is not necessarily improved by increasing the number of accumulated constraints (Nacc) or the number of approximation iterations (Nit), we need to find the ranges of Nacc and Nit associated with good weight-sum deviations and robustness. We desire Nacc and Nit to be acceptable under all design scenarios and be insensitive to scenarios changing, so we measure μ_{Nacc}, σ_{Nacc}, μ_{Nit}, and σ_{Nit} and identify their appropriate ranges.

In summary, we tune the RMC by satisfying the EIs—μ_{Nab}, σ_{Nab}, m_{Naoc}, σ_{Naoc}, μ_{Nacc}, σ_{Nacc}, μ_{Nit}, and σ_{Nit}— to obtain desired ranges (DEI). Then, the RMC tuning becomes: "given" the parameters and variables, "find" the value of decision variables that can "satisfy" constraints and bounds and "minimize" an objective. Here, we rank the EIs in the i^{th} iteration among all iterations and choose the first κ items with minimum values. These are the most important κ where EIs attain their best and most stable performance. Algorithm 3 is a summary of the RMC parameter learning algorithm.

Table 6. Develop evaluation indices (EIs)—mean (μ) and standard deviation (SD) (σ).

Criteria	Information	EIs	Meaning
Weight-sum deviation	$Z = \sum_{k \in K} W_k \cdot (d_k^- + d_k^+)$	μ_z	The average weight-sum deviations
		σ_z	The stability of the weight-sum deviations when changing design scenarios
Robustness	The number of active bounds:	μ_{Nab}	The average sensitivity of the solutions under all design scenarios to variable bounds
		σ_{Nab}	The stability of the sensitivity of the solutions to variable bounds when changing design scenarios
	The number of active original constraints:	μ_{Naoc}	The average sensitivity of the solutions under all design scenarios to original constraints
		σ_{Naoc}	The stability of the sensitivity of the solutions to original constraints when changing design scenarios
Approximation accuracy	The number of accumulated constraints:	μ_{Nacc}	The average complexity of the approximated problem under all design scenarios
		σ_{Nacc}	The stability of the complexity of the approximated problem when changing design scenarios
	The number of iterations:	μ_{Nit}	The average convergence speed under all design scenarios
		σ_{Ni}	The stability of the convergence speed when changing design scenarios

Algorithm 3. #Define Parameter-learning function

```
FUNCTION ParameterLearning(Model, Design scenarios, RMC, EIs, κ, I):
WHILE i < I:
    IF RMC[i-1] = RMC_best:
        RMC[i] = average(RMC[i-2], RMC_best)
    ELSE:
        RMC[i] = average(RMC[i-1], RMC_best)
    FOR N Design Scenarios:
        Linearize Model using ALP with RMC[i] into Linear_Model
        Solve Linear_Model using Dual Simplex
    RETURN EIs[i]

    FOR j in Range(0, i):
        Rank EIs[i][j] from minimum to maximum
        DEI = EIs[i][round(κ*i)]

    IF (number of EIs[i] ∈ DEI for all Design scenarios) >
(number of EIs[Cycle of RMC_best] ∈ DEI for all Design scenarios):
        RMC_best = RMC[i]

RETURN RMC_best

#Initialize parameters
Model = Original_Model
Design scenarios = N weight scenarios
RMC [0] = 0.5
EIs = [μZ, σZ, μNab, σNab, μNaoc, σNaoc, μNacc, σNacc, μNit, σNit]
Z, Nab, Naoc, Nacc, Nit = 0
DEI = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
κ = 0.6
I = 50

#Call Parameter-Learning function
FUNCTION ParameterLearning(Model, Design scenarios, RMC, EIs, κ, I)
```

3.3. Step 3—Learning the DEI to Tune the RMC

We identify the desired range of the EIs (DEI), learn the connections between the RMC and the EIs, and bring the EIs into the DEI by setting the RMC. To make the process

efficient, we combine off-line learning and the on-line learning. First, we use off-line learning using a sample of RMC values to initialize the DEI and parameters and then adopt on-line learning to update the DEI and tune the RMC. In Figure 10, we illustrate the two processes (Rectangles B and C) and show their relationship with the synthesis cycle A.

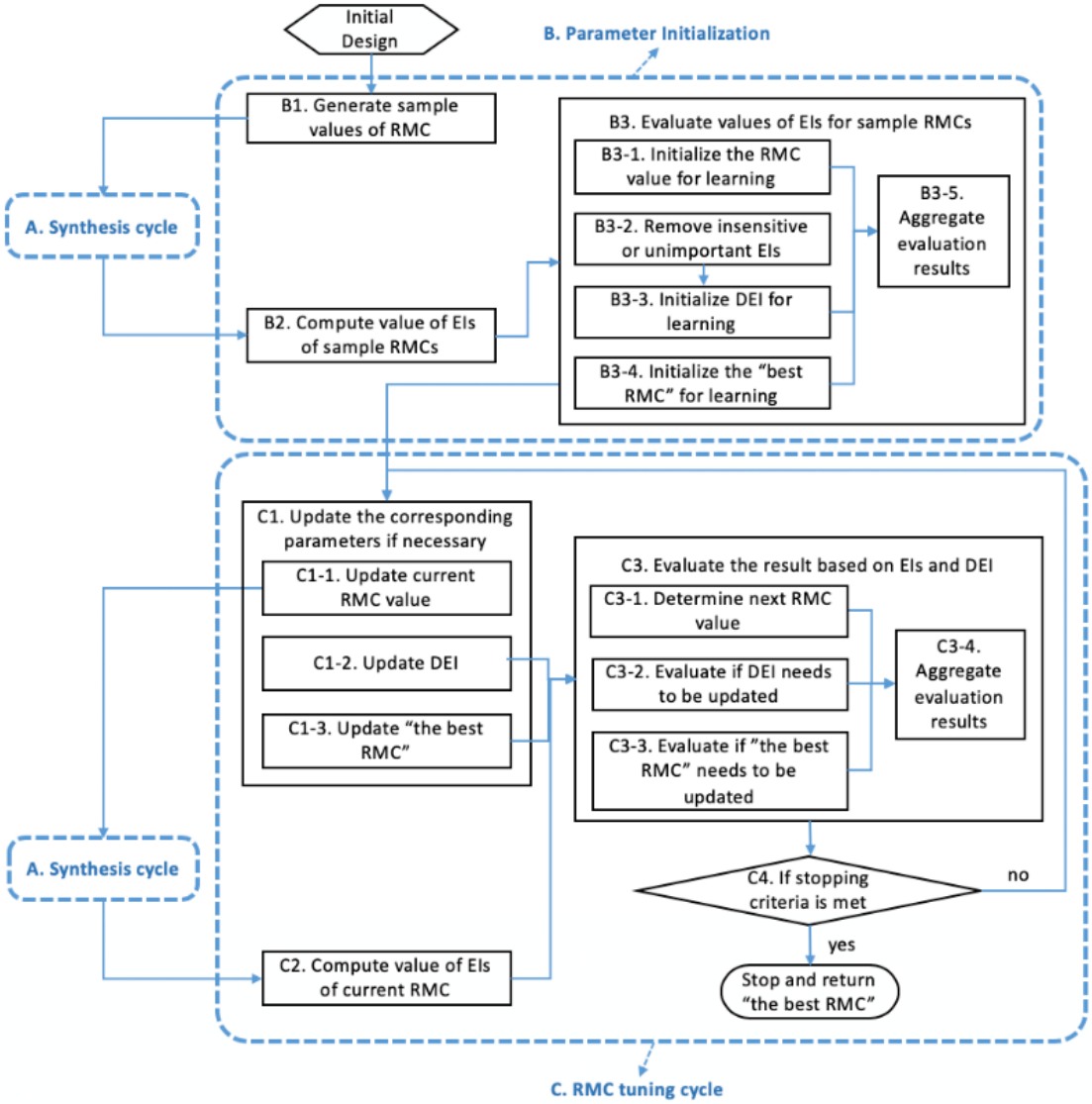

Figure 10. ALPPL including parameter initialization and RMC tuning.

