European Journal of Operational Research 308 (2023) 1037–1055

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Tabu search exploiting local optimality in binary optimization

Saïd Hanafi [a], Yang Wang [b,*], Fred Glover [c], Wei Yang [b], Rick Hennig [c]

[a] INSA Hauts-de-France / Université Polytechnique des Hauts-de-France, LAMIH, CNRS UMR 8201, Valenciennes 59313, France
[b] School of Management, Northwestern Polytechnical University, 127 Youyi West Road, 710072 Xi'an, China
[c] Entanglement, Inc. Boulder, CO, USA

ARTICLE INFO

Article history:
Received 21 May 2022
Accepted 2 January 2023
Available online 5 January 2023

Keywords:
Metaheuristics
Local optimality
Adaptive memory
Tabu search
Binary optimization
QUBO

ABSTRACT

A variety of strategies have been proposed for overcoming local optimality in metaheuristic search. This paper examines characteristics of moves that can be exploited to make good decisions about steps that lead away from a local optimum and then lead toward a new local optimum. We introduce strategies to identify and take advantage of useful features of solution history with an adaptive memory metaheuristic, to provide rules for selecting moves that offer promise for discovering improved local optima. Our approach uses a new type of adaptive memory based on a construction called exponential extrapolation. The memory operates by means of threshold inequalities that ensure selected moves will not lead to a specified number of most recently encountered local optima. Associated thresholds are embodied in choice rule strategies that further exploit the exponential extrapolation concept and open a variety of research possibilities for exploration. The considerations treated in this study are illustrated in an implementation to solve the Quadratic Unconstrained Binary Optimization (QUBO) problem. We show that the AA algorithm obtains an average objective gap of 0.0315% to the best known values for the 21 largest Palubeckis instances. This solution quality is considered to be quite attractive because less than 20 s on average are taken by AA, which is 1 to 2 orders of magnitude less than the time required by most algorithms reporting the best known results.

1. Introduction

The Tabu search (TS) metaheuristic (Glover (1986)) incorporates adaptive memory and responsive exploration to guide a local search procedure to explore the solution space beyond local optimality. The memory approaches underlying TS are usually based on recency and frequency memories, while the responsive exploration imposes restraints and inducements such as Tabu conditions, aspiration levels, intensification and diversification processes. The principal goal of the adaptive memory framework of TS is to create a balance between search intensification and diversification. Intensification strategies encourage move combinations and solution features historically found good. Diversification strategies incorporate new attributes and attribute combinations that were not included within solutions generated in the past (see the book by Glover and Laguna (1997) for a detailed examination of TS). Several variants of TS have been proposed, including the Tabu cycle method and conditional probability methods (Glover (1989)), as well as the Tabu thresholding methods (Glover (1995)), each being successfully applied to solve hard combinatorial optimization problems (see for example Gendreau and Potvin (2005)), Qiu et al. (2018), Guemri et al. (2019), Servranckx and Vanhoucke (2019), Karamichailidou et al. (2021)).

Recently, Glover (2020) has proposed a new Alternating Ascent (AA) algorithm for exploiting local optimality in metaheuristic search for zero-one programming problems. The present paper focuses on the simplest version of the TS metaheuristic exploiting local optimality in binary optimization, without including intensification and diversification phases, and similarly disregarding path relinking and multi pass strategies.

In outline, the AA Algorithm alternates between an Ascent Phase and a Post-Ascent Phase using thresholds to identify variables to change their values and to transition from one phase to another. A high-level overview of the AA Algorithm (that removes essential features subsequently described) is as follows:

* Corresponding author.
E-mail addresses: Said.Hanafi@uphf.fr (S. Hanafi), yangw@nwpu.edu.cn (Y. Wang), fred@entanglement.ai (F. Glover), yangwei123@mail.nwpu.edu.cn (W. Yang), rick@entanglement.ai (R. Hennig).

https://doi.org/10.1016/j.ejor.2023.01.001
0377-2217/

Overview of an Alternating Ascent (AA) Algorithm
While an outer loop termination criterion is not met **do**
  Choose a starting solution
  While an inner loop termination criterion is not met **do**
    Execute the following two phases:
    Ascent Phase: go to a local optimum
    (which may also be the starting solution on the first pass)
    Post Ascent Phase: move away from the local optimum *and*
      away from some number of other previous local optima.
  Endwhile
Endwhile

In outline, the AA Algorithm alternates between an Ascent Phase and a Post-Ascent Phase using thresholds to identify variables to change their values and to transition from one phase to another. The thresholds embody a form of adaptive memory based on a function called exponential extrapolation, which makes it possible to track the number of times that variables receive their current values in any selected number Q of most recent local optima represented by the set $\mathbb{Q}$. An exponential extrapolation measure $EE_j(\mathbb{Q}, x)$ is associated with a variable x_j that gives rise to a recency threshold of the form $EE_j(\mathbb{Q}, x) \geq Threshold_r(\mathbb{Q})$, which assures that changing the current value for x_j will not duplicate its value in the r most recent local optima. By reference to a standard evaluation $Eval_j(x)$ for x_j that identifies the change in the objective function when x_j changes its value, and taking advantage of a rudimentary tabu search restriction and aspiration criterion, this in turn gives rise to two status conditions denoted by $S^{=}$ and $S^{\neq}$, where an $S^{=}$ status identifies a variable that should change the value it received in the most recent local optimum and an $S^{\neq}$ status identifies a variable that should retain its value that differs from its value received in the most recent local optimum. These conditions are additionally exploited using counters $nS^{=}$ and $nS^{\neq}$ of the number of variables that have an $S^{\neq}$ and $S^{=}$ status, embodied in a trigger threshold of the form $nS^{=} + nS^{\neq} \geq Trigger$. The trigger threshold determines when a new Ascent Phase should be launched by removing all Tabu restrictions except the one that caused the threshold to be satisfied. The resulting ascent first reaches a conditional local optimum where the last Tabu restriction remains in force, and where it is assured that the solution cannot duplicate any of the r most recent local optima. Then this last restriction is also removed to complete the ascent to a true local optimum, and to begin a new Post-Ascent Phase.

Once no more improving moves remain (for the non-Tabu variables) in an Ascent Phase, the resulting ascent reaches a conditional local optimum (subject to keeping x_k at its new value). At this point, we may remove the tabu restriction on x_k as well, to continue to a solution that is a true local optimum which ends the Ascent Phase. Given that the conditional local optimum does not duplicate the previous local optimum, and that the choice of moves leading to this conditional local optimum is influenced by the value assigned to x_k, there is a strong likelihood that the new local optimum will also differ from the previous local optimum.

To exploit this observation, we have to decide of whether to immediately use the change from $Eval_j(x) \leq 0$ to $Eval_j(x) > 0$ to trigger an ascent to a conditional local optimum, or whether to wait until more than one variable x_j selected to be x_k has undergone this change before launching such an ascent.

This study introduces a general procedure for launching a new ascent based on exponential extrapolation to exploit local optimality without recording the local optima. Exponential extrapolation provides a significant saving of both memory and computation over consulting the actual values of variables in previous local optima. Numerical examples are given to illustrate the use of exponential extrapolation and the key processes involved in exploiting local optimality via the recency and trigger thresholds.

2. Background

2.1. Binary optimization problem and move evaluation $Eval_j(x)$

A binary optimization problem can be expressed as follows:

$$(P)\ \begin{cases} maximize & x_0 = f(x) \\ s.t. & x \in X \subseteq \{0,1\}^n \end{cases}$$

where f is a linear or no-linear function on the binary vector x characterized by its components x_i for $i \in N = \{1, \ldots, n\}$ and the feasible set X reprensents the imposed constraints on x.

Given a current binary solution x, a neighbor solution x' is obtained by flipping the value of a single variable from x_j to $x'_j = 1 - x_j$. Hence, we have $x' = x + (1 - 2x_j)e^j$, where e^j is the basic unit vector with all components are zero except the jth component equal to one, and the reverse (complementary) move of setting $x_j = 1$ consists of setting $x_j = 0$, and vice versa. A neighbor solution x' is well determined by only the index $j \in N$ called the attribute of the move where change occurs while other variables are held constant. Let $N(x)$ denote the set of feasible moves which corresponds to N in the special case for the Quadratic Unconstrained Binary Optimization (QUBO) problem, whose formulation is introduced in Section 7, i.e., $N(x)$ defines the neighborhood set of solution x.

We will refer to the use of an evaluation function $Eval_j(x)$ for a binary variable $x_j, j \in N = \{1, \ldots, n\}$, to identify the change in the objective function x_0 at the current solution x produced by reversing the assignment $x_j = x_j^{\#}$, where $x_j^{\#}$ is the current value for x_j. More precisely, the evaluation $Eval_j(x)$ for flipping variable x_j of x that identifies the change in the objective function when x_j changes its value, i.e.

$$Eval_j(x) = x_0^{'} - x_0 = f\left(x'\right) - f(x)$$

Since the objective is to maximize x_0, the sign of $Eval_j(x)$ offers a partition of $N(x)$ $(= N$ for QUBO$)$ into $N^+(x) = \{j \in N : Eval_j(x) > 0\}$, $N^-(x) = \{j \in N : Eval_j(x) < 0\}$ and $N^0(x) = \{j \in N : Eval_j(x) = 0\}$ to differentiate the set of improving moves, worsening (strictly non-improving) moves and simple non-improving moves respectively. In the following, we will refer only to $N^+(x)$ and $N^-(x)$.

Note that the descent method exploits this partition by choosing at each iteration a move from $N^+(x)$ or $N^+(x) \cap N^0(x)$ until $N^+(x) = \emptyset$ or $N^+(x) \cap N^0(x) = \emptyset$. Hence, the final solution of a descent method is a local optimum x such all move evaluations $Eval_j(x)$ are non-positive, yielding $Eval_j(x) \leq 0$ for all variables x_j, i.e. $N^+(x) \cap N^0(x) = \emptyset$. For example, the steepest descent local search method selects at each iteration a variable x_k such that

$$k = argmax\{Eval_j(x) : j \in N^+(x) \cap N^0(x)\}.$$

2.2. Adaptive memory Tabu search approach

An early experiment with metaheuristic search (Laguna and Glover (1993)) for a class of sequencing problems disclosed that improving moves were more likely to select attributes of optimal solutions than non-improving moves. This was notably reflected in the fact that moves made when approaching a local optimum were more likely to create solutions that shared elements in common with optimal solutions than moves made when retreating from a local optimum. We are motivated by this study to change the rules customarily used by metaheuristic procedures to provide new approaches for responding to local optimality. Our focus is on using adaptive memory strategies that incorporate special threshold inequalities to guide the search. As a starting point, consider a method that begins from a local optimum and employs rules of

the following types, which are commonly employed in a rudimentary form of an adaptive memory Tabu search approach.

Tabu Rule: When reaching a local optimum and selecting moves that lead away from this optimum, employ restrictions that temporarily do not allow moves to be reversed and hence that would potentially return to the local optimum.

Aspiration Rule: Identify conditions for modifying Tabu Rule to permit certain moves to be made that violate the restrictions and allow previous moves to be reversed.

We observe that a simple form of Tabu search is often based on a version of these two rules that has two features. We describe these for the purpose of identifying a different way to apply Tabu and Aspiration Rules.

Tabu Feature: A tenure value is used to prevent a move from being reversed for *Tenure* iterations, thereby making the reverse move tabu by reference to the current iteration *Iter*, by setting

$$TabuIter(ReverseMove) = Iter + Tenure$$

where *TabuIter* is an array representing recency memory. In the case of binary optimization, the reverse move of $x_j = 1$ is $x_j = 0$, and vice versa. Then $TabuIter(ReverseMove)$ can be represented simply by $TabuIter(j)$, with the interpretation that

$$Iter \leq TabuIter(j)$$

means x_j is tabu to change its current value $x_j = x_j^\#$ to the reverse (complementary) value $x_j = 1 - x_j^\#$.

Aspiration Feature: The rule for choosing moves selects a highest evaluation move that is not Tabu or that satisfies an aspiration criterion. Since all moves at local optimality cause the objective function to deteriorate or remain unchanged, a highest evaluation move is one that causes the objective function to deteriorate the least. The aspiration criterion most commonly employed considers a Reverse Move to be admissible to be chosen if it leads to a solution better than the best one found so far.

A Tabu search approach that relies more fully on adaptive memory refines the partition of the neighborhood set $N(x)$ by involving the set $N^{\bar{T}}(x) = \{j \in N : TabuIter(j) < Iter\}$ which corresponds to the set of non-Tabu moves and the set $N^A(x) = \{j \in N : x_0 + Eval_j(x) > x_0^*\}$ where x_0^* is the best objective function found so far, to the set of the moves satisfying the aspiration criterion.

Several variants of Tabu Search algorithm have been proposed in the literature exploiting partitions involving the sets $N^-(x), N^0(x), N^+(x), N^{\bar{T}}(x)$ and $N^A(x)$. For example, the simplest form of tabu search without intensification or diversification, results where the default aspiration is applied when $N^{\bar{T}}(x) = \emptyset$ to choose the least tabu move, i.e. $k \in Argmin\{TabuIter(j) : j \in N(x)\}$. Then the search selects at each iteration a variable x_k according to the following rule

If $N^A(x) \neq \emptyset$ **then** select $k \in Argmax\{Eval_j(x) : j \in N^A(x)\}$
Else If $N^{\bar{T}}(x) \neq \emptyset$ **then** $k \in Argmax\{Eval_j(x) : j \in N^{\bar{T}}(x)\}$
Else $k \in Argmin\{TabuIter(j) : j \in N^{\bar{T}}(x)\}$
EndIf

This simplest deterministic Tabu search algorithms based on the recency or frequency memories are proved to converge to an optimal solution in finite number of iterations (see Hanafi, 2001, Glover & Hanafi, 2002). Faigle and Kern (1992) proposed some convergence results for Probabilistic Tabu Search.