During parameter initialization (B), we generate sample values of RMC (B1). By running the synthesis cycle (A) with each RMC value, we obtain the corresponding EIs (B2) and evaluate them (B3). We choose the RMC value associated with the best EIs as the starting RMC value for tuning (B3-1) and remove the EIs that are insensitive to RMC changes (B3-2). We initialize the DEI based on the sample results (B3-3) to allow a certain proportion (e.g., 75%) of RMC values to fall into the DEI. We choose the best RMC among the sample (B3-4) to start the on-line learning. These results (B3-1 to B3-4) are aggregated as the "actions to be taken" (B3-5) and the input of the RMC tuning cycle (C).

For the RMC tuning cycle (C), with an RMC value (C1-1), we run the synthesis cycle (A) and obtain the results (C2). By evaluating the results using the DEI and comparing with previous iterations (C3), we determine the next RMC value (C3-1), update the DEI if necessary—either restrict or relax the DEI based on the tradeoffs among EIs (C3-2)—and update the best RMC if necessary (C3-3). These evaluation results are aggregated (C3-4) for judging whether to stop iterating (C4)—the program either goes to the next iteration of

RMC tuning with the aggregated results (C3-4) as input or stops with the best RMC as the returned value. The stopping criteria include the number of total iterations and the number of iterations without updating the best RMC. This part is summarized as Algorithm 4.

Algorithm 4. The RMC parameter-learning algorithm

1 Given: the best RMC sample value, updating rules
2 Initialize: t = 0, best = the best RMC sample value, RMC_0 = the best RMC sample value, the maximum iteration number T, stopping criterion 2 = {best has not been updated in n iterations}
3 **While** t = do T // Define stopping criterion 1
4 RMC_t = Next_RMC
5 Run synthesis cycle

4. A Test Problem

In this section, we apply ALPPL to an industry-inspired problem, the integrated design of a hot-rolling process chain for the production of a steel rod [36]. It is a nonlinear problem, and the RMC value has a significant impact on the result.

4.1. The Hot Rod-Rolling Process Chain

Hot rod rolling is a multi-stage manufacturing process in which a reheated billet, slab, or bloom produced after the casting process is further thermo-mechanically processed by passing it through a series of rollers [36]. During the thermo-mechanical processing, there is an evolution of microstructure of the material, in this case, steel. Columnar grains in the cast material are broken down to equiaxed grains. Along with the evolution of grain size, there is a phase transformation of the steel. The phase transformation is predominant during the cooling stage that follows the hot rod-rolling process chain. The transformation of the austenite phase of steel to other phases like ferrite, pearlite, or martensite takes place during this stage. The final microstructure of the material after the rolling and cooling process defines the mechanical properties of the product.

Many plant trials are required to produce a new steel grade with improved properties and performance. These trials are usually expensive and time-consuming. Hence, there is a need to address the problem from a simulation-based design perspective to explore solutions reaching multiple conflicting property/performance goals. The requirement is to produce steel rods with improved mechanical properties such as yield strength (YS), tensile strength (TS), and hardness (HV). These mechanical properties are defined by the microstructure after cooling, which includes the phase fractions (ferrite and pearlite phases are only considered in this problem), pearlite interlamellar spacing, ferrite grain size, and chemical compositions of steel. Nellippallil et al. [36] identify the microstructural requirements after the cooling stage to meet the mechanical properties of the rod. The microstructural requirements are to achieve a high ferrite fraction value, low pearlite interlamellar spacing, and low ferrite grain size values within the defined ranges. The requirement is to carry out the integrated design of the material and the process by managing the cooling rate (cooling process variable), final austenite grain size after rolling (rolling microstructure variable), and the chemical compositions of the material. Hence, we explore the solution space of the defined variables using ALPPL to meet the target values identified for the microstructure after the cooling stage such that the mechanical property requirements of the steel rod are met. Our focus in this paper is to use this example in improving the solution algorithm rather than the design of the material and the manufacturing. The initial design formulation of the problem is shown as follows.

Given
Target values for microstructure after cooling
Ferrite grain size target, $D_{\alpha,Target}$ = 8 μm
Ferrite fraction target, $X_{f,Target}$ = 0
Pearlite interlamellar spacing target, $S_{0,Target}$ = 0.15
Find
System variables

X_1 cooling rate (CR)
X_2 austenite grain size (D)
X_3 the carbon concentration ([C])
X_4 the manganese concentration after rolling ([Mn])
Deviation variables
d_i^-, d_i^+, i =1,2,3
Satisfy
System constraints

Minimum ferrite grain size constraint $D_\alpha \geq 8$ μm (18)

Maximum ferrite grain size constraint $D_\alpha \leq 20$ μm (19)

Minimum pearlite interlamellar spacing constraint $S_o \geq 0.15$ μm (20)

Maximum pearlite interlamellar spacing constraint $S_o \leq 0.25$ μm (21)

Minimum ferrite phase fraction constraint (manage banding) $X_f \geq 0.5$ (22)

Maximum ferrite phase fraction constraint (manage banding) $X_f \leq 0.9$ (23)

Maximum carbon equivalent constraint $C_{eq} = (C + Mn)/6$; $C_{eq} \leq 0.35$ (24)

Mechanical Property Constraints

Minimum yield strength constraint YS ≥ 250 MPa (25)

Maximum yield strength constraint YS ≤ 330 MPa (26)

Minimum tensile strength constraint TS ≤ 480 MPa (27)

Maximum tensile strength constraint TS ≥ 625 MPa (28)

Minimum hardness constraint HV ≥ 130 (29)

Maximum hardness constraint HV ≤ 150 (30)

System goals

The target values for system goals are identified in [36] and are listed under the keyword Given above.

Goal 1 : Achieve ferrite grain size target $D_{\alpha Target}/D_\alpha(X_i) + d_1^+ - d_1^- = 1$ (31)

Goal 2 : Achieve ferrite fraction target $X_f(X_i)/X_{fTarget} + d_2^- - d_2^+ = 1$ (32)

Goal 3 : Achieve pearlite interlamellar spacing target $S_{oTarget}/S_o(X_i) + d_3^+ - d_3^- = 1$ (33)

Variable bounds
$11 \leq X_1 \leq 100$ (K/min)
$30 \leq X_2 \leq 100$ (μm)
$0.18 \leq X_3 \leq 0.3$ (%)
$0.7 \leq X_4 \leq 1.5$ (%)
Bounds on deviation variables

$$d_i^-, d_i^+ \geq 0 \text{ and } d_i^- * d_i^+ = 0\ ,\ i = 1,2,3 \quad (34)$$

Minimize
Minimize the deviation function in the initial design

$$Z = \sum_{i=1}^{3} W_i\left(d_i^- + d_i^+\right);\ \sum_{i=1}^{3} W_i = 1 \quad (35)$$

There are three goals in the problem—(i) minimize ferrite grain size (D_α), (ii) maximize ferrite Fraction (X_f), and (iii) minimize interlamellar spacing (S_o). The targets and the acceptable values of the three goals are given. All three goals are nonlinear. There are four design/system variables—cooling rate (CR), final austenite grain size after rolling (D), the carbon concentration ([C]), and the manganese concentration after rolling ([Mn]). We aim to obtain:

- The range of the system variables to reach the target of each goal of the best RMC for different design preferences and
- The weight set, $W^s = \cap_{k \in K} W_k^s$, that is a compromise of the three goals for different design preferences.

In [36], the authors formulate and execute the initial compromise decision support problem for the hot-rolling process chain problem and carry out a weight sensitivity analysis to identify variable values and the weight set. Ternary plots are generated to visualize and explore the weight set. In each ternary plot, the three axes represent the weights assigned to the three goals, respectively, and the contours indicate the fulfillment of each goal, $\frac{G_k}{T_k}$, $k = 1, 2, 3$. Since the goals conflict, compromise solutions are desired. Weight sensitivity analysis is a way to mediate compromise around the conflicts among the goals. W_1^s, W_2^s W_3^s, and W^s are the satisficing weight regions identified in the ternary plots, see areas marked using arrows in Figure 11a–d, respectively. To compare $\frac{G_k}{T_k}$, $k = 1, 2, 3$ at the same scale, we normalize the fulfillments of each goal under all weight scenarios in the range [0, 1]. An acceptable fulfillment $\frac{G_k}{T_k}$ of each goal is the dashed line in each ternary plot. The area between the corner (best $\frac{G_k}{T_k}$) and the dashed line (acceptable $\frac{G_k}{T_k}$) is the satisficing weight area of Goal k, W_k^s and the superimposed area is the satisficing weight set of all three goals, W^s.

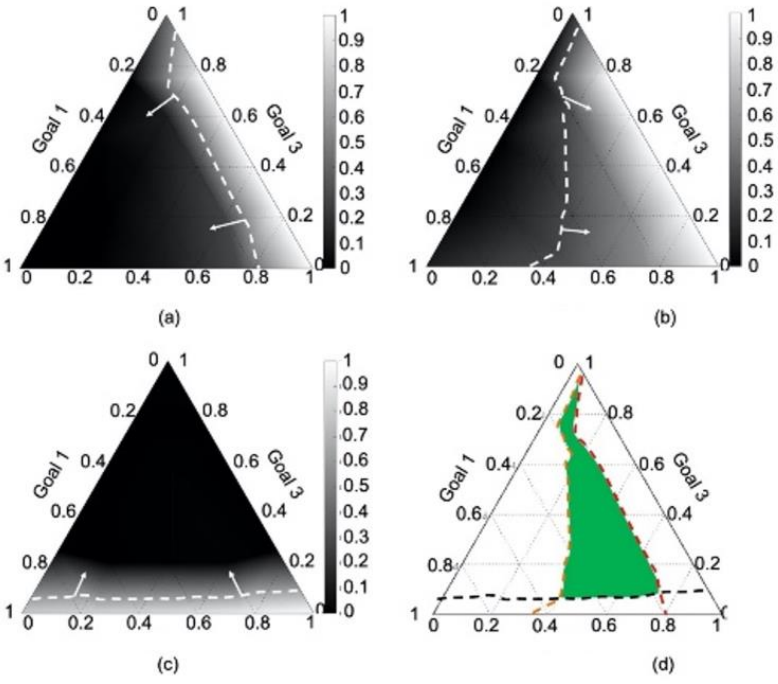

Figure 11. The satisficing weight set when setting RMC = 0.1. Subfigure (**a**–**c**) show the acceptable regions of weights for the three goals, respectively. Subfigure (**d**) shows the acceptable region weights for all three goals. The dashed lines demarcate acceptable regions for the goals and the arrows indicate the direction of acceptability.

The steps in the RMC tuning algorithm as applied to the hot rod-rolling problem are given in Algorithm A1, Appendix A. In Figures 11d, 12d and 13d, the satisficing weight set W^s when the RMC is 0.1, 0.5, and 0.8 is shown, respectively. When RMC is 0.1, as Figure 11d

shows, W^s is large, whereas when RMC is 0.8, as Figure 13d shows, W^s is small. However, using the ALP, there is no mechanism to evaluate which RMC value results in a relatively accurate approximation that gives us a robust W^s. So, we fill this gap using ALPPL.

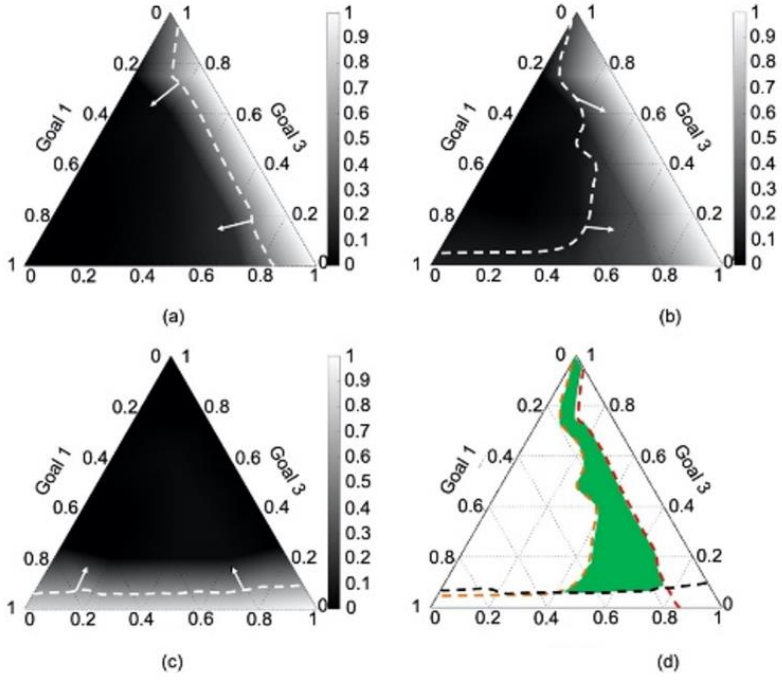

Figure 12. The satisficing weight set when setting RMC = 0.5. Subfigure (**a**–**c**) show the acceptable regions of weights for the three goals, respectively. Subfigure (**d**) shows the acceptable region weights for all three goals. The dashed lines demarcate acceptable regions for the goals and the arrows indicate the direction of acceptability.