The target analysis experiment described in Laguna and Glover (1993), shows that something about the combination of Tabu Feature and Aspiration Feature when moving away from a local optimum tends to produce moves whose attributes do not correspond to those of an optimal solution. Under the assumption that this finding is applicable to other settings, this motivates an examination of versions of Tabu and Aspiration Rules that modify Tabu and Aspiration Features to produce a different behavior.

Alternative forms of Tabu and aspiration rules. As a starting point for analyzing conditions that hold at a local optimum x, all move evaluations are non-positive when a local optimum is reached, i.e., $Eval_j(x) \leq 0$ for all variables x_j. In the situation where $Eval_j(x) \leq 0$, suppose we assign a tabu tenure to a move that changes $x_j = x_j^\#$ to $x_j = 1 - x_j^\#$ as is customarily done to prevent the move from being immediately reversed. Consider the process that takes place at this point, as the search begins moving away from a local optimum. To begin, all moves selected will consist of reversing values received by variables $x_j = x_j^\#$, in the local optimum to produce new assignments $x_j = 1 - x_j^\#$, and given $Eval_j(x) \leq 0$, these will cause x_0 to decrease or remain unchanged. After reversing an assignment for $Eval_j(x) \leq 0$, the new evaluation $Eval_j(x)$ will be the negative of its previous value, and hence if $Eval_j(x)$ began negative it will now be positive. However, the improving move that returns x_j to its previous value will be prevented because of the tabu tenure assigned to it. We will build on these simple observations to uncover aspects of adaptive memory choices that have previously been overlooked.

Overriding Tabu restrictions. As previously intimated, a key question to be addressed in developing an effective algorithm is how to usefully override the customary Tabu restriction by freeing certain variables so they are no longer Tabu. Accompanying this question is the associated question of identifying the circumstances under which this override should be done. An answer to these questions is suggested by considering the situation where the evaluation for a variable x_j changes from $Eval_j(x) \leq 0$ to $Eval_j(x) > 0$ when moving away from a local optimum, without having assigned a new value to x_0. We are prompted to ask whether there something noteworthy about this change from a non-improving evaluation to an improving evaluation during a sequence of iterations after reaching a local optimum.

If the current value $x_j^\#$ of x_j is also the value x_0 received at the local optimum (when $Eval_j(x) \leq 0$), and if now $Eval_j(x) > 0$, then this has the significant feature that the profitable (i.e., improving) move $x_j = 1 - x_j^\#$ gives x_j a different value than it had at the local optimum. If x_j is selected as the variable x_k that changes its value on the current iteration, then by making x_k tabu to change its value, the search cannot return to the local optimum while x_k remains tabu. Consequently, we are motivated to consider the result of freeing the Tabu restrictions on all variables x_j except for x_k, to launch an ascent in which the procedure cannot return to the previous local optimum. We call the iterations that occur upon launching such an ascent until reaching a new local optimum an Ascent Phase.

Once no more improving moves remain (for the non-Tabu variables) in an Ascent Phase, the resulting ascent reaches a conditional local optimum (subject to keeping x_k at its new value). At this point, we may remove the tabu restriction on x_k as well, to continue to a solution that is a true local optimum which ends the Ascent Phase. Given that the conditional local optimum does not duplicate the previous local optimum, and that the choice of moves leading to this conditional local optimum is influenced by the value assigned to x_k, there is a strong likelihood that the new local optimum will also differ from the previous local optimum.

To exploit this observation, we are presented with the decision of whether to immediately use the change from $Eval_j(x) \leq 0$ to $Eval_j(x) > 0$ to trigger an ascent to a conditional local optimum, or whether to wait until more than one variable x_j selected to be x_k has undergone this change before launching such an ascent. We examine this issue in a broader context in the next section.

3. A more general procedure for launching a new ascent

Instead of only considering the most recent local optimum, the issue of identifying a variable x_j to change its value in this local optimum can be generalized to refer to some number Q of the most recent local optima. We describe a way of doing this that makes it possible to maintain appropriate updated information without recording the local optima. This approach, called exponential extrapolation, provides a significant saving of both memory and computation over consulting the actual values of variables in previous local optima.

3.1. Exponential extrapolation $EE_j(\mathbb{Q}, x)$

The term "exponential extrapolation" is motivated by the term "exponential smoothing," which refers to a procedure that choses a value λ between 0 and 1 and uses the simple formula

$$y_{q+1} = \lambda y_q + (1-\lambda) y_{q-1}$$

to determine the new value of y_{q+1} based on the two preceding values y_q and y_{q-1}. The procedure can start from chosen values y_q for $q = 0$ and 1. (More precisely, y_{q+1} and y_{q-1} refer to forecast values and y_q refers to an observed value. We do not require this distinction here.) Exponential extrapolation instead uses the formula, expressed in terms of the weights w_q

$$w_{q+1} = \alpha w_q + \beta q + \gamma \tag{1}$$

where we choose $w_1 = 1$. For simplicity, the parameters α, β and γ may be restricted to α between 1 and 2, and β and γ between 0 and 3. Even simpler, we will chiefly focus on the special case $\alpha = 2$ and $\beta = \gamma = 0$. It is possible to establish a connection between exponential extrapolation and exponential smoothing whereby (1) can be seen as a generalization of exponential smoothing, but we will not pursue this here. Exponential smoothing has been applied with Tabu search for solving fixed charge network problems in Barr et al. (2021), using a different type of design than we use for exploiting exponential extrapolation, but we note that exponential extrapolation affords an alternative to exponential smoothing in the fixed charge setting too.

For the special case of (1) where $\alpha > 0$ and $\beta = \gamma = 0$ we are particularly interested in the situation where $= 2$, to give

$$w_{q+1} = 2w_q = 2^q \tag{2}$$

In general, the formula $w_{q+1} = \alpha w_q$ can be expressed as $w_{q+1} = \alpha^q$ for $q \geq 0$.

Denote the set of Q most recent local optima by $\mathbb{Q} = \{x^q : q = 1, \ldots, Q\}$, with $x^q = (x_1^q, \ldots, x_n^q) \in \{0,1\}^n$. Let $x \in \{0,1\}^n$ denotes a binary solution, in the following we refer to exponential extrapolation by the acronym EE and we are interested in weighted $EE(\mathbb{Q}, x)$ values for each variable x_j:

$$EE_j(\mathbb{Q}, x) = \begin{cases} EE_j^1(\mathbb{Q}) = \sum_{q=1}^{Q} 2^{q-1} x_j^q & if\ x_j = 1 \\ EE_j^0(\mathbb{Q}) = \sum_{q=1}^{Q} 2^{q-1}\left(1 - x_j^q\right) & if\ x_j = 0 \end{cases} \tag{3}$$

The $EE_j^1(\mathbb{Q})$ value weights the values x_j^q that equal 1 in the vectors x^q, $q = 1$ to Q, and $EE_j^0(\mathbb{Q})$ which weights the values x_j^q that equal 0 in these vectors. By defining

$$EEbase(\mathbb{Q}) = \sum_{q=1}^{Q} 2^{q-1} = 2^Q - 1 \tag{3.1}$$

We have a useful equation

$$EEbase(\mathbb{Q}) = EE_j^0(\mathbb{Q}) + EE_j^1(\mathbb{Q}) \tag{3.2}$$

For each $q = 2$ to Q, we have $w_q = \sum_{h=1}^{q-1} w_h + 1 = \sum_{h=1}^{q-1} 2^{h-1} + 1 = 2^{q-1}$. Hence w_q is greater than $\sum_{h=1}^{q-1} w_h$, and as a special case $w_Q > \sum_{q=1}^{Q-1} w_q$. Consequently, the value $EE_j^1(\mathbb{Q})$ will be larger when $x_j^Q = 1$ than it will be when $x_j^Q = 0$, regardless of the values w_q for $q < Q$. Another way of expressing this is that (2) creates a lexicographic ordering of the binary value assignments to the variables x_j^q, where the value $EE_j^1(\mathbb{Q})$ is larger as the vector $V_j(\mathbb{Q}) = (x_j^Q, x_j^{Q-1}, \ldots, x_j^1)$ increases lexicographically. For our purposes, this means that by using the EE value $EE_j^1(\mathbb{Q})$, the most recent local optimum recorded x^Q will dominate any combination of all other local optima, and the second most recent local optimum x^{Q-1} will dominate any combination of all local optima preceding it, and so forth. A useful implication is that if we require that we only select a variable x_j to change its value from $x_j^\#$ to $1 - x_j^\#$ if $x_j = x_j^\#$ in the r most recent local optima by stipulating

$$EE_j(\mathbb{Q}, x) \geq Threshold_r(\mathbb{Q}) \tag{4}$$

where the *threshold value* sums the r largest weights given by

$$Threshold_r(\mathbb{Q}) = \sum_{q=1}^{r} 2^{Q-q} = 2^{Q-r}(2^r - 1) \tag{4.0}$$

We call the inequality (4) the *recency threshold.* We treat recency threshold as embodying the two inequalities

$$EE_j^1(\mathbb{Q}) \geq Threshold_r(\mathbb{Q}) \tag{4.1}$$

$$EE_j^0(\mathbb{Q}) \geq Threshold_r(\mathbb{Q}) \tag{4.2}$$

where (4.1) applies to $x_j = 1$ and requires that x_j cannot be chosen to change from 1 to 0 unless x_j also equals 1 in each of the r most recent local optima, and (4.2) applies to $x_j = 0$ and requires that x_j cannot be chosen to change from 0 to 1 unless x_j also equals 0 in each of the r most recent local optima.

In short, by requiring the recency threshold (4) to be satisfied (for any choice of the index j, and for a specified value $x_j = x_j^\#$), we assure that we will not risk duplicating any of the r most recent local optima by changing x_j to equal $1 - x_j^\#$. The utility of this requirement is that we do not need to record the most recent local optima to verify – or compel – that $x_j = 1$ or 0 in any specified number r of these most recent solutions. All that is necessary is to specify that $EE_j^1(\mathbb{Q})$ satisfy (4.1) or that $EE_j^0(\mathbb{Q})$ satisfy (4.2), according to whether $x_j^\# = 1$ or 0. More precisely, we have the following proposition.

Proposition 1. *Let x be a binary solution where for any index $j \in N$, a specified value is assigned $x_j = x_j^\#$, the recency threshold inequality*

$$EE_j(\mathbb{Q}, x) \geq Threshold_r(\mathbb{Q})$$

implies that $x_j^\# = x_j^q$ in any solution x^q for $q = Q-1, \ldots, Q-r+1$.

Justification. See Appendix 1. Implications of the recency threshold for using different α values.

Moreover, this approach can grow Q to a selected value Q_{max}, and then perform a diversification step such as the focal distance diversification strategy of Glover and Lu (2020) to start over again with $Q = 1$. By using more than one set of values for the parameters α, β and γ (or even just changing the value of the parameter α as in the strategies with $\beta = \gamma = 0$), these parameters can be applied for different Q_{max}, values, so that when one set is renewed by starting over another set will continue to apply to earlier local optima until its Q_{max}, value is reached. This "staggered" approach can then permit a parameter set that is renewed before another one to

continue in operation when the second is renewed, and so on, so that there is always a connection to local optima from a relatively long period ago. In the case where $\beta = \gamma = 0$, which permits the inductive update, the use of real arithmetic allows the "effective" value of Q_{max}, to be quite large while keeping Q constant.

Complementary recency threshold. Let $\bar{\mathbb{Q}} = \{\bar{x} : x \in \mathbb{Q}\}$, it is easy to check the useful properties:

- $EE_j(\mathbb{Q}, \bar{x}) = EEbase(\mathbb{Q}) - EE_j(\mathbb{Q}, x)$ $(\overline{4.1})$
- $EEbase(\mathbb{Q}) = EEbase(\bar{\mathbb{Q}})$
- $EE_j^0(\mathbb{Q}) = EE_j^1(\bar{\mathbb{Q}})$
- $EE_j^1(\mathbb{Q}) = EE_j^0(\bar{\mathbb{Q}})$
- $EE_j(\bar{\mathbb{Q}}, x) = EE_j(\mathbb{Q}, \bar{x})$
- $EE_j(\mathbb{Q}, \mathrm{x}) = EE_j(\bar{\mathbb{Q}}, \bar{x})$

Moreover, for any binary vector x and x', we have

$$EE_j\big(\mathbb{Q}, x'\big) = \big(1 - \big|x'_j - x_j\big|\big)EE_j(\mathbb{Q}, x) + \big|x'_j - x_j\big|EE_j(\mathbb{Q}, \bar{x}) \quad (4.3)$$

Or equivalently

$$EE_j\big(\mathbb{Q}, x'\big) = \begin{cases} EE_j(\mathbb{Q}, x) & if\ x'_j = x_j \\ EE_j(\mathbb{Q}, \bar{x}) = EEbase(\mathbb{Q}) - EE_j(\mathbb{Q}, x) & if\ x'_j{=}1{-}x_j \end{cases} \quad (4'.3)$$

Consequently, the recency threshold (4) is equivalent to

$$EE_j(\mathbb{Q}, \bar{x}) \le EEbase(\mathbb{Q}) - Threshold_r(\mathbb{Q}) \quad (\bar{4})$$

Finally, we note that $Threshold_r(\mathbb{Q})$ is independent of the value assigned to x_j and define its complement by

$$\overline{Threshold}_r(\mathbb{Q}) = EEbase(\mathbb{Q}) - Threshold_r(\mathbb{Q})$$

which similarly yields $Threshold_r(\mathbb{Q}) = EEbase(\mathbb{Q}) - \overline{Threshold}_r$

From these definitions it may be verified that the recency threshold $EE_j(\mathbb{Q}, x) \ge Threshold_r(\mathbb{Q})$ of (4) gives rise to the *complementary recency threshold* (in the opposite direction)

$EE_j(\mathbb{Q}, \bar{x}) \le \overline{Threshold}_r(\mathbb{Q})$ $(\bar{4})$

The significance of $(\bar{4})$ is that whenever the recency threshold $EE_j(\mathbb{Q}, x) \ge Threshold_r(\mathbb{Q})$ of (4) holds and x_j is chosen to change its value from $x_j^{\#}$ to $1 - x_j^{\#}$, after the assignment, for the new value of x_j, we will have

$$EE_j(\mathbb{Q}, x) \le \overline{Threshold}_r(\mathbb{Q}) = EEbase(\mathbb{Q}) - Threshold_r(\mathbb{Q}) \quad (4^*)$$

From the definitions $EEbase(\mathbb{Q}) = \sum_{q=1}^{Q} 2^{q-1} = 2^Q - 1$ and $Threshold_r(\mathbb{Q}) = \sum_{q=1}^{r} 2^{Q-q} = 2^{Q-r}(2^r - 1)$, the quantity $\overline{Threshold}_r(\mathbb{Q})$ can also be written

$$\overline{Threshold}_r(\mathbb{Q}) = \sum_{q=1}^{Q-r} 2^{q-1} = 2^{Q-r} - 1$$

which is evidently much smaller than $Threshold_r(\mathbb{Q})$ (since $2^{Q-r} > \overline{Threshold}_r(\mathbb{Q})$ by the relationship $2^{Q-r} = \sum_{q=1}^{Q-r} 2^{q-1} + 1$). Hence when the recency threshold is satisfied for $x_j = x_j^{\#}$, (4^*) implies the threshold cannot be satisfied after changing x_j to $1 - x_j^{\#}$. The converse is also true, if $EE_j(\mathbb{Q}, x) \le \overline{Threshold}_r(\mathbb{Q})$ is satisfied for $x_j = x_j^{\#}$, then the recency threshold will be satisfied when x_j is changed to equal $1 - x_j^{\#}$.

The AA approach refines the partition of the neighborhood set $N(x)$ by introducing the set $N^E(x) = \{j \in N : EE_j(\mathbb{Q}, x) \ge Threshold_r(\mathbb{Q})\}$ that identifies moves satisfying the recency threshold inequality and the set $N^{\bar{E}}(x) = \{j \in N : EE_j(\mathbb{Q}, x) \le Ebase(\mathbb{Q}) - Threshold_r(\mathbb{Q})\}$ that identifies reverse moves satisfying the recency threshold inequality.

An inductive updating formula. Now we show how to conveniently update the value $EE_j(\mathbb{Q}, x)$ after adding a new local optimum x^{Q+1} to $\mathbb{Q}$ or adding the new x^{Q+1} while simultaneously dropping the first local optimum x^1 from $\mathbb{Q}$ to maintains the number of local optima Q constant. We write $EE_j(\mathbb{Q}, x)$ with the most recent value Q first:

$$EE_j(\mathbb{Q}, x) = \sum_{q=1}^{Q} 2^{q-1}\big(1 - \big|x_j - x_j^q\big|\big)$$

Adding a new local optimum x^{Q+1} at the end of the current set $\mathbb{Q}$ without dropping any solution from $\mathbb{Q}$ is achieved by

$$EE_j\big(\mathbb{Q} + x^{Q+1}, x\big) = \sum_{q=1}^{Q+1} 2^{q-1}(1 - \big|x_j - x_j^q\big|)$$

$$EE_j\big(\mathbb{Q} + x^{Q+1}, x\big) = EE_j(\mathbb{Q}, x) + 2^Q\big(1 - \big|x_j - x_j^{Q+1}\big|\big)$$

Dropping the first local optimum x^1 from the current set $\mathbb{Q}$ without dropping any solution from $\mathbb{Q}$ is achieved by

$$EE_j\big(\mathbb{Q} - x^1, x\big) = \sum_{q=2}^{Q} 2^{q-2}(1 - \big|x_j - x_j^q\big|)$$

$$EE_j\big(\mathbb{Q} - x^1, x\big) = 2^{-1}EE_j(\mathbb{Q}, x) - 2^{-1}\big(1 - \big|x_j - x_j^1\big|\big)$$

Adding the new x^{Q+1} while simultaneously dropping the first local optimum x^1 from $\mathbb{Q}$ to maintains the number of local optima Q constant, is achieved by

$$EE_j\big(\mathbb{Q} - x^1 + x^{Q+1}, x\big) = 2^{-1}EE_j(\mathbb{Q}, x) + 2^{Q-1}\big(1 - \big|x_j - x_j^{Q+1}\big|\big) - 2^{-1}\big(1 - \big|x_j - x_j^1\big|\big) \quad (5)$$

If we use integer arithmetic that rounds fractional values less than 1 down to 0, the update Eq. (5) becomes

$$EE_j\big(\mathbb{Q} - x^1 + x^{Q+1}, x\big) = 2^{-1}EE_j(\mathbb{Q}, x) + 2^{Q-1}\big(1 - \big|x_j - x_j^{Q+1}\big|\big) \quad (5.1)$$

If real (floating point) arithmetic is used instead of integer arithmetic, the declining influence of earlier x_j values will proceed in the same manner as if Q had been chosen to be larger, or equivalently as if we allowed q to become negative, with each weight $w_{q-1} = 2^{-1}w_q$. By the relationship $w_q = 2^{q-1}$ this corresponds to the weights 2^{-1}, 2^{-2}, 2^{-3}, ... and so forth. The values $q = 0, -1, -2, \ldots$ need not be created or accessed, of course, since they are merely a notational convention to convey how using real arithmetic will have the same effect as permitting Q to be larger. This can be relevant when using α values different than 2, as discussed in Appendix 1.

The inductive update conveniently permits us to start with Q at its maximum desired value and use the formula (5.1) at each iteration of generating a new local optimum to update $EE_j(\mathbb{Q}, x)$. Until Q local optima have been generated, $EE_j(\mathbb{Q}, x)$ will not refer to terms that go all the way back to $q = 1$. For example, after generating s local optima for $s < Q$, $EE_j(\mathbb{Q}, x)$ determined by (5.1) will effectively yield $EE_j(\mathbb{Q}, x) = \sum_{q=s}^{Q} 2^{q-1}(1 - |x_j - x_j^q|)$.

Hence, if we want to apply the recency threshold to assure $x_j = x_j^{\#}$ in the r most recent local optima, we must remember that r cannot exceed s. Fortunately, the inductive update handles this automatically.

First, we observe that when the first local optimum is obtained, both $EE_j(\mathbb{Q}, x)$ and $EEbase(\mathbb{Q})$ can be determined by setting

$$EEbase(\mathbb{Q}) = 2^{Q-1} \quad (5.2a)$$

$$EE_j(\mathbb{Q}, x) = 2^{Q-1}\big(1 - \big|x_j - x_j^Q\big|\big) \text{ for } j = 1 \ = \ 1 \text{ to } n \quad (5.2b)$$

The update formula (5.1) can be simplified if the evaluation $EE_j(\mathbb{Q}, x)$ refers to the last local optimum x^Q. Let

$$EE_j\big(\mathbb{Q}, x^Q\big) = \sum_{q=1}^{Q} 2^{q-1}\big(1 - \big|x_j^Q - x_j^q\big|\big).$$

From (5.1) we have

$$EE_j\big(\mathbb{Q} - x^1 + x^{Q+1}, x^{Q+1}\big) = 2^{-1}EE_j\big(\mathbb{Q}, x^{Q+1}\big) + 2^{Q-1}$$

While from (4.3) we have

$$EE_j\big(\mathbb{Q}, x^{Q+1}\big) = \begin{cases} EE_j\big(\mathbb{Q}, x^Q\big) & if\ x_j^{Q+1} = x_j^Q \quad (5.2c) \\ EE_j\big(\mathbb{Q}, \overline{x^Q}\big) & if\ x_j^{Q+1} = 1 - x_j^Q \quad (5.2d) \end{cases}$$

Hence

$$EE_j\big(\mathbb{Q} - x^1 + x^{Q+1}, x^{Q+1}\big) = \begin{cases} 2^{Q-1} + 2^{-1}EE_j\big(\mathbb{Q}, x^Q\big) & if\ x_j^{Q+1} = x_j^Q \\ 2^{Q-1} + 2^{-1}EE_j\big(\mathbb{Q}, \overline{x^Q}\big) & if\ x_j^{Q+1} = 1 - x_j^Q \end{cases}$$

This update can be expressed by the inductive formula

$$EE_j\big(\mathbb{Q} - x^1 + x^{Q+1}, x^{Q+1}\big) = 2^{Q-1} + 2^{-1}EE_j\big(\mathbb{Q}, x^Q\big)$$

Due to the complementary relationship expressed in (4.1) and if the new local optimum x^{Q+1} corresponds to the current solution x, the update (5) can then be expressed by

$$EE_j\big(\mathbb{Q} - x^1 + x^{Q+1}, x\big) = \begin{cases} 2^{Q-1} + 2^{-1}EE_j\big(\mathbb{Q}, x^Q\big) & if\ x_j^{\#} = x_j^Q \\ 2^{Q-1} + 2^{-1}\big(EEbase(\mathbb{Q}) - EE_j\big(\mathbb{Q}, x^Q\big)\big) & if\ x_j^{\#} = 1 - x_j^Q \end{cases} \quad (5.2)$$

We can perform this update without having saved the value x_j^Q since each time a variable x_k changes its current value $x_k^{\#}$ by setting $x_k = 1 - x_k^{\#}$ during the iterations between obtaining successive local optima, the value $EE_k(\mathbb{Q}, x^Q)$ is updated by setting

$$EE_k\big(\mathbb{Q}, x^Q\big) = EEbase(\mathbb{Q}) - EE_k\big(\mathbb{Q}, x^Q\big) \quad (5.3)$$

which is essential to assure that the update Eq. (5.2) is equivalent to

$$EE_j(\mathbb{Q}) = 2^{Q-1} + 2^{-1}EE_j(\mathbb{Q})$$

Following this update, $EEbase(\mathbb{Q})$ itself can also be updated by the inductive formula

$$EEbase(\mathbb{Q}) = 2^{Q-1} + 2^{-1}EEbase(\mathbb{Q}) \quad (5.4)$$

By consequence, the initialization step (5.2b) becomes

$$EE_j(\mathbb{Q}) = 2^{Q-1} \text{ for } j = 1 \text{ to } n \quad (5.5)$$

Finally, rather than wait until obtaining a first local optimum as a basis for determining the first $EEbase(\mathbb{Q})$ and $EE_j(\mathbb{Q})$ values by (5.1), we can perform the following simple initialization to precede the first iteration of the algorithm

$$EEbase(\mathbb{Q}) = 0 \text{ and } EE_j(\mathbb{Q}) = 0 \text{ for } j = 1 \text{ to } n \quad (5.6)$$

Then the updates of (5.2), (5.3) and (5.4) will automatically yield the correct values for $EEbase(\mathbb{Q})$ and $EE_j(\mathbb{Q})$ given by (5.1) when the first local optimum is obtained.

In the same way, the value of $Threshold_r$ can only refer to the s most recent local optima when $s < r$. We can inductively update $Threshold_r$ by letting

$$Threshold_r = \min\big(EEbase(\mathbb{Q}), 2^{Q-r}(2^r - 1)\big) \quad (5.7)$$

where $2^{Q-r}(2^r - 1)$ corresponds to $Threshold_r = \sum_{q=Q-r+1}^{Q} 2^{q-1}$ where once r is selected.

3.2. Exploiting local optimality based on exponential extrapolation

We refer to the iterations that begin upon reaching a local optimum with an Ascent Phase as a Post-Ascent Phase. We are interested in two principal strategies to guide the Post-Ascent Phase. Each depends on the existence of an opportunity to interrupt the search by removing Tabu restrictions and then to proceed to a new local optimum.

Status $S^=$.

We refer to three key conditions that may be satisfied by a variable x_j after a local optimum is reached (where, to begin, $Eval_j(x) \le 0$ for all variables x_j).

(i) $x_j^{\#} = x_j^Q$.
(ii) $Eval_j(x) > 0$ and x_j is not tabu.
(iii) $EE_j(\mathbb{Q}, x) \ge Threshold_r(\mathbb{Q})$.