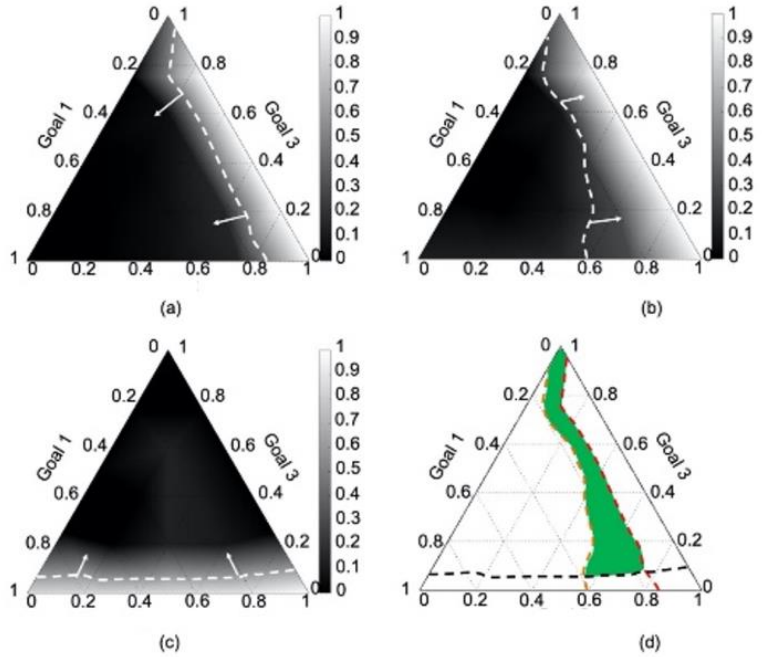

Figure 13. The satisficing weight set when setting RMC = 0.8. Subfigure (**a**–**c**) show the acceptable regions of weights for the three goals, respectively. Subfigure (**d**) shows the acceptable region weights for all three goals. The dashed lines demarcate acceptable regions for the goals and the arrows indicate the direction of acceptability.

4.2. Parameter Initialization

There are nineteen weight scenarios used in Table 7, representing a variety of design preferences.

Table 7. Weight vectors used in [36] as different design scenarios.

	W1	*W2*	*W3*		*W1*	*W2*	*W3*
1	1	0	0	11	0	0.75	0.25
2	0	1	0	12	0	0.25	0.75
3	0	0	1	13	0.33	0.33	0.33
4	0.5	0.5	0	14	0.2	0.2	0.6
5	0.5	0	0.5	15	0.4	0.2	0.4
6	0	0.5	0.5	16	0.2	0.4	0.4
7	0.25	0.75	0	17	0.6	0.2	0.2
8	0.25	0	0.75	18	0.4	0.4	0.2
9	0.75	0	0.25	19	0.2	0.6	0.2
10	0.75	0.25	0				

Running Processes B1 and B2 using the sample RMC (0.1, 0.5, and 0.8), we obtain the EIs in Table 8.

Table 8. Results of EIs using sample RMC values with nineteen design scenarios.

RMC	Statistics	Z	Nit	Nacc	Nab	Naoc
0.1	μ	0.1480	46.58	18.74	1.79	0.84
	σ	0.0679	5.95	0.87	0.63	0.50
0.5	μ	0.1467	20.42	19.47	2.05	0.79
	σ	0.0675	9.47	0.84	0.62	0.42
0.8	μ	0.1480	8.32	14.16	2.21	0.79
	σ	0.0675	5.56	6.94	0.63	0.71

Running Process B3, we obtain the initial RMC (RMC_0) and the best RMC as 0.5 (its μ_Z and σ_Z are bold italic), and we initialize the DEI as shown in Table 9. The results of the parameter initialization—RMC_0 (0.5), EIs (Table 5), DEIs (Table 9), and "best" RMC value 0.5—are the input of RMC tuning (C).

Table 9. The initial DEIs.

DEI of μ_z	DEI of σ_z	DEI of μ_{Naoc}	DEI of σ_{Naoc}	DEI of μ_{Nab}	DEI of σ_{Nab}
[0, 0.1477]	[0, 0.0677]	[0, 0.82]	[0, 0.55]	[0, 1.95]	[0, 0.63]

4.3. RMC Tuning

We make rules for each procedure of RMC tuning based on heuristics. The heuristics are generalized from parameter learning and can be adjusted through the search process.

C3: Evaluate the result of current RMC based on EIs and DEI.

C3-1: Determine the next RMC value.

Rule 1: Compare the performance of multiple EIs and define the comparison rules. Lines 22–30 in Algorithm A1 in Appendix A are an expansion of this rule. RMC A is better than RMC B because no less than κ of the EI(A) are better than EI(B), whereas other EI(A) do not exceed γ of the upper and lower bound of DEI. In this problem, we set $\kappa = 1/2$ and $\gamma = 3$ 0%.

Rule 2: Determine when and how the RMC should be updated. Lines 6–13 in Appendix A explain this rule. We use a hill-climbing approach to update the RMC. If the updating in the previous RMC-tuning cycle improves the performance, then the previous updating is in the "hill-climbing direction," and we continue updating the RMC in this

direction with a step size α; otherwise, we need the best RMC to "pull us back" to the right direction with a portion β; hence, we update the RMC as a linear combination of the best RMC (elite) and the RMC two cycles ago (parent). In this problem, we set α as a random value in [0, 1] and β as a random value in [0.5, 1]. In this way, we incorporate greediness, elitism, and randomness in evolution.

C3-2: Evaluate whether DEI needs to be updated.

Rule 3: Determine when and how the DEI should be updated. See Algorithm A1 in Appendix A, Lines 18–38. In an RMC-tuning cycle, if more than κ EIs are better than the previous cycle, and more than ι EIs are in the DEI, whereas no more than $\gamma\gamma$ EIs have minor violations, then the DEI is updated. Here, we set $\kappa = \iota = 2/3$. For a problem with more EIs, κ and ι can be tuned using the performance improvement rate. This rule prevents an over restrictive DEI from blocking us from a better range while ensuring gradual and relatively conservative updating of the DEI.

C3-3: Evaluate whether "the best RMC" needs to be updated.

Rule 4: Determine when and how the best RMC is updated. See Algorithm A1 in Appendix A, Lines 34–36. We use a variable ("best") to store the best RMC. If more than κ EIs of the current RMC are better than the EIs of the best, the current RMC becomes the new best.

C3-4: We aggregate the results of the RMC tuning RMC $_{t+1}$, DEI, and best as inputs for the next tuning iteration.

C4: Determine whether to stop iterating.

Rule 5: Make the stopping criteria. To stop RMC tuning at the appropriate time, we use "the maximum number of RMC-tuning iterations" and "the maximum number of RMC-tuning iterations without updating the best RMC" as the stopping criteria.

4.4. Results

Using our test problem, the RMC tuning stops after fourteen iterations. We identify 0.55 as the best RMC for the test problem. Compared with the initial RMC of 0.5, the final best RMC of 0.55 improves σ_Z, σ_{Naoc}, and σ_{Nab}. The RMCs and EIs in the fourteen iterations are in Table 10. The DEI is updated four times, and the best RMC is updated three times. The final best RMC is in the ninth iteration.

Table 10. The *EIs*, *DEI*, *RMC*, *best RMC* of the fourteen iterations of *RMC* tuning.