A variable x_j that satisfies (i) and (ii) will be said to have an S^+ *status*. The "S" in S^+ simply refers to "Status," while the "+" refers to the fact that $Eval_j(x) > 0$, which implies that changing the value of x_j will produce an improvement in the objective function x_0. A variable with an S^+ status will be given a higher priority to be selected as the variable x_k than a variable that does not have an S^+ status. In other words, any non-tabu variable with a profitable evaluation has a higher priority of receiving a new value than one with a non-profitable evaluation.

A variable x_j that satisfies all three conditions (i), (ii) and (iii) (or equivalently, the conditions (i) and (iii)) will be said to have an $S^=$ status. The $S^=$ status dominates the S^+ status by having a higher priority to be chosen as the variable x_k that receives a new value. Note this implies that the recency threshold acts like an aspiration criterion that overrides a tabu restriction to allow a variable x_j to be selected when $Eval_j(x) > 0$. We do not allow this to happen when $Eval_j(x) \le 0$. The importance of the $S^=$ status is that it means that the choice of x_j to become x_k can participate in a decision to trigger an assent to a new local optimum, as described below. The way that the $S^=$ status contributes to this process is as follows. The recency threshold $EE_j(\mathbb{Q}, x) \ge Threshold_r(\mathbb{Q})$ implies that $x_j = x_j^{\#}$ in each of the r most recent local optima. Hence if x_j is selected as x_k to set $x_k = 1 - x_k^{\#}$ on the current move, the resulting solution cannot move toward any of these local optima. An $S^=$ status is *realized* by assigning x_k its new profitable value, thereby causing its new $x_k^{\#}$ value to be the complement of its current value.

The value of r must be chosen large enough (analogous to the choice of a tabu tenure in tabu search) to drive the search away from an appropriate number of previous local optima. At the same time, the inequality $EE_j(\mathbb{Q}, x) \ge Threshold_r(\mathbb{Q})$ is stronger than necessary to avoid visiting these r local optima. Consequently, it can be preferable to avoid making r too large, which may unduly restrict the new solutions that can be reached. (This observation also suggests that it may be valuable to explore options for setting α smaller than 2. This issue is examined in Appendix 1.)

The principal observation to be made at present is that the greater the number of variables x_j that receive an $S^=$ status and that have been chosen to be x_k, the greater is the motive for triggering an ascent to a new local optimum.

Status $S^{\neq}$.

The second strategy arises where a variable x_j satisfies the following conditions:

(i) $x_j^{\#} \neq x_j^Q$.
(ii) $Eval_j(x) < 0$.
(iii) $EE_j(\mathbb{Q}, x^Q) \ge Threshold_r(\mathbb{Q})$.

Note that a variable x_j that satisfies $x_j^{\#} \neq x_j^Q$ will have been made tabu after reaching the local optimum x^Q by the customary approach of making any variable tabu when it changes its value.

The variable will continue to be tabu unless its *Tenure* value has expired in the interim after receiving this new value. We will suppose that we make *Tenure* large enough to avoid this eventuality. When the move occurred to change x_j's value to x_j^Q, the original evaluation $Eval_j(x) \leq 0$ would have reversed its sign to become $Eval_j(x) \geq 0$ (which, if $Eval_j(x) > 0$, would have made x_j profitable to change back to its previous value except for the Tabu restriction). The current evaluation $Eval_j(x) < 0$ by (v) contrasts with the situation that created (iv). This current evaluation is therefore consistent with considering the previous change of x_j to have been a profitable move rather than a non-profitable move (since the negation of $Eval_j(x)$ for a profitable move would cause $Eval_j(x) < 0$ as in (v)).

A variable x_j that satisfies all three conditions (iv), (v) and (vi) will be said to have an $S^{\not=}$ status. The importance of the $S^{\not=}$ status is that, like an $S^{=}$ status, it qualifies x_j participate in a decision to trigger an assent to a new local optimum. (This results from the fact that the move that has given x_j its new value $x_j^{\#}$ receives an evaluation as if it had originally been profitable and, in addition, causes x_j to satisfy the recency threshold for $x_j = x_j^Q$.) In sum, as we indicate below, once a sufficient number of variables have either an $S^{=}$ status or an $S^{\not=}$ status, then these variables activate an Ascent Phase that proceeds to a new local optimum. It may be seen that (vi) is identical to (iii) by noting that $x_j^{\#}$ in (iii) also corresponds to x_j^Q. The difference between (iii) and (vi) is that the value x_j^Q in (vi) differs from the current value $x_j^{\#}$ for x_j. Because variables are made tabu when they are assigned a new value after reaching a local optimum, this implies that an $S^{=}$ status deals with the case where x_j has not yet changed its local optimum value and an $S^{\not=}$ status deals with the case where such a change has already occurred.

In short, the new value not yet assigned to x_j in (iii) and the new value already assigned to x_j in (vi) must be different than the value of x_j in each of the r most recent local optima, and hence by receiving this value, the current solution moves in a direction away from these most recent local optima. It should also be observed that $EE_j(\mathbb{Q}, x^Q)$ is determined in (vi) refer to the complement $EE_j(\mathbb{Q}, \bar{x})$. As shown earlier in (4*), this implies that the inequality of (vi) becomes

$$EE_j(\mathbb{Q}, x) \leq EEbase(\mathbb{Q}) - Threshold_r(\mathbb{Q})$$

This is relevant for choice rules, since it further implies that the value of $EE_j(\mathbb{Q}, x)$ will be relatively small than $EE_j(\mathbb{Q}, x^Q)$, and in general, just as $EE_j(\mathbb{Q}, x)$ benefits from being larger in order for the recency threshold $EE_j(\mathbb{Q}, x) \geq Threshold_r(\mathbb{Q})$ to hold (which establishes $S^{=}$ status when $Eval_j(x) > 0$ and makes it desirable to select x_j to become x_k and change its value), the corresponding inequality $EE_j(\mathbb{Q}, \bar{x}) \leq EEbase(\mathbb{Q}) - Threshold_r(\mathbb{Q})$ from (vi) shows that when $Eval_j(x) \leq 0$, a smaller $EE_j(\mathbb{Q}, x)$ is associated with a case where it is undesirable to select x_j to change its value. Consequently, regardless of the sign of $Eval_j(x)$, there is a motivation to favor a larger $EE_j(\mathbb{Q}, x)$ when choosing a variable x_j to change its value.

The two status conditions $S^{=}$ and $S^{\not=}$, where an $S^{=}$ status identifies a variable that should change the value it received in the most recent local optimum and an $S^{\not=}$ status identifies a variable that should retain its value that differs from its value received in the most recent local optimum, make it possible to further refine the partition of the neighborhood set $N(x)$ by considering the $N^{S^{=}}(x)$ to be the set of moves satisfying the three conditions of *status* $S^{=}$, and $N^{S^{\not=}}(x)$ to the set of moves satisfying the three conditions of *status* $S^{\not=}$. First, observe that the partition of the neighborhood set $N(x)$ defined by the two disjoint sets $N^{=}(x) = \{j \in N : x_j^{\#} = x_j^Q\}$ and $N^{\not=}(x) = \{j \in N : x_j^{\#} \neq x_j^Q\}$ can be constructed without having saved the value x_j^Q as a result of knowing the value $EE_j(\mathbb{Q}, x)$. In particular, we can construct the last local optimum from the value of the current solution $x_j^{\#}$ and the value $EE_j(\mathbb{Q}, x)$:

$$x_j^Q = \begin{cases} x_j & if\ EE_j(\mathbb{Q}, x) \geq 2^Q \\ 1 - x_j & if\ EE_j(\mathbb{Q}, x) \leq 2^Q - 1 \end{cases}$$

Hence, we have $N^{=}(x) = \{j \in N :\ EE_j(\mathbb{Q}, x) \geq 2^Q\}$ and $N^{\neq}(x) = \{j \in N : EE_j(\mathbb{Q}, x) \leq 2^Q - 1\}$. Consequently, the sets $N^{S^{=}}(x)$ and $N^{S^{\neq}}(x)$ may be defined as $N^{S^{=}}(x) = N^{=}(x) \cap N^{+}(x) \cap N^{\bar{T}}(x) \cap N^{E}(x)$ and $N^{S^{\neq}}(x) = N^{\neq}(x) \cap N^{-}(x) \cap N^{\bar{E}}(x)$.

Trigger an ascent phase based on status $S^{=}$ and $S^{\neq}$. We introduce variables $nS^{=}$ and $nS^{\neq}$ that count the number of variables x_j that have an $S^{=}$ or $S^{\neq}$ status. Temporarily, for convenience, we refer to the sum of these by *nS*. As illustrated in the next section, *nS* may decrease on some iterations, because a variable x_j with an $S^{=}$ or $S^{\neq}$ status may lose this status after a move involving a different variable is made. The following *trigger threshold* provides a rule for launching an ascent to a new local optimum

$$nS \geq Trigger \tag{6}$$

Once the trigger threshold is satisfied, we know that if we continue to hold any of the x_j variables with an $S^{=}$ or $S^{\neq}$ status at its current value then we cannot duplicate any of the r most recent local optima. Thus, we can select the last of these variables to remain tabu and, as intimated earlier, remove the tabu restrictions on all other variables and freely choose those with positive evaluations to ascend to a *conditional local optimum* – a solution that is locally optimal subject to retaining the Tabu restriction on the last variable. Upon reaching the conditional local optimum, the Tabu restriction on the remaining variable is likewise removed and the method proceeds to a true local optimum. (A natural variation is to allow all Tabu restrictions to be removed from the beginning of the ascent in the expectation that the Trigger threshold will create a high probability that the new local optimum reached will not duplicate any of the r most recent local optima. A contrasting variation would retain all variables with an $S^{=}$ and $S^{\neq}$ status tabu and release them all together from their Tabu restrictions at the conditional local optimum.)

Candidate list exploiting status $S^{=}$ and $S^{\neq}$. To differentiate moves related to the criteria used to determine an $S^{=}$ and $S^{\neq}$ status, the AA algorithm uses two subsets of (x) :

- $N^1(x) = N^{+}(x) \cap (N^{\bar{T}}(x) \cap N^{E}(x))$: $N^1(x) = \{j \in N(x) : Eval_j(x) > 0$ and either $TabuIter(j) < Iter$ or $EE_j(\mathbb{Q}, x) \geq Threshold_r(\mathbb{Q})\}$. This set is relevant in an Ascent Phase and in a Post-Ascent Phase where some variable has an S^{+} or $S^{=}$ status.
- $N^2(x) = (N - N^{+}(x)) \cap N^{\bar{T}}(x)$: $N^2(x) = \{Eval_j(x) \leq 0$ and $TabuIter(j) < Iter\}$. This set is relevant in a Post-Ascent Phase.

The AA algorithm explores a candidate list $CL \subseteq N(x)$ that depends on the introduced subsets, and the phase of research. The *CandidateList* function return the candidate list explored at each iteration.

```
Function CandidateList(x, Ascent){
  If N^A(x)≠∅ then CL = N^A(x)
  Else If N^1(x)≠∅ then
      If N^{S=}(x)≠∅ then CL = N^{S=}(x)
      Else CL = N^1(x)
    Else If Ascent = False then
          If N^2(x)≠∅ then CL = N^2(x)
          If N^{S=}(x) = ∅ then CL = N^{S≠}(x)
        EndIf
      EndIf
  EndIf
  Return CL
} // End CandidateList
```

The next section provides an extended example of how these relationships are exploited.

4. Extended illustration of exploiting strategies $S^=$ and $S^{\neq}$

We illustrate how the preceding data structures can be used to implement the two strategies $S^=$ and $S^{\neq}$ by an example of the steps following an Ascent Phase to move away from the most recent local optimum in a Post-Ascent Phase. We choose $Q = 4$ to identify the most recent local optimum x^Q. For clarity, the following Working Table shows all 4 of the most recent local optima x^1 to x^4, although it isn't necessary to keep a record of these solutions in order to execute the method. The Working Table also shows the weighted sums $EE_j^1(\mathbb{Q})$ and $EE_j^0(\mathbb{Q})$ whose values appear just beneath the solution $x^Q = x^4$ (the last solution shown) (Table 1).

As a prelude to discussing the moves shown in the Working Table, recall that Strategies $S^=$ and $S^{\neq}$ always use (a) $EE_j(\mathbb{Q}, x) = EE_j^1(\mathbb{Q})$ when the most recent solution x^Q has $x_j^Q = 1$ and (b) $EE_j(\mathbb{Q}, x) = EE_j^0(\mathbb{Q})$ when the most recent solution x^Q has $x_j^Q = 0$. The values shown in the row for "$EE_j(\mathbb{Q}, x)$" in the Working Table therefore refer to $EE_j^1(\mathbb{Q})$ when $x_j^Q = 1$ and refer to $EE_j^0(\mathbb{Q})$ when $x_j^Q = 0$ in the local optimum x^4. (This correspondence can be confirmed by computing $EE_j^1(\mathbb{Q})$ and $EE_j^0(\mathbb{Q})$ using (3) and (3.2).)

We have chosen a *Trigger* value of 3 for the trigger threshold $nS \geq Trigger$ in (6) that launches an ascent to a new local optimum. Details of the following Working Table are discussed immediately after the table.

Working table explanation. The method begins the Post-Ascent Phase with the most recent local optimum $x^4 = x^Q$ with the corresponding values for $EE_j(\mathbb{Q}, x)$ ($EE_j^1(\mathbb{Q})$ and $EE_j^0(\mathbb{Q})$). An asterisk (*) has been attached to each value $EE_j(\mathbb{Q}, x)$ value that satisfies $EE_j(\mathbb{Q}, x) \geq Threshold_r$, which is relevant for identifying moves with an $S^=$ or $S^{\neq}$ status that will cause nS to change. Here we have chosen $r = 3$, yielding $Threshold_3 = 8 + 4 + 2 = 14$ (the sum of the three largest w_q values). Hence an asterisk is attached to each $EE_j(\mathbb{Q}, x)$ value that is at least 14.

For this example, we do not bother to specify the choice rule used to select variables x_j to set equal to 0 or 1. A discussion of choice rules is given in Section 6. As a basis for tracking the choices made, recall that $Eval_j(x)$ is nonpositive for all variables at a local optimum. Hence the first choice of a variable x_j to change its value after reaching the local optimum x^4 will be for a variable with $Eval_j(x) \leq 0$. Each choice of such a variable will reverse the sign of $Eval_j(x)$ to produce $Eval_j(x) \geq 0$, as noted in condition (iv) of the $S^{\neq}$ strategy.

As previously noted, we assume that each variable selected to change its value is made Tabu to prevent a move that changes the variable back to its previous value. We also assume in the present example that the variables x_7 and x_9 are tabu in the local optimum x^4, as indicated by the superscript T attached to the values for these variables in the x^4 row. (Variables may receive a Tabu restriction in this way by a rule that, upon obtaining a local optimum, selects some number of the variables that were most recently assigned values leading to this local optimum to be Tabu. Here we may suppose x_7 and x_9 were the last Two variables to be assigned their current values to reach this local optimum. It would also be possible to apply a rule that does not make any variables in the local optimum Tabu. However, we include the situation with x_7 and x_9 tabu to increase the scope of the illustration.)

After one or more moves have been made, the evaluation for one of the previously selected variables x_j can change. This can be the basis for identifying a x_j that qualifies to receive an $S^{\neq}$ status because its evaluation has changed to become $Eval_j(x) \leq 0$. Similarly, the evaluation of a variable x_j that has not previously been selected can change from $Eval_j(x) \leq 0$ to $Eval_j(x) > 0$, qualifying x_j to receive an S^+ or an $S^=$ status.

Description of successive moves. As shown in the Working Table, Move 1 selects x_1 as the first variable to become x_k to change its value, changing $x_1 = 1$ to $x_1 = 0$, with its new value 0 shown in the row for Move 1.

Similarly, Move 2 chooses x_2 to change from 1 to 0, as indicated by the value 0 shown in the row for Move 2, and Move 3 chooses

Table 1
Working Table, [1]The new x^1, x^2, x^3 are the previous solutions x^2, x^3, x^4.

q	w_q	x^q	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	
1	1	x^1	1	1	1	1	0	0	0	1	1	1	
2	2	x^2	0	1	0	1	1	0	1	1	0	1	
3	4	x^3	0	1	0	0	0	0	1	1	1	0	
4	8	x^4	1	1	0	0	0	0	1^T	1	1^T	0	
$EE_j(\mathbb{Q}, x)$			9	15*	14*	12	13	15*	14*	15*	13	12	* for ≥ 14
Move			(Begin a Post-Ascent Phase)										nS
1			**0**										
2				**0**									
3					**1**								
4				$S^{\neq}$		**1**							1
5							**1**		$S^=$			S^+	2
6				**X**					**0**			**X**	1
7				$S^{\neq}$				**1**				S^+	2
8					$S^{\neq}$							**1**	3
Launch a new Ascent Phase													
Q	w(q)	x^q	(Obtain a new local optimum x^4)[1]										
4	8	x^4	1	0	1	0	0	1	1	1	1	1	
$EE_j(\mathbb{Q}, x)$			12	8	8	14*	14*	8	15*	15*	14*	9	(* for ≥ 14)

x_3 to change from 0 to 1, as indicated by the value 1 shown in the row for Move 3.

The next move, Move 4, chooses x_4 to change its current value, yielding $x_4 = 1$, indicated by the value 1 shown in the row for Move 4. In addition, the symbol $S^{\neq}$ for Strategy $S^{\neq}$ is inserted in this row in the column for x_2 to disclose that the result of setting $x_4 = 1$ has changed a current evaluation $Eval_2(x) > 0$ for x_2 to a new evaluation $Eval_2(x) < 0$, and in addition $EE_2^1(\mathbb{Q}) \geq 14$, as indicated by asterisk attached to the value 15 for $EE_2^1(\mathbb{Q})$ (where $EE_2(\mathbb{Q}, x) = EE_2^1(\mathbb{Q})$ is defined in relation to $x_2 = x_2^Q = 1$), shows that x_2 qualifies for the $S^{\neq}$ status of Strategy $S^{\neq}$. As a result, the nS in the far-right column of the table is incremented to 1 (from an implicit initial value of 0).

Move 5 selects x_5 to change from 0 to 1, and now this move causes x_7 and x_{10}, which have not yet changed their values in the local optimum x^4, to receive new evaluations $Eval_7(x)$a nd $Eval_{10}(x) > 0$, hence making them profitable and qualifying them for the S^{+} status. In addition, x_7 satisfies the recency threshold and hence qualifies for the $S^{=}$ status. Thus, we show $S^{=}$ in the column for x_7 and S^{+} in the column for x_{10}, and the $S^{=}$ status for x_7 results in incrementing nS to 2 in the far-right column.

The status $S^{=}$ and S^{+} for x_7 and x_{10} (which also identifies them as improving moves) give both variables priority to be selected to change their values. Since the $S^{=}$ status is higher than the S^{+} status, we select x_7 to change its value from 1 to 0 in Move 6. We make this move in spite of the fact that x_7 begins tabu (as indicated by the superscript T attached to its value in x^4), because the $S^{=}$ priority also overrules the tabu status.

Move 6 to set $x_7 = 0$ additionally has two other consequences in this example. The X's in the columns for x_2 and x_{10} are used to indicate that $S^{\neq}$ and S^{+} statuses of these variables have been canceled because of the move setting $x_7 = 0$ – a situation indicating that setting $x_7 = 0$ causes to become positive and $Eval_{10}(x)$ to become nonpositive. Because of cancelling the $S^{\neq}$ status of x_2, nS is reduced from 2 to 1.

There now remain three variables that are not Tabu, x_6, x_8 and x_{10} (unless the tabu tenure attached to x_9 is small enough that the tabu status of x_9 has expired). Move 7 selects x_6 to change its value from 0 to 1. According to the table, this move causes $Eval_2(x)$ and $Eval_{10}(x)$ once again to become nonpositive and positive, respectively, and consequently reinstates their $S^{\neq}$ and S^{+} status that was canceled on the previous move. (Such a rapid fluctuation of the nonpositive and positive evaluations of variables may be unlikely, but we show such a change to illustrate conditions that potentially may happen.) The recovery of the $S^{\neq}$ status by x_2 causes nS again to grow to 2.

Variable x_{10} with its S^{+} status now has priority above other variables to be chosen as x_k, and the assignment $x_{10} = 1$ occurs in Move 8. This move causes x_3 to receive an $S^{\neq}$ status (by changing $Eval_3(x) > 0$ back to $Eval_2(x) \leq 0$ and observing $EE_2^1(\mathbb{Q}) = 15$ which is larger enough for x_2 to satisfy the recency threshold). Now nS increases again, to equal 3.

Since we have chosen $Trigger$ to be 3, the trigger threshold $nS \geq Trigger$ is now satisfied and the ascent to a new local optimum is launched. The variable x_3 is held tabu until reaching a conditional local optimum, and then its tabu restriction is released as well to proceed to a true local optimum.

The next to last row of the Working Table identifies the new local optimum, again designated x^4 by keeping $Q = 4$. This shifts the indexing of the previous local optima so that the previous x^2, x^3 and x^4 now become x^1, x^2 and x^3. The new $EE_j^1(\mathbb{Q})$ and $EE_j^0(\mathbb{Q})$ values may be verified by consulting the new vectors that now qualify as x^1 through x^4. Alternatively, these values can be computed from the inductive formula $EE_j(\mathbb{Q}, x) = 2^{Q-1} + EE_j(\mathbb{Q}, x)/2$ as expressed in (5.2c) and (5.2d). (For example, in the case of x_1, which currently equals 1 and also equals 1 in the previous x^4 solution, the new value for $EE_j(\mathbb{Q}, x) = EE_1^1(\mathbb{Q}))$ is given by (5.2c) as $EE_j(\mathbb{Q}, x) = 2^{Q-1} + \frac{EE_j(\mathbb{Q},x)}{2} = 8 + \frac{9}{2} = 12.5$, and rounding down with integer arithmetic gives $= EE_1^1(\mathbb{Q}) = 12$. (There is no necessity to round down, of course, and there is some advantage for not doing so, particularly when α is chosen less than 2 as discussed in Appendix 1.) Similarly, in the case of x_2, which currently equals 0 but equals 1 in the previous x^4 solution, the new value for $EE_2(\mathbb{Q}, x)$ $(= EE_2^0(\mathbb{Q}))$ is given by (5.2d) as $EE_j(\mathbb{Q}, x) = 2^{Q-1} + (EEbase(\mathbb{Q}) - EE_j(\mathbb{Q}, x))/2 = 8 + \frac{15-15}{2} = 8.$)

This example brings up an additional characteristic of the method. The final Move 8 that gives x_3 an $S^{\neq}$ status affords the simplest way to launch an Ascent Phase. Specifically, a variable x_j with an $S^{\neq}$ status that becomes the "last variable" to satisfy the trigger threshold already has received an evaluation $Eval_j(x) \leq 0$ and is already tabu. Thus, no change is required in x_j or its tabu status to launch a new ascent.

However, if a last variable to satisfy the trigger threshold does so by receiving an $S^{=}$ status, then it would be necessary to make the move that gives x_j its new value. (This could have happened in the Working Table if Move 8 had caused x_8 to qualify for an $S^{=}$ status instead of causing x_3 to qualify for an $S^{\neq}$ status.) Then, after giving x_j its new value, its evaluation $Eval_j(x)$ will be negated to yield $Eval_j(x) < 0$ and x_j will be made tabu to launch the new ascent. This final move could cause nS to drop if it cancels the $S^{=}$ or $S^{\neq}$ status of some other variable(s), but there is a simple way to handle this. By keeping a separate value $nS^{=}$ for an $S^{=}$ status and $nS^{\neq}$ for an $S^{\neq}$ status, it is not necessary to keep track of cancellations. We only increase $nS^{=}$ by 1 when a variable x_j with an $S^{=}$ status is chosen to change its value (which locks in the value for x_j). Then, each time a variable changes its value we recompute $nS^{\neq}$ (starting over from $nS^{\neq} = 0$). Consequently, we identify the current $nS^{\neq}$ value at the same time as scanning the variables to select a new x_j to change its value.

However, there is an added subtlety. The properties that define an $S^{\neq}$ status, as previously noted, are the same as those exhibited by a variable that has a Post-$S^{=}$ status, i.e., a variable that previously had an $S^{=}$ status and then was chosen to change its value. Consequently, by counting the variables with an $S^{\neq}$ status, we also are counting the variables with a Post-$S^{=}$ status – or more precisely, variables with an *active* Post-$S^{=}$ status, that still satisfy the conditions when they changed their values. Thus, the trigger threshold inequality $nS^{=} \geq Trigger$ tallies just the variables that have Post-$S^{=}$ status, but not necessarily an active one (because the Eval value may have subsequently changed its sign). This provides a useful way to take advantage of both types of trigger threshold inequalities, as disclosed in the pseudocode subsequently provided for an advanced version of the algorithm illustrated in the Working Table. For a uniform notation, we refer to $Trigger^{=}$ and $Trigger^{\neq}$ to identify the trigger thresholds $nS^{=} \geq Trigger^{=}$ and $nS^{\neq} \geq Trigger^{\neq}$. In our present design we let $Trigger^{=} = Trigger^{\neq} = Trigger$, but other options are possible.

We call this algorithm, that alternates between an Ascent Phase and a Post-Ascent Phase, by exploiting the recency threshold and the trigger thresholds, an Alternating Ascent (AA) algorithm. Choice rules for selecting a variable x_j to become x_k in the AA algorithm are discussed in Section 5 followed by the pseudocode in Section 6 that provides an effective way to implement the algorithm.

5. Choice rules of exploiting $Eval_j(x)$ and $EE_j(\mathbb{Q}, x)$

Customary choice rules to select a variable x_j to become x_k and change its value from $x_k = x_k^{\#}$ to $x_k = 1 - x_k^{\#}$ can be extended in the context of the AA algorithm to take advantage of the val-

ues $EE_j(\mathbb{Q}, x)$ incorporated in the recency threshold. A good choice consists of selecting a variable x_k that optimizes simultaneously both $Eval_j(x)$ and $EE_j(\mathbb{Q}, x)$ over the selected candidate list CL. In a multi-objective optimization problem, the quality of a solution is determined by the dominance relation. This bi-objective selection problem has not just one "optimal" solution but several "efficient" solutions that satisfy tradeoffs between $EE_j(\mathbb{Q}, x)$ and $Eval_j(x)$. We identify three options for doing this, a simple weighting scheme, a simple cutoff (threshold) scheme and a more advanced cutoff procedure using an evaluation tradeoff analysis. The algorithm will only use one of these options, hence providing three different versions of the algorithm. These choice rule options depend on differentiating two conditions related to the criteria used to determine an $S^{=}$ and $S^{\neq}$ status identified by the candidate list function.