Iteration	RMC	Weight-Sum Deviation		Number of Active Original Constraints		Number of Active Bounds		Better Than Cycle (t-1)	Better Than Best RMC	Update DEI
		μ_z	σ_z	μ_{Naoc}	σ_{Naoc}	μ_{Nab}	σ_{Nab}			
1	0.5	0.147	0.068	0.79	0.42	2.05	0.62	-	-	[1, 1.95] -> [1, 2.05]
2	1.0	0.152	0.071	0.95	0.78	2.26	0.45	N	N	-
3	0.8	0.148	0.068	0.79	0.71	2.21	0.63	N	N	-
4	0.6	0.147	0.067	0.95	0.52	2.00	0.58	Y	Y	->
5	0.4	0.147	0.068	0.68	0.48	2.00	0.58	Y	Y	-
6	0.2	0.147	0.068	0.84	0.60	1.84	0.60	N	N	->
7	0.3	0.154	0.069	0.95	0.52	1.79	0.71	N	N	-
8	0.45	0.147	0.068	0.89	0.66	2.00	0.58	Y	N	-
9	0.55	0.147	0.067	0.84	0.37	2.05	0.52	Y	Y	-
10	0.65	0.147	0.067	1.00	0.58	2.05	0.62	N	N	->
11	0.48	0.147	0.068	0.89	0.66	2.05	0.62	N	N	-
12	0.53	0.147	0.067	0.84	0.69	2.00	0.58	Y	N	-
13	0.57	0.147	0.067	0.84	0.37	2.11	0.57	N	N	-
14	0.43	0.145	0.069	0.84	0.60	2.00	0.58	N	N	-

4.5. Verification and Validation

To verify the efficacy of ALPPL, we evaluate its adequacy and the necessity in returning robust solutions.

Adequacy—We tested 20 RMC values randomly spread in [0, 1] and identified the relatively insensitive range(s); see Figure 14. If the best RMC returned by the ALPPL falls into the relatively insensitive range(s), then we show that the ALPPL can identify the best RMC. It turns out that the best RMC (0.55) ensures the solutions to fall in a relatively insensitive range. For each pair of EIs (the mean and standard deviation), we identify the range(s) of RMC values, within which, both the mean and the standard deviation have acceptable values and are flat (stable). For example, for the deviation Z, we desire both its mean μ_z and standard deviation σ_z to be low values and flat. The two ranges bounded by the dotted rectangles are such ranges. For all EIs, the overlapped desired range is [0.5, 0.55], so this is the relatively insensitive range. The value of 0.55 is in the insensitive RMC range for all EIs, so it is verified that when RMC is 0.55, it gives a relatively robust performance.

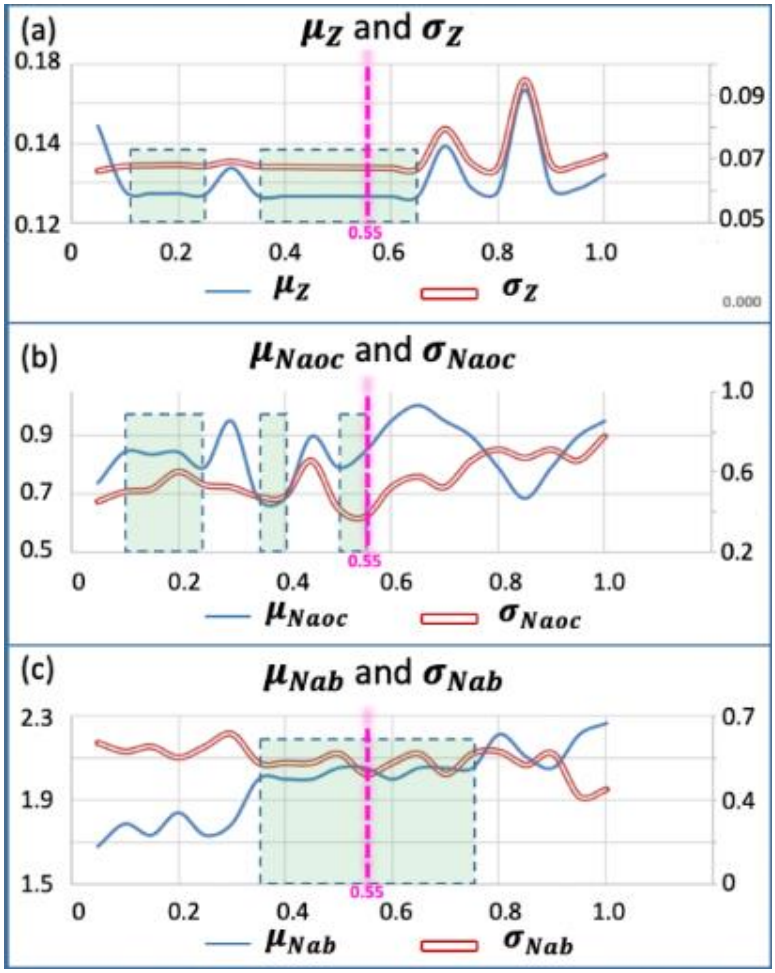

Figure 14. Identifying the insensitive range of RMC value using twenty RMC values. Subfigure (**a**–**c**) show the insensitive ranges of the mean and standard deviation of Z, Naoc, and Nab, respectively.

Necessity—The insensitive range is sufficiently explored during the RMC tuning. First, we identify the insensitive range of RMC values—testing 20 RMC values uniformly distributed in [0, 1] with their EIs, we identify two ranges where the solutions are relatively insensitive to scenario changing, [0.35, 0.4] and [0.5, 0.55]. In Figure 15a, we illustrate all fourteen RMC values in RMC tuning. The horizontal axis represents the iteration number and the vertical axis represents the RMC value. Four of the fourteen RMC values fall in the two insensitive ranges, so 28.5% of the tested RMC values fall in the insensi-

tive ranges, whereas the insensitive ranges only occupy 10% of the whole RMC range. Hence, we conclude that our RMC tuning enables a relatively sufficient exploration of the insensitive ranges.

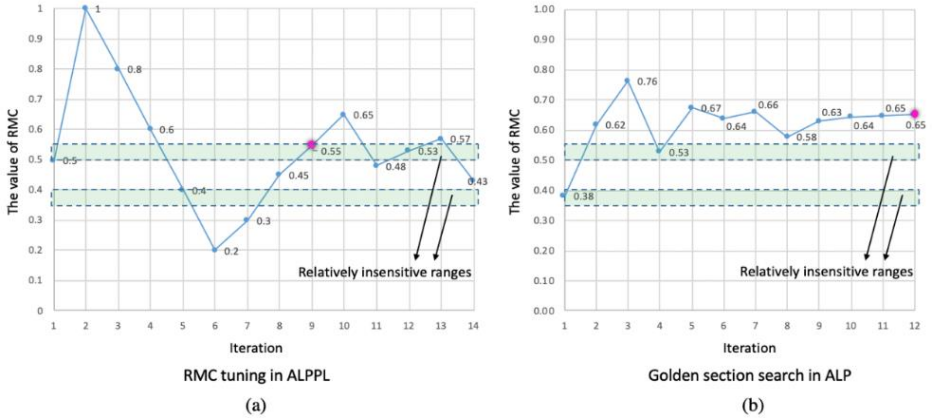

Figure 15. The comparison of ALPPL and ALP regarding RMC updating. Subfigure (**a**) illustrates the best RMC of the fourteen iterations when applying ALPPL. Subfigure (**b**) illustrates the best RMC of the twelve iterations when applying ALP.

Validation of the improvement of ALPPL versus the ALP. We compare the tested RMC values using parameter learning (ALPPL) in Figure 15a with the tested RMC values using the golden section search (ALP) in Figure 15b. The best RMC identified using the golden section search is 0.65, which is not in the insensitive range; in addition, the RMC values tested in the golden section search are concentrated in [0.57, 0.77], which misses the insensitive ranges [0.35, 0.4] and [0.5, 0.55]. For a solution algorithm that sequentially linearizes the problems in multiple synthesis cycles, the computational complexity is $O(n^2)$ [37]—this applies to both the ALP and ALPPL, so the parameter-learning method in this paper does not require much computational power. Other improvements of ALPPL over ALP are summarized in Table 11.