To provide a compact description of the move selection function, we introduce the following notation. Let f be a scalar vector with components f_j for $j \in L$. Then we denote

$$f_L^{Max} = Max\{f_j : j \in L\}$$
$$Max_L^f = Argmax\{f_j : j \in L\} = \{j \in L : f_j = f_L^{Max}\}.$$

First, we look for a move that maximizes the two objectives $Eval_j(x)$ and $EE_j(\mathbb{Q}, x)$ simultaneously. This move, if it exists, corresponds to an ideal decision. Identifying this move is accomplished by the following instruction

If $Max_{CL}^{Eval} \cap Max_{CL}^{EE} \neq \emptyset$ **then** select $k \in Max_{CL}^{Eval} \cap Max_{CL}^{EE}$

If an ideal move is not available, then the method uses one of the following three options.

Simple Weighted Sum Rule: The evaluation $Eval_j(x)$ is modified to take in account of the evaluation $EE_j(\mathbb{Q}, x)$ to produce the surrogate evaluation $S_j(\mathbb{Q}, x, w)$ given by

$$S_j(\mathbb{Q}, x, w) = Eval_j(x) + w \times \frac{EE_j(\mathbb{Q}, x)}{EEbase(\mathbb{Q})}$$

The pseudo code of this Simple Weighted Sum Rule is described bellow:

Function SimpleWeightedSum(***CL***, ***w***){
  If $Max_{CL}^{Eval} \cap Max_{CL}^{EE} \neq \emptyset$ **then** select $k \in Max_{CL}^{Eval} \cap Max_{CL}^{EE}$
  Else select $k \in Argmax\{S_j(\mathbb{Q}, x, w) : j \in CL\}$
  Return k
} // **End** ***SimpleWeightedSum***

The normalization of dividing $EE_j(\mathbb{Q}, x)$ by $EEbase(\mathbb{Q})$ gives $0 \leq EE_j(\mathbb{Q}, x)/EEbase(\mathbb{Q}) \leq 1$, which makes the calibration of w easier. Weight parameters w_1 for the candidate list $CL = N^1(x)$ and w_2 for the candidate list $CL = N^2(x)$ are differentiated. The weight value $w = w_1$ producing $S_j(\mathbb{Q}, x, w)$ for $N^1(x)$ will generally be small, as in performing a tie-breaking function. The value $w = w_2$ for $N^2(x)$ may also be small, but intuition suggests it may preferably be larger, perhaps in some instances large enough to cause $EE_j(\mathbb{Q}, x)$ to dominate $Eval_j(x)$. The possibilities for both w_1 and w_2 may range, for example, from 0.1 to $Eval_{CL}^{Max}$ the maximum expected value for $Eval_j(x)$.

Simple and advanced cutoff rules. The remaining choice rule options are given by the Simple Cutoff Rule and the Advanced Cutoff Rule, and are preceded by checking a secondary dominance condition. This is assured by the following instructions

$$CL_1^* = Argmax\{Eval_j(x) : EE_j(\mathbb{Q}, x) \geq EE_{CL}^{Max}, j \in CL\}$$

If $CL_1^* \neq \emptyset$ **then** select $k \in CL_1^*$

in the Simple Cutoff and the Advanced Cutoff function. When the secondary dominance condition is not satisfied (i.e. $CL_1^* = \emptyset$) the Simple and Advanced Cutoff Rules (which are applied separately in different versions of the algorithm) are as follows.

Simple Cutoff Rule: This rule select a move from the following candidate list CL_2^*, if it is not empty:

$$CL_2^* = Argmax\{Eval_j(x) : EE_j(\mathbb{Q}, x) \geq Cutoff, j \in CL\}$$

If $CL_2^* \neq \emptyset$ **then** select $k \in CL_2^*$

where $Cutoff = F \times EE_{CL}^{Max}$ and F is a fraction chosen between 0.5 and 0.9 (or more restrictively, between 0.7 and 0.9). In the special case where the $S^{=}$ status applies (i.e. $N^{S^{=}}(x) \neq \emptyset$), $Cutoff = \max(F \times EE_{CL}^{Max}, Threshold_r)$, the Simple Cutoff function is as follows

Function SimpleCutoff(***CL***, ***F***){
  $k = 0$
  If $Max_{CL}^{Eval} \cap Max_{CL}^{EE} \neq \emptyset$ **then** select $k \in Max_{CL}^{Eval} \cap Max_{CL}^{EE}$
  Else $CL_1^* = Argmax\{Eval_j(x) : EE_j(\mathbb{Q}, x) \geq EE_{CL}^{Max}, j \in CL\}$
    If $CL_1^* \neq \emptyset$ **then** select $k \in CL_1^*$
    Else $Cutoff = F \times EE_{CL}^{Max}$
      If $N^{S^{=}}(x) \neq \emptyset$ **then** $Cutoff = Max(Cutoff, Threshold_r)$
      $CL_2^* = Argmax\{Eval_j(x) : EE_j(\mathbb{Q}, x) \geq Cutoff, j \in CL\}$
      If $CL_2^* \neq \emptyset$ **then** select $k \in CL_2^*$
    EndIf
  EndIf
  Return k
} // **End** ***SimpleCutoff***

Advanced Cutoff Rule: The Advanced Cutoff Rule is based on the same $Cutoff$ value but uses a criterion to identify tradeoffs between $EE_j(\mathbb{Q}, x)$ and $Eval_j(x)$, expressed as

$CL_1^* = Argmax\{Eval_j(x) \times EE_j(\mathbb{Q}, x) : EE_j(\mathbb{Q}, x) \geq Cutoff, j \in CL\}$
If $CL_1^* \neq \emptyset$ **then** select $k \in CL_1^*$
in the case $N^1(x) \neq \emptyset$, and as
$CL_2^* = Argmax\{\frac{EE_j(\mathbb{Q},x)}{Eval_j(x)} : EE_j(\mathbb{Q}, x) \geq Cutoff, j \in CL\}$
If $CL_2^* \neq \emptyset$ **then** select $k \in CL_2^*$
when $N^1(x) = \emptyset$ and $Ascent = False$. The Advanced Cutoff function is

Function AdvancedCutoff(***CL***, ***F***, ***Ascent***){
  $k = 0$
  If $Max_{CL}^{Eval} \cap Max_{CL}^{EE} \neq \emptyset$ **then** select $k \in Max_{CL}^{Eval} \cap Max_{CL}^{EE}$
    Else $Cutoff = F \times EE_{CL}^{Max}$
      If $N^1(x) \neq \emptyset$ **then**
    If $N^{S^{=}}(x) \neq \emptyset$ **then** $Cutoff = Max(Cutoff, Threshold_r)$
      $CL_1^* = Argmax\{Eval_j(x) \times EE_j(\mathbb{Q}, x) : EE_j(\mathbb{Q}, x) \geq Cutoff, j \in CL\}$
    If $CL_1^* \neq \emptyset$ **then** select $k \in CL_1^*$
    Else If $Ascent = False$ **then**
      $CL_2^* = Argmax\{\frac{EE_j(\mathbb{Q},x)}{Eval_j(x)} : EE_j(\mathbb{Q}, x) \geq Cutoff, j \in CL\}$
      If $CL_2^* \neq \emptyset$ **then** select $k \in CL_2^*$
      EndIf
      EndIf
  EndIf
  Return k
} // **End** ***AdvancedCutoff***

The analysis underlying the tradeoff choices in the advanced cutoff rule are explained in Appendix 2. For the move selection using the cutoff thresholds, there may be merit in choosing the fraction F larger (for example, closer to 0.5) when $N^1(x) \neq \emptyset$, and perhaps larger still when $N^1(x) \neq \emptyset$, because in these cases a somewhat smaller range of x_j variables are candidates to be selected for x_k. For example, setting $F = 0.7$ when restricting attention to variables with $Eval_j(x) > 0$, and setting $F = 0.5$ when additionally restricting attention to variables satisfying the recency threshold, may roughly correspond to setting $F = 0.9$ when considering all variables without restriction.

As previously noted, each of these choice rule options gives rise to a different version of the AA algorithm. Hence, given the parameters: candidate list CL, weight w, fraction F, state of AA $Ascent$ and choice rule option $Choice$, the following $SelectMove$ function returns the selected move $k \in CL$ if it exists and returns 0 otherwise.

```
Function SelectMove(CL, w, F, Ascent, Choice){
  Switch (Choice){
    Case 1: k = SimpleWeightedSum(CL, w)
    Case 2: k = SimpleCutoff(CL, F)
    Case 3: k = AdvancedCutoff(CL, F, Ascent)
  }
  Return k
} // End SelectMove
```

6. General AA algorithm design and pseudocode

An AA algorithm oscillates between the Ascent Phase and the Post-Ascent Phase which is controlled by the boolean variable $Ascent = True$ if the state of the AA algorithm is in Ascent Phase and $Ascent = False$ when the AA algorithm is in Post-Ascent Phase. The pseudocode that follows is organized to facilitate experimentation with the ideas for exploiting local optimality described in the preceding sections.

The AA algorithm starts with an initialization phase where all global variables of AA are determined once the parameters are fixed. At each current iteration, a $CurrentIter()$ procedure is called to choose a next move k to be selected depending on whether the search is in the Ascent Phase or the Post-Ascent Phase (i.e. $Ascent = True$ *or* $False$). After this current phase, a post update procedure is launched to update the state of search. The pseudo code of the main AA algorithm may be stated as follows:

```
Algorithm AA(){
  Inititalization()
  For Iter = 1 to IterMax do // or until the expiration of a time limit
    k = CurrentIter()
    If k > 0 then PostIterUpdateMove(k) // move k is selected
    Else PostIterUpdateNoMove() // no move exists, i.e. k = 0
  EndFor
} // End AA
```

The instructions for the three key components of the AA algorithm – the Initialization, the Current Iteration Routine and the Post Iteration Update – are as follows. To recapitulate, we assume that we start with an initial solution $x = 0$, $Ascent = True$ and we do not keep the best solution found but just its value denoted x_0^*. The AA algorithm can be easily adjusted to keep also the best found solution x^* during the search. In the algorithms presented below, x denotes the current solution and we abbreviate the notation of $Eval_j(x)$, $EE_j(\mathbb{Q}, x)$, $EEbase(\mathbb{Q})$, $Threshold_r(\mathbb{Q})$ by referring to $Eval_j$, EE_j, $EEbase$, and $Threshold_r$. Starting with null solution $x = 0$, this simplifies the initialization of $Eval_j$, EE_j, $EEbase$, x_0, and x_0^* as described in (Section 2.2 and Section 3.3), see $Inititalization(Q, r, Ascent)$ algorithm.

The Tabu restrictions required during the Ascent Phase and the Post-Ascent Phase of the AA algorithm take a simple form where the tenure is given by setting $Tenure = Large$, where $Large$ represents a large positive number. This approach is made possible by the fact that Tabu restrictions will be overruled by the aspiration criterion and by an $S^=$ status, which, together with the trigger threshold, implicitly determine the duration of a tabu restriction. The AA algorithm starts with no variables tabu (i.e. $TabuIter(j) = 0$ for $j = 1$ to n) and initializes the record of the 3 most recent variables x_j assigned values in the Ascent Phase. This can be done for any number of recent variables assigned values in the Ascent Phase (i.e. $R_1 = R_2 = R_3 = 0$). This memory refresh is also done during the search except that a $Last$ move is identified to avoid cycling as in Tabu Search. For this reason, we introduce an important convention which introduces a term $TabuIter(0)$, which is assigned a value as $TabuIter(Last)$ when $Last = 0$. Likewise, we introduce a term $Eval_0$ which is permanently assigned the value $Eval_0 = 0$. With these conventions, the AA algorithm calls the following $ResetMemory(Last)$ with $Last = 0$ in the initialization phase:

```
Procedure ResetMemory(Last){
  For j = 1 to n do TabuIter(j) = 0
  TabuIter(Last) = Iter + Tenure
  R1 = R2 = R3 = 0
} // End ResetMemory
```

These dummy values $TabuIter(0)$ and $Eval_0$ save computational time for checking valid cases. Our AA algorithm oscillates between the Ascent Phase and the Post-Ascent Phase which is controlled by the boolean variable $Ascent$. $Ascent = True$ if the AA algorithm is in Ascent Phase and $Ascent = False$ when AA algorithm is in Post-Ascent Phase.

The initialization phase of the AA algorithm starts with an initial solution $x^\#$ with objective value $x_0^\#$, and sets $x = x^\#$, $x_0^* = x_0^\#$ where x_0^* denotes the best known value, and initializes the record of the 3 most recent variables x_j assigned values in the Ascent Phase. This can be done for any number of recent variables assigned values in the Ascent Phase ($R_1 = R_2 = R_3 = 0$). Consequently, the initialization phase of the AA algorithm is described as follows:

```
Procedure Initialization(){
  Set x0* = x0#, x = x#
  Eval0 = 0, For j = 1 to n do Compute Evalj for the initial solution with xj = xj#
  Tenure = Large; ReducedTenure = min(16, n/12)
  R1 = R2 = R3 = 0, For j = 1 to n do TabuIter(j) = 0
  Last = Last_{S=} = 0
  EEbase = 0, For j = 1 to n do EEj = 0
  Threshold_r = 2^{Q-r}(2^r - 1)
  nS= = 0
  ResetMemory(Last)
  Ascent = True
} // End Initialization
```

Each iteration of the AA algorithm begins by checking the aspiration criterion for overriding a Tabu restriction to see whether changing the value of x_j will yield a value for x_0 (currently given by $x_0 = x_0^\#$) that improves upon the best value x_0^*, as indicated by $x_0^\# + Eval_j(x) > x_0^*$. Hence, when $N^A(x) \neq \emptyset$, the method selects a best move $k \in Argmax\{Eval_j(x) : j \in N^A(x)\}$.

```
Function CurrentIter(){
  If N^A(x) ≠ ∅ then select k ∈ Argmax{Evalj : j ∈ N^A(x)}; Return k ;
  If N^{S=}(x) ≠ ∅ then k = SelectMove(N^{S=}(x), w1, F, Ascent, Choice1); Return k ;
  If N^1(x) ≠ ∅ then k = SelectMove(N^1(x), w1, F, Ascent, Choice1); Return k ;
  If Ascent = False then
    If N^2(x) ≠ ∅ then k = SelectMove(N^2(x), w2, 1/F, Ascent, Choice2); Return k ;
    If N^{S≠}(x) ≠ ∅ then select Last_{S≠} ∈ N^{S≠}(x)
  EndIf
  Return 0
} // End CurrentIter
```

The Post Iteration Update $PostIterUpdateMove(k)$ procedure has as argument the index k of the chosen variable x_k to change its value. When $N^A(x) = \emptyset$, $nS^{\neq}$ includes the count for active Post-$S^=$ status. Whenever $nS^= \geq Trigger$ (= Trigger in the current design), launch a new ascent even if $N^1(x) \neq \emptyset$, because even if $Eval_j > 0$ is encountered, possibly the influence of the previous x_k assignment could create $nS^{\neq} \geq Trigger$ (by increasing $nS^{\neq}$ in the Current Iteration Routine). The only exception is if $N^{S^=}(x) \neq \emptyset$, since then we first update x_k before considering the possibility of launching a new ascent. Hence, next checks for $N^{S^=}(x) = \emptyset$ as a basis for checking if $nS^{\neq} \geq Trigger$ will launch an Ascent before too many improving choices are made. Don't launch new ascent if $N^A(x) \neq \emptyset$ until after updating x_k. It would be possible to drop "$N^{S^=}(x) = \emptyset$" next, because this will be checked in the Post Iteration Update routine. Moreover, if $N^A(x) = N^{S^=}(x) = \emptyset$ and $nS^{\neq} \geq Trigger$ then $Ascent = False$ is implicit here because $nS^{\neq}$ only changes in the Current Iteration Routine to become greater than 0 when $Ascent = False$. In addition, no variable should be assigned a value, just as if $k = 0$. In this case, we free all variables from Tabu restric-

tions, except for $Last = Last_{S^{\neq}}$ by calling $ResetMemory(Last)$ and setting $Ascent = True$. These updates are done before updating for the choice of x_k, because the x_k choice may be different when a new ascent is launched as here. Subsequently, launch a new ascent after updating x_k if $nS^{\neq} \geq Trigger$.

The $PostIterUpdateMove(k)$ procedure calls the $UpdateMove(k, Ascent)$ procedure after the flip move k, which updates the value of the current solution x_0, $Eval_j$ for $j = 1$ to n, x_k, EE_k and also updates the memory when the AA algorithm is in the Ascent Phase (see Section 2.2 and Section 3.3):

```
Procedure UpdateMove(k){
  x_0 = x_0 + Eval_k
  For j = 1 to n do Update Eval_j
  x_k = 1 − x_k
  EE_k = EEbase − EE_k
  If Ascent = True then R_3 = R_2; R_2 = R_1; R_1 = k
} // End UpdateMove
```

When $Ascent = False$ the search is in the Post-Ascent Phase and the condition $nS^{\neq} \geq Trigger$ allows an ascent to be launched when $N^A(x) \neq \emptyset$. The condition $Eval_{Last_{S^{=}}} < 0$ could be needed because $Last_{S^{=}}$ might have been recorded on a previous iteration. However, only accept $Last_{S^{=}}$ as $Last$ if $Eval_{Last_{S^{=}}} < 0$, as it would be if its Post-$S^{=}$ status still applies; and check for $Last_{S^{\neq}}$ on same condition it would be preferable to hold Tabu unless it is now profitable to change back. Since we are only keeping variables Tabu that are unprofitable anyway, it seems we don't really need to hold anything Tabu during an Ascent phase, and this extra fuss is wasted effort.

```
Procedure PostIterUpdateMove(k){ // k > 0
  If N^A(x) = N^{S^-}(x) = ∅ and nS^= ≥ Trigger then
    // PA Completed: Launch a new Ascent Phase
    ResetMemory(Last_{S^-}); Ascent = True
    Return; // Exit the Post Iteration Update
  EndIf
  UpdateMove(k)
  If Ascent = False then
      Tabulter(k) = Iter + Tenure
    If N_{S^≠} ≠ ∅ then Last_{S^≠} = k
    If N_A ≠ ∅ or N_{S^≠} ≠ ∅ then nS^≠ = nS^≠ + 1
    If nS^≠ ≥ Trigger then // PA Completed: Launch a new Ascent Phase
      If Eval_{Last_{S^≠}} < 0 then Last = Last_{S^≠}
        ResetMemory(Last) ; Ascent = True
    EndIf
EndIf
} // End PostIterUpdateMove
```

In the $PostIterUpdateNoMove()$ routine, no variable x_k could be chosen to change its value, hence must end an Ascent Phase if $Ascent = True$ or must begin an Ascent Phase if $Ascent = False$. The outcome $k = 0$ is the only way to end an Ascent Phase. $Eval_j$ does not need to be updated. Each time a true local optimum is obtained, the $PostIterUpdateNoMove$ procedure calls the $UpdateEE()$ procedure which updates the value of EE_j for $j = 1$ to n, $EEbase$ and $Threshold_r$ (see Section 2.2 and Section 3.3):

```
Procedure UpdateEE(){
  For j = 1 to n do EE_j = 2^{Q−1} + 2^{−1} EE_j
  EEbase = 2^{Q−1} + 2^{−1} EEbase
  Threshold_r = min(EEbase, 2^{Q−r}(2^r − 1))
} // End UpdateEE
```

The end of Ascent Phase is in two steps: first, free $Last$ from its tabu restriction to complete the ascent to a local optimum, and then at the local optimum perform updates for the Post-Ascent Phase when $Ascent = False$.

```
Procedure PostIterUpdateNoMove(){ // k = 0
  If Ascent = True then // End Ascent Phase
    Tabulter(Last) = 0
    If Eval_{Last} > 0 then // A conditional local optimum is reached
      UpdateMove(Last); Last = 0
    Else // A true local optimum is obtained, and a PA Phase begins
      Ascent = False; nS^≠ = 0
      Last = Last_{S^≠} = Last_{S^=} = 0
      UpdateEE()
      Tabulter(j) = Iter + ReducedTenure for j ∈ {R_1, R_2, R_3}
    EndIf
  Else // Launch a new Ascent Phase
    If nS^≠ > 0 and Eval_{Last_{S^≠}} < 0 then Last = Last_{S^≠}
    Else If nS^= > 0 and Eval_{Last_{S^=}} < 0 then Last = Last_{S^=}
     ResetMemory(Last) ; Ascent = True
  EndIf
} // End PostIterUpdateNoMove
```

7. Computational results of the AA algorithm on QUBO

This section presents computational results of applying the AA algorithm to the quadratic unconstrained binary optimization (QUBO) problem. We start by providing an efficient 1-flip move evaluation and describing input and output parameters of the AA algorithm.

7.1. QUBO problem and its 1-flip move evaluation

The quadratic unconstrained binary optimization (QUBO) problem is an NP-hard combinatorial optimization problem introduced by Hammer and Rudeanu (1968), which can be expressed as follows:

$$(QUBO)\ \begin{cases} maximize & x_0 = xAx \\ s.t. & x \in \{0,1\}^n \end{cases}$$

where $A = (a_{ij})$ is a symmetric matrix of dimension $n \times n$ where component a_{ij} are real values for $i, j \in N = \{1, \ldots, n\}$. The evaluation $Eval_j(x)$ for flipping variable x_j of x that identifies the change in the objective function when x_j changes its value, i.e.

$$Eval_j(x) = x'_0 - x_0 = x'Ax' - xAx$$

The move evaluation $Eval_j(x)$ can alternatively be expressed as

$$Eval_j(x) = \left(1 - 2x_j\right)\left(A^j + A_j\right)x + a_{jj}$$

The last equation is obtained since $(1 - 2x_j)^2 = 1$ and $a_{jj} = e^j A e^j$ where A^j and A_j refer to column and row j of matrix A respectively. If the input matrix A is a symmetric matrix the term $A^j + A_j$ is equal to $2A^j = 2A_j$, hence

$$Eval_j(x) = 2\left(1 - 2x_j\right)A_j x + a_{jj}.$$

Proposition 2. *Let x' be the solution obtained from x by flipping the variable x_k,* i.e. *$x' = x + (1 - 2x_k)e^k$. Then the update of the evaluation $Eval_j(x')$ can be computed using the rule*

$$Eval_j\left(x'\right) = \begin{cases} -Eval_j(x) & if\ j = k \\ Eval_j(x) + a'_{jk} & if\ j \neq k\ and\ x_j = x_k \\ Eval_j(x) - a'_{jk} & if\ j \neq k\ and\ x_j \neq x_k \end{cases}$$

where

$$a'_{jk} = \begin{cases} a_{jk} & if\ j < k \\ a_{jj} & if\ j = k \\ a_{kj} & if\ j > k \end{cases}$$

Note that if the starting solution is null, i.e. $x = 0$, the initial evaluation can be computed simply as follows $Eval_j(0) = a'_{jj}$.

Table 2
Options for choice rules.

	$Choice_1$	$Choice_2$
1	Weighted_Sum_Rule (w_1)	Weighted_Sum_Rule (w_2)
2	Simple_Cutoff_Rule (F)	Simple_Cutoff_Rule ($1/F$)
3	Advanced_Cutoff_Rule (F)	Advanced_Cutoff_Rule ($1/F$)

This efficient means of evaluating 1-flip moves is an improved version of the procedure proposed by Glover and Hao (2010), which is used in a variety of different algorithms for QUBO (e.g. Glover, Lü, & Hao (2010), Hanafi, Rebai & Vasquez (2013)).

Experiments are carried out on two sets of benchmark instances, where the first set is composed of 60 instances from OR-Library with a density of 0.1 and the second set is composed of 21 instances from Palubeckis with densities from 0.5 to 1.0. According to the instance size, we further divide the instances into four subsets. The '*small*' set is composed of 20 instances with 50 to 100 variables. The '*medium*' set is composed of 20 instances with 250 to 500 variables. The 'midsize' set is composed of 20 instances with 1000 to 2500 variables. The '*large*' set is composed of 21 instances with 3000 to 7000 variables. The OR-Library instances are available on the website http://people.brunel.ac.uk/~mastjjb/jeb/orlib/bqpinfo.html and the Palubeckis instances are available upon request since the previous website is not available. An optimal value for small instances is obtained by solving the standard linearization (see Glover & Woolsey (1974), Billionnet & Calmels (1996)) where each quadratic term in the objective function, $x_i x_j$, is replaced by a new binary variable, y_{ij}, and adding new constraints $y_{ij} \leq x_i$, $y_{ij} \leq x_j$, and $x_i + x_j \leq 1 + y_{ij}$ to require that $y_{ij} = 1$ if and only if $x_i = x_j = 1$. Using Cplex software with 1 a time limit of hour, optimal values are known for $n \leq 250$, except for $QUBO_{250,6}$ and $QUBO_{250,8}$. Note that the best known values x_0^{**} are available for those instances.

7.2. Input and output parameters of the AA algorithm

The input parameters of the AA algorithm are:

- An initial solution $x^{\#}$ with objective value $x_0^{\#}$. In our experiments, the starting solution is $x^{\#} = 0$.
- Number of recent local optima: Q (the maximum value of Q depends on the largest integer or real value that can be supported by the computer and/or software used). For simplicity, we choose $Q = 30$.
- Number of most recent local optima: $r < Q$, for $Threshold_r$ (e.g., $r = 8$ to 20).
- Trigger thresholds: *Trigger* (e.g. from 5 to 9).
- Base of Exponential Extrapolation $\alpha > 0$. For simplicity, we choose $\alpha = 2$.
- Weights for the Sum Rule: w_1 and w_2 for moves in $N^1(x)$ and $N^2(x)$: range from 0.1 to 100.
- Fraction of EE_{CL}^{Max} used for cutoff: F e.g., from 0.7 to 0.9.
- Memories: $Tenure = Large$; $ReducedTenure = \min(16, n/12)$.
- Options $Choice_1$, $Choice_2 \in \{1, 2, 3\}$ for $N^1(x)$ and $N^2(x)$, respectively, as identified in Table 2 following.