Table 11. ALPPL with RMS tuning versus ALP with golden section search.

		ALPPL	**ALP**
General comparison	Search method	Rule-based parameter learning	Golden section search
	Criteria used for evaluation of the RMC	Deviation (fulfillment of the goals), the robustness of the solution	Fulfillment of the goals
	If the approximation is sensitive to the scenario changing	Considering different scenarios, the most appropriate RMC is identified. The approximation is relatively insensitive to scenario changes	In each scenario, the best RMC is identified; it may vary as the scenario changes. The approximation is relatively sensitive to scenario changes
	Stopping criteria	The best RMC has not been updated for n iterations, or the total iteration number reaches a threshold	The distance between two golden section points is less than a threshold ε
	Complexity	$O(n^2)$	$O(n^2)$
Comparison of the cooling problem results	Number of search iterations	14	12
	If the identified best RMC is in the insensitive range	Yes	No
	Number of tested RMC values falling into insensitive range	4	2
	Is the insensitive range explored sufficiently	Relatively sufficiently	Insufficiently

5. Closing Remarks

Our aim in this paper is to improve the determination of one critical parameter in a satisficing method regarding the robustness of the solution and the accuracy of the linearization. We show the benefits of using a satisficing strategy, the cDSP, and ALP to manage engineering design problems, especially when there are multiple goals, nonlinear and nonconvex equations, and the goals have different levels of achievability. However, there is a drawback of the current ALP—one of its parameters, the RMC, has an impact on the linearization and the solution space. For years, the RMC was either user-defined or determined using golden section search. However, there was no mechanism for obtaining insight to improve the approximation by controlling the RMC. The golden section approach may result in missing the sub-range of the RMC with good approximation performance. The best RMC value is sensitive to design scenario changes; only the weighted-sum deviations and feasibility of the solutions are considered when evaluating approximation performance. Hence, to improve the ALP, we incorporate parameter learning into the algorithm and upgrade it to the adaptive linear programming algorithm with parameter learning algorithm (ALPPL). In the ALPPL, we improve the approximation performance in three steps—identifying the criteria for approximation performance, developing evaluation indices (EIs), and tuning the RMC. We use an industry-inspired problem, namely, the hot-rolling process for a steel rod, to demonstrate the improvements of ALPPL (to access DSIDES and ALPPL, please contact the corresponding author. The tutorial for using DSIDES to formulate a cDSP is in this video: https://www.youtube.com/watch?v=tUpVC97Y1L8, accessed on 25 November 2020) over the ALP. The computational complexity of ALPPL is the same as the ALP, which is $O(n^2)$.

The parameter-learning method used in this paper can be expanded to other algorithms that apply heuristics to determine the value of parameters. Domain expertise can be helpful in identifying the criteria for performance evaluation and developing evaluation indices. With domain knowledge, the computational complexity of the parameter-learning algorithms can be well controlled within an inexpensive range. This can customize the parameter learning for each algorithm and each problem.

One limitation of this paper is that we did not discuss in great detail the computational complexity of the ALPPL or the parameter learning for algorithm improvement in general, because it can vary for different algorithms or problems. One thing that we can ensure is that, with designers' domain expertise, by identifying the most critical criteria and then quantifying them as the most representative evaluation indices, the computational power required for the proposed parameter-learning algorithm can be less than that for parameter tuning without domain expertise, since designers can remove the less relevant criteria and insensitive evaluation indices.

In future research, there are several promising directions to explore in the realm of engineering design using the satisficing strategy. Firstly, investigating the applicability and effectiveness of the proposed rule-based parameter-learning procedure across a broader range of engineering problems could provide valuable insights into its generalizability and robustness. Additionally, exploring alternative satisficing algorithms or enhancements to existing algorithms could further improve the efficiency and robustness of design processes. Furthermore, incorporating advanced machine learning techniques, such as reinforcement learning or neural networks, into the satisficing framework may offer new opportunities for adaptive and intelligent decision making in engineering design. Finally, conducting empirical studies and applications in diverse industries and contexts could validate the practical value and benefits of the proposed approach, thereby facilitating its adoption and integration into engineering practice. Overall, these future research directions hold the potential to advance the state of the art in satisficing engineering design and contribute to the development of more automated and robust design methods.

6. Glossary

Accumulated constraint(s): When a nonlinear constraint is concave (the degree of the convexity is lower than -0.015), the approximated linear constraint of the previous iteration is carried forward and combined with its approximated linear constraint in the current iteration. Thus, a concave nonlinear constraint is represented by multiple linear constraints, and those carried forward are called accumulated constraints.

Active constraint: If the slack or surplus of a constraint at a solution point is zero, then it is an active constraint.

Convexity degree: The average value of the diagonal terms of the Hessian matrix of a function. This definition is applied to any paper on the cDSP and ALP.

Deviation: The percentage difference between the value of a goal at a solution and its target.

DSIDES: DSIDES is short for decision support in the design of engineering systems, which is a computational platform for designers to formulate engineering design problems using the compromise decision support problem (cDSP) and linearize and solve the problems using the adaptive linear programming (ALP) algorithm.

Parameter learning: This includes a set of activities to maximize the approximation performance—identifying the evaluation indices (EIs), the desired range of each evaluation index (DEI), and the value of a parameter, namely, the reduced move coefficient (RMC), that makes each EI fall into its desired range, DEI.

Parameter tuning: This means finding the value of a parameter, namely, the reduced move coefficient (RMC), that makes each EI fall into its desired range, DEI. It is a part of parameter learning.

Author Contributions: L.G. was responsible for developing and implementing the advanced adaptive linear programming algorithm with parameter learning and writing this paper with input from her co-authors. F.M. is the originator of the adaptive linear programming (ALP) algorithm. F.M. is responsible for coding the algorithm; see [6]. W.F.S. incorporated the ALP algorithm into the decision support environment, DSIDES. He was instrumental in advising L.G. about the use of DSIDES and ensuring what is written is accurate; see [30]. A.B.N. provided the test example for L.G. to use in her dissertation. He was instrumental in advising L.G. about how the data were structured, the interpretation of the results, and ensuring that what is written is accurate; see [36]. J.K.A. and F.M. mentored L.G.to complete her PhD dissertation. Both supported L.G. financially during her PhD program. Both revised and edited the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: Lin Guo was supported by the LA Comp Chair and the John and Mary Moore Chair at the University of Oklahoma till August 2021, then is supported by the Pietz Professorship funds and the start-up fund from the Research Affairs Office at South Dakota School of Mines and Technology since September 2021. All the three funding sources are internal.

Data Availability Statement: The data used in this work are taken from the publication by Nellippallil [36].

Acknowledgments: We are grateful to the late O.F. Hughes for originating the adaptive linear programming (ALP) algorithm. We are grateful to H.B. Phouc for coding the ALP algorithm. We are grateful to Krishna Atluri and Zhenjun Ming for their help in testing the ALPPL algorithm. Lin Guo acknowledges the Pietz Professorship funds and the start-up fund from the Research Affairs Office at South Dakota School of Mines and Technology. Farrokh Mistree and Janet K. Allen gratefully acknowledge support from the LA Comp Chair and the John and Mary Moore Chair at the University of Oklahoma.

Conflicts of Interest: On behalf of all authors, the corresponding author states that there is no conflict of interest.

Nomenclature

ALP	Adaptive Linear Programming
ALPPL	Adaptive Linear Programming with Parameter Learning
cDSP	Compromise Decision Support Problem
DEI	Desired Range of Evaluation Index
DSP	Decision Support Problem
EI/EIs	Evaluation Index/Indices
RMC	Reduced Move Coefficient

Appendix A

The RMC Parameter-Learning Algorithm Customized for the Hot Rolling Process Chain Problem

Appendix A is the RMC parameter-learning algorithm (Algorithm 4) customized for the Cooling Procedure of the Hot Rolling Problem. Appendix A is referenced in Section 4. This algorithm is an extension of the Algorithm 4. More auxiliary parameters are defined to assist parameter learning—T, α, β, γγ, θθ, ι, κ, M. For the parameters that are relatively more important (the results are more sensitive to their values), e.g., θθ, we tune their values. For the parameters that are relatively less important, e.g., ι, κ, M, we set values to them with heuristics.

Steps in the RMC-tuning algorithm: Determine the evaluation indices (EIs) based on multiple criteria to classify good results from the bad ones; initialize the desired range of each EI (DEI) of the test problem; identify auxiliary parameters to assist RMC tuning; bring the EIs into DEI by tuning the auxiliary parameters; update DEI to ensure a proportion of good results out of all results; tradeoff between elitism and randomness to ensure a diversity while obtaining rapid convergence.

Algorithm A1. The RMC parameter-learning algorithm (Algorithm 4) customized for the Cooling Procedure of the Hot Rolling Problem

t <- 0 // Initiate the number of synthesis cycles

b <- 0 // Initiate the time of updating the best RMC

RMC_t <- a random value // Initiate RMC with a random value (here we set RMC_0<- 0.5)

$best_b$ <- a random value // Initiate the "best RMC" with a random value (here we set $best_0$<- 0.5)

While $t \le T$ **Do** // Search for best RMC for T synthesis cycles (the first stop criterion, here we set T <- 20)

  if $EI[RMC_{t-1}] \succ EI[RMC_{t-2}]$ // If RMC_{t-1} performs better than RMC_{t-2}

    RMC_t <- $RMC_{t-1} + \alpha \cdot (RMC_{t-1} - RMC_{t-2})$ // Update RMC_t in the improving direction ($\alpha \in [0, 1]$, and here we set α as random values that uniformly distributed in [0, 1])

  else if $best_b \neq RMC_{t-2}$

    RMC_t <- $\beta \cdot best_b + (1 - \beta) \cdot PEI^{-1}[better\{PEI[RMC_{t-2}], PEI[best_{b-1}]\}]$ // Update RMC_t as the linear combination of best RMC and the last best RMC ($best_{b-1}$) iff $best_{b-1}$ performs better than RMC_{t-2}; otherwise update RMC_t as the linear combination of best RMC and RMC_{t-2} ($\beta \in [0,1]$, and here we set β as random values that uniformly distributed in [0.5, 1])

  else if $best_{b-1}$

    RMC_t <- $\beta \cdot best_b + (1 - \beta) \cdot best_{b-1}$ // Update RMC_t as the linear combination of the current best RMC and the last best RMC

    else

      RMC_t <- $\beta \cdot RMC_{t-1} + (1 - \beta) \cdot RMC_{t-2}$

  if $RMC_t > 1$ // If RMC_t is larger than its upper bound

    RMC_t <- 1 // Pull RMC_t back to range

Algorithm A1. *Cont*

if $RMC_t < 0$ // if RMC_t is lower than its lower bound
  $RMC_t = 0$ // Pull RMC_t back to range
I <- 0 // Initiate the number of EIs of RMC_t in DEI
J <- 0 // Initiate the number of EIs of RMC_t that are better than the EIs of best RMC
K <- 0 // Initiate the number of EIs of RMC_t that violate DEI within θ. We tune θ maximizing the $L_2 - norm$ distance between EIs of two adjacent iterations and get 10%
L <- 0 // Initiate the number of EIs of RMC_t that are better than the EIs of RMC_{t-1}
for i in 1 to n // For all the EIs (n is the number of EIs)
  if $PEI[RMC_t]_i \in DEI_i$ // If the i^{th} EI of RMC_t is in the desired range of the i^{th} EI
    I <- I + 1 // Update the number of EIs of RMC_t in DEI
  if $EI[RMC_t]_i \succcurlyeq EI[\text{best}]_i$[11] // If for the i^{th} EI RMC_t performs better than or equal to best RMC
    J <- J + 1 // Update the number of EIs of RMC_t that are better than best RMC
  if $EI[RMC_t]_i \notin DEI_i$ and $EI[RMC_t]_i \in [DEI_i]^{+\gamma\gamma_i}_{-\gamma\gamma_i}$ // If the i^{th} EI of RMC_t violates the desired range of the i^{th} EI within $\gamma\gamma_i$ (in this problem, $\gamma\gamma_i$=30%)
    K <- K + 1 // Update the number of EIs of RMC_t that violate DEI within $\gamma\gamma_i$
  if $EI[RMC_t]_i \succcurlyeq EI[RMC_{t-1}]_i$ // If for the i^{th} EI, RMC_t performs better than or equal to RMC_{t-1}
    L <- L + 1 // Update the number of EIs that RMC_t improves versus RMC_{t-1}
if $I \geq \iota \cdot n$ **and** $I + K = n$ // If for at least ι (we set it as 2/3) EIs are in DEI, and the violation rate are all within $\gamma\gamma_i$
  if $L \geq \kappa \cdot n$ // If at least κ (we set it as 2/3) EIs are better than previous synthesis cycle
    $EI[RMC_t] \succ EI[RMC_{t-1}]$ // We define that RMC_t overall performs better than RMC_{t-1}
  if $J \geq \kappa \cdot n$ // If at least κ EIs is better than best RMC
    $best$ <- RMC_t // Update best RMC
    nupdt <- −1 // Reset no updating pointer "nupdt" as "−1"
  if $K \geq 1$ // If at least one violation EI
    DEI_i <- $EI[RMC_t]_i$ which $EI[RMC_t]_i \notin DEI_i$ and $EI[RMC_t]_i \in [DEI_i]^{+\theta}$ // Update $DEI_{i-\theta}$
    nupdt <- nupdt + 1 // Increase no updating pointer "nupdt" by 1
else
  nupdt <- nupdt + 1 // Increase no updating pointer "nupdt" by 1
if nupdt ≥ M // If no updating in M synthesis cycles in a row (the second stop criterion, and here we set M<−5)
  Break
t <- t+1 // Move on to the next synthesis cycle
Return best // Return the final best RMC as the appropriate RMC

References

1. Rios, L.M.; Sahinidis, N.V. Derivative-Free Optimization: A Review of Algorithms and Comparison of Software Implementations. *J. Glob. Optim.* **2013**, *56*, 1247–1293. [CrossRef]
2. Vrahatis, M.N.; Kontogiorgos, P.; Papavassilopoulos, G.P. Particle Swarm Optimization for Computing Nash and Stackelberg Equilibria in Energy Markets. *SN Oper. Res. Forum* **2020**, *1*, 20. [CrossRef]
3. Behmanesh, E.; Pannek, J. A Comparison between Memetic Algorithm and Genetic Algorithm for an Integrated Logistics Network with Flexible Delivery Path. *Oper. Res. Forum* **2021**, *2*, 47. [CrossRef]
4. Viswanathan, J.; Grossmann, I.E. A Combined Penalty Function and Outer-Approximation Method for MINLP Optimization. *Comput. Chem. Eng.* **1990**, *14*, 769–782. [CrossRef]

5. Nagadurga, T.; Devarapalli, R.; Knypiński, Ł. Comparison of Meta-Heuristic Optimization Algorithms for Global Maximum Power Point Tracking of Partially Shaded Solar Photovoltaic Systems. *Algorithms* **2023**, *16*, 376. [CrossRef]
6. Mistree, F.; Hughes, O.F.; Bras, B. Compromise decision support problem and the adaptive linear programming algorithm. *Prog. Astronaut. Aeronaut. Struct. Optim. Status Promise* **1993**, *150*, 251.
7. Teng, Z.; Lu, L. A FEAST algorithm for the linear response eigenvalue problem. *Algorithms* **2019**, *12*, 181. [CrossRef]
8. Rao, S.; Mulkay, E. Engineering design optimization using interior-point algorithms. *AIAA J.* **2000**, *38*, 2127–2132. [CrossRef]
9. Asghari, M.; Fathollahi-Fard, A.M.; Al-E-Hashem, S.M.; Dulebenets, M.A. Transformation and linearization techniques in optimization: A state-of-the-art survey. *Mathematics* **2022**, *10*, 283. [CrossRef]
10. Reich, D.; Green, R.E.; Kircher, M.; Krause, J.; Patterson, N.; Durand, E.Y.; Viola, B.; Briggs, A.W.; Stenzel, U.; Johnson, P.L. Genetic history of an archaic hominin group from Denisova Cave in Siberia. *Nature* **2010**, *468*, 1053–1060. [CrossRef]
11. Fishburn, P.C. Utility theory. *Manag. Sci.* **1968**, *14*, 335–378. [CrossRef]
12. Nash, J.F., Jr. The bargaining problem. *Econom. J. Econom. Soc.* **1950**, *18*, 155–162. [CrossRef]
13. Saaty, T.L. *What Is the Analytic Hierarchy Process?* Springer: Berlin/Heidelberg, Germany, 1988.
14. Calpine, H.; Golding, A. Some properties of Pareto-optimal choices in decision problems. *Omega* **1976**, *4*, 141–147. [CrossRef]
15. Guo, L. Model Evolution for the Realization of Complex Systems. Ph.D. Thesis, University of Oklahoma, Norman, OK, USA, 2021.
16. Speakman, J.; Francois, G. Robust modifier adaptation via worst-case and probabilistic approaches. *Ind. Eng. Chem. Res.* **2021**, *61*, 515–529. [CrossRef]
17. Souza, J.C.O.; Oliveira, P.R.; Soubeyran, A. Global convergence of a proximal linearized algorithm for difference of convex functions. *Optim. Lett.* **2016**, *10*, 1529–1539. [CrossRef]
18. Su, X.; Yang, X.; Xu, Y. Adaptive parameter learning and neural network control for uncertain permanent magnet linear synchronous motors. *J. Frankl. Inst.* **2023**, *360*, 11665–11682. [CrossRef]
19. Courant, R.; Hilbert, D. *Methods of Mathematical Physics*; Interscience: New York, NY, USA, 1953; Volume 1.
20. Guo, L.; Chen, S. Satisficing Strategy in Engineering Design. *J. Mech. Des.* **2024**, *146*, 050801. [CrossRef]
21. Powell, M.J. An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives. *Comput. J.* **1964**, *7*, 155–162. [CrossRef]
22. Straeter, T.A. *On the Extension of the Davidon-Broyden Class of Rank One, Quasi-Newton Minimization Methods to an Infinite Dimensional Hilbert Space with Applications to Optimal Control Problems*; North Carolina State University: Raleigh, NC, USA, 1971.
23. Fletcher, R. *Practical Methods of Optimization*, 2nd ed.; John Wiley & Sons: Hoboken, NJ, USA, 1987.
24. Khosla, P.; Rubin, S. A Conjugate Gradient Iterative Method. *Comput. Fluids* **1981**, *9*, 109–121. [CrossRef]
25. Nash, S.G. Newton-type Minimization via the Lanczos Method. *SIAM J. Numer. Anal.* **1984**, *21*, 770–788. [CrossRef]
26. Zhu, C.; Byrd, R.H.; Lu, P.; Nocedal, J. Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-scale Bound-Constrained Optimization. *ACM Trans. Math. Softw. (TOMS)* **1997**, *23*, 550–560. [CrossRef]
27. Powell, M. A Tolerant Algorithm for Linearly Constrained Optimization Calculations. *Math. Program.* **1989**, *45*, 547–566. [CrossRef]
28. Powell, M.J. A View of Algorithms for Optimization without Derivatives. *Math. Today Bull. Inst. Math. Its Appl.* **2007**, *43*, 170–174.
29. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
30. Guo, L.; Balu Nellippallil, A.; Smith, W.F.; Allen, J.K.; Mistree, F. Adaptive Linear Programming Algorithm with Parameter Learning for Managing Engineering-Design Problems. In Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Online, 17–19 August 2020; American Society of Mechanical Engineers: New York, NY, USA, 2020; p. V11BT11A029.
31. Chen, W.; Allen, J.K.; Tsui, K.-L.; Mistree, F. A Procedure for Robust Design: Minimizing Variations Caused by Noise Factors and Control Factors. *J. Mech. Des.* **1996**, *118*, 478–485. [CrossRef]
32. Maniezzo, V.; Zhou, T. Learning Individualized Hyperparameter Settings. *Algorithms* **2023**, *16*, 267. [CrossRef]
33. Fianu, S.; Davis, L.B. Heuristic algorithm for nested Markov decision process: Solution quality and computational complexity. *Comput. Oper. Res.* **2023**, *159*, 106297. [CrossRef]
34. Sabahno, H.; Amiri, A. New statistical and machine learning based control charts with variable parameters for monitoring generalized linear model profiles. *Comput. Ind. Eng.* **2023**, *184*, 109562. [CrossRef]
35. Choi, H.-J.; Austin, R.; Allen, J.K.; McDowell, D.L.; Mistree, F.; Benson, D.J. An Approach for Robust Design of Reactive Power Metal Mixtures based on Non-Deterministic Micro-Scale Shock Simulation. *J. Comput. Aided Mater. Des.* **2005**, *12*, 57–85. [CrossRef]
36. Nellippallil, A.B.; Rangaraj, V.; Gautham, B.; Singh, A.K.; Allen, J.K.; Mistree, F. An inverse, decision-based design method for integrated design exploration of materials, products, and manufacturing processes. *J. Mech. Des.* **2018**, *140*, 111403. [CrossRef]
37. Sohrabi, S.; Ziarati, K.; Keshtkaran, M. Revised eight-step feasibility checking procedure with linear time complexity for the Dial-a-Ride Problem (DARP). *Comput. Oper. Res.* **2024**, *164*, 106530. [CrossRef]