The AA algorithm also uses a common stopping criterion shared with many other heuristics: the maximum number of iterations *MaxIter* which can be represented as a multiple of dimension n (i.e. $MaxIter \in \{50n, \ldots, 200n\}$) or the time limit (i.e. a multiple of the time of uploading an instance of QUBO). For simplicity, we use $C1$ and $C2$ to denote $Choice_1$ and $Choice_2$, respectively, and *Cab* to denote the combined choice $C1 = a$ and $C2 = b$. We use the IRACE automatic tuning tool to determine the best parameter settings for the AA algorithm. Table 3 shows the specified range of the parameters "w_1, w_2, F, Q, r, $C1$, $C2$, *Trigger*" for the IRACE experiments,

Table 3
Parameters types and ranges for the IRACE experiments.

Name	Type	Range
w_1	r	(0.1, 1)
w_2	r	(0.1, 100)
F	r	(0.7, 0.9)
Q	i	(15, 30)
r	i	(10, 20)
$C1$	c	(1, 2, 3)
$C2$	c	(1, 2, 3)
Trigger	i	(5, 9)

where type "r", "i" and "c" denote real number, integer and category respectively. For the types "r" and "i", a pair of numbers represents the minimum and maximum values of the parameter settings, where we set the precision to 0.1. The maximum experiment budget is set to 10,000.

During the IRACE experiments, we observe that when all 81 instances are included to determine the best parameter settings, the overall computational results are not as good as when the instances are divided into two categories, one consisting of 50 instances with the number of variables ranging from 50 to 1000 and the other consisting of 31 instances with no less than 2500 variables. Hence, we report the two best parameter settings recommended from IRACE for each category, which respectively for the first 50 instances and the last 31 instances are: $w_1 = 0.7$, $w_2 = 1$, $F = 0.9$, $Q = 24$, $r = 12$, $C1 = 2$, $C2 = 1$, $Trigger = 5$ and $w_1 = 0.4$, $w_2 = 0.6$, $F = 0.9$, $Q = 17$, $r = 11$, $C1 = 2$, $C2 = 1$, $Trigger = 8$. By reference to these two settings, we observe that $C1$ always receives the value 2 and $C2$ always receives the value 1, suggesting that the preferred options for the choice rules are the Simple_Cutoff_Rule and the Weighted_Sum_Rule. As the number of variables is increased, we note that the value of *Trigger* should be larger.

7.3. Computational experimentation

In this section we assess the behavior of the AA algorithm on the 61 instances from the OR-Library and the 21 Palubeckis instances. Algorithms described in this paper were implemented in C++ and compiled using GNU GCC 10.2.0 with -O3 flag on a Linux 3.10.0–862.el7.x86_64 operating system. The computer used for the experiments is equipped with a Intel(R) Xeon(R) Gold 6226R (2.90 GHz) processor. All CPU times reported in seconds were obtained using the clock function and the CLOCKS_PER_SEC macro.

Relevant options of $Choice_1$, $Choice_2 \in \{1, 2, 3\}$. In total we have 9 options $(C1, C2) \in \{1, 2, 3\}^2$ where choice $C1$ is used for candidate list N^1 and choice $C2$ for N^2. The computational results show that only 4 options are relevant where $(C1, C2) \in \{(1, 1), (1, 2), (2, 1), (3, 1)\}$ on the 81 tested instances. For each class C of instances *Small, Medium, Midsize, Large,* we report in Table 3 the computational results obtained by fixing $MaxIter = 50 \times n$. The quality of the performance of each execution of the AA algorithm on a given instance I is computed as $Gap_I = \frac{x_0^{**} - x_0^*}{x_0^{**}}$, where x_0^{**} denotes the best known values reported in the literature and x_0^* denotes the best value returned by the AA algorithm. We provide the average values $Gap\% = 10^2 \times Gap_{avg}$ where $Gap_{avg} = \frac{\sum_{I \in C} Gap_I}{|C|}$ and the number #*Best* of instances I where $Gap_I = 0$ (i.e. the AA algorithm reaches the best known value). The column *Best* indicates the best result over the 4 options of the AA algorithm.

From Table 4, we observe that the options $C11$, $C21$ and $C31$ perform similarly well by obtaining percentage *gaps* to the best known values of 0.0190%, 0.0147% and 0.0152% and matching these values for 48, 51 and 53 instances, respectively, collectively reach-

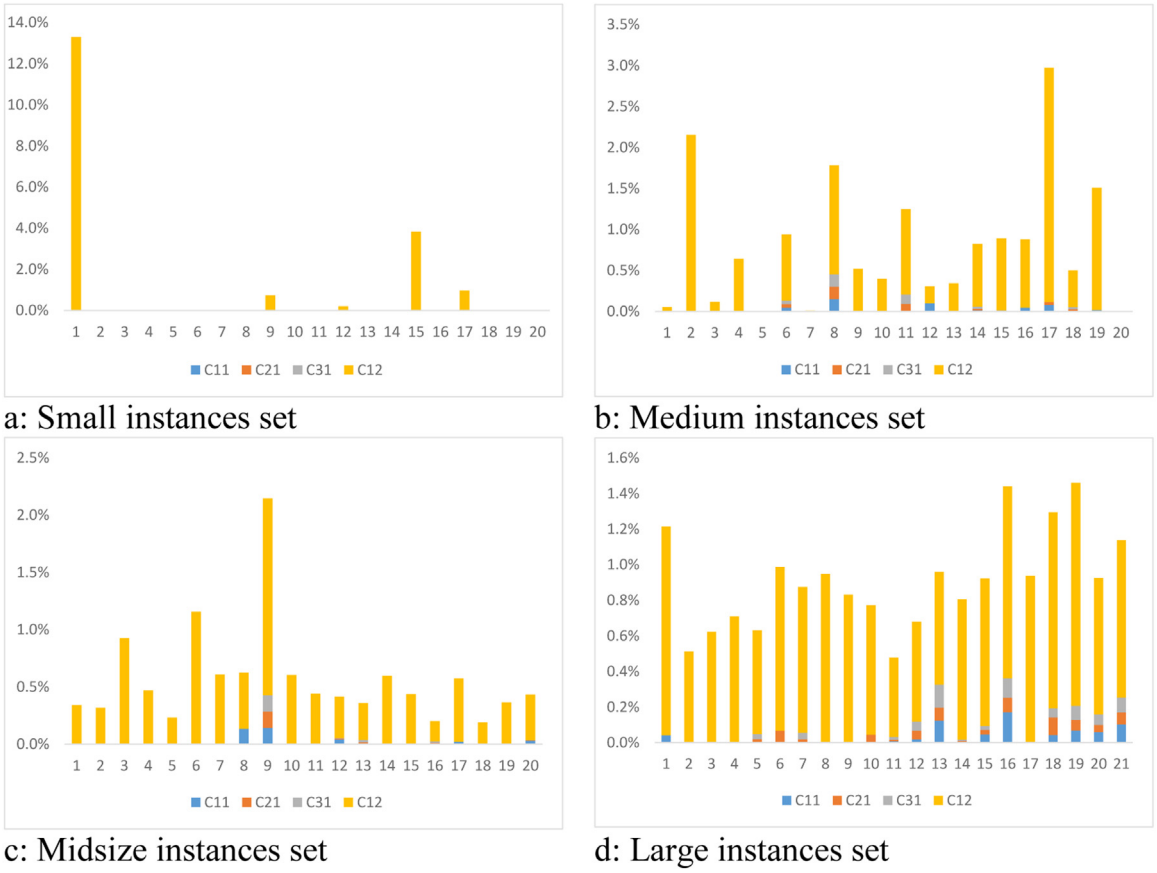

Fig. 1. Comparison of 4 options of AA algorithm on all instances sets.

Table 4
Comparison of 4 options of AA algorithm.

		*C*11	*C*12	*C*21	*C*31	*Best*
Small	*Gap%*	**0**	0.9519	**0**	**0**	0
20	*#Best*	**20**	15	**20**	**20**	20
Medium	*Gap%*	0.0231	0.7453	**0.0177**	0.0185	0.0108
20	*#Best*	11	1	**14**	12	15
Midsize	*Gap%*	0.0194	0.5347	**0.0087**	0.0092	0.0077
20	*#Best*	9	0	**12**	**12**	13
Large	*Gap%*	0.0328	0.8155	**0.0315**	0.0322	0.0214
21	*#Best*	8	0	5	**9**	11
All	*Gap%*	0.0190	0.7625	**0.0147**	0.0152	0.0101
81	*#Best*	48	16	51	**53**	**59**

ing 59 best known values out of 81 instances. The option *C*12 performs significantly worse than the other 3 options. The option *C*21 performs better than the other 3 options in terms of *Gap* and *#Best* for medium and midsize instances of the QUBO problem, and for the small instances, *C*21 performs as well as *C*11 and *C*31 by finding all the best known values. For the large instances, *C*31 performs better by obtaining the best known values but *C*21 obtains the best average gap. The overall observation that setting *C*2 to 1 always leads to better results indicates that the AA algorithm is significantly affected by *C*2 and relatively insensitive to *C*1.

Furthermore, we show in Fig. 1 the comparison results of the AA algorithm under each option when solving each instance from the small, medium, midsize and large instances. The x-axis represents each instance and the y-axis denotes the percentage gap to the best known value. If the percentage gap of a choice option is 0, the y-axis is not displayed. From Fig. 1, we observe that *C*12 fails to reach the best known values for 5 small instances, while the other options (*C*11, *C*12, *C*31) can reach these values for all 20 small instances. For solving each instance from the other sets of instances, the percentage gap of *C*12 is much larger than that of *C*11, *C*21 and *C*31. The choice *C*12 only finds the best known value for the instance 5 (QUBO250.5) while the options *C*11, *C*21 and *C*31 obtain the best known values for most instances.

AA algorithm behavior: time and iteration. Table 5 presents more information about the AA algorithm behavior. Column *Time* gives the average CPU time needed to execute the $Max_{Iter} = 50n$ iterations over each set of instances under each choice option of the AA algorithm. Column T^* corresponds to the average CPU time to reach the best solution found by a run on the AA algorithm over each set of instances. Column%I corresponds to $10^2 \times \frac{Iter^*}{Max_{Iter}}$, where $Iter^*$ denotes the iteration that produced the best value. Looking at the option *C*21, for instance, the small (resp. medium, midsize and large) class requires on average less than 0.01 (resp. 0.12, 2.71 and 16.69) seconds for each run. The best solutions are found for small (resp. medium, midsize and large) instances in less than 0.01 (resp. 0.06, 1.25 and 13.05) seconds.

The CPU time increases as the dimension n of the QUBO problem increases. However, there is no general behavior regarding the iteration or time when the best value is reached. As the value%$I = 100 \times \frac{Iter^*}{Max_{Iter}}$ gets closer to 100%, the more frequently the best solution is found in the later iterations. For example, *C*11 reaches the best objective values at%$I = 58\%$ for small instances and%$I = 24\%$ for medium instances. By comparison, *C*12 reaches the best objec-

Table 5
AA Algorithm Behavior: CPU time in seconds and iterations.

Instances	C11			C12			C21			C31		
	Time	T^*	%*I*	*Time*	T^*	%*I*	*Time*	T^*	%*I*	*Time*	T^*	%*I*
Small	0	0	58%	0	0	80%	0	0	53%	0	0	53%
Medium	0.12	0.03	24%	0.08	0.08	95%	0.12	0.06	49%	0.12	0.03	33%
Midsize	2.70	1.17	51%	2.03	1.30	71%	2.71	1.25	53%	2.71	1.31	60%
Large	16.68	11.34	71%	13.13	11.56	86%	16.69	13.05	76%	16.67	13.99	79%

tive values later at%$I = 80\%$ for small instances and%$I = 95\%$ for medium instances.

8. Concluding observations and future steps

The departure from the classical approaches for responding to local optimality in the strategies of the AA algorithm open a variety of possibilities for exploration. Exponential Extrapolation EE_j compresses Q recent local optima into a single vector. Moreover, the recency threshold $EE_j \geq Threshold_r$ is new aspiration criterion that prevents duplication from occurring among the r most recent local optima. The organization of the pseudocode is designed to make these possibilities visible and easy to pursue. Questions that invite investigation concern the determination of preferred threshold parameters and the choice of values other than 2 for the exponential extrapolation parameter α (particularly in the "mixed α strategy" discussed in Appendix 1). Relevant questions include:

- What are the tradeoffs between r and *Trigger* of the recency and trigger thresholds?
- Does an α value less than 2 become more effective as r or *Trigger* becomes larger?
- How can the ability to start the AA algorithm with any solution $x^{\#}$ be exploited most effectively in a diversification strategy?
- Do answers depend on the state of the search, e.g., on how many iterations have elapsed or on how many Ascent and Post-Ascent phases have been performed?
- Are there advantages to joining path relinking with the AA algorithm?

Exploring variants of the AA algorithm that are tailored for different classes of problems likewise presents an appealing avenue for future research. The computational results for applying this first version of the AA algorithm to quadratic unconstrained binary optimization (QUBO) problems with up to 7000 variables demonstrate its effectiveness in terms of both solution quality and computational effort. More advanced AA algorithms, including a Double-Pass AA Algorithm and an AA Algorithm with dynamic diversification strategies, will be examined in a sequel.

Acknowledgment

We thank the referees for the suggestive comments which helped us to improve the paper quality. This work was partially supported by the National Natural Science Foundation of China (No. 71971172) and the Fundamental Research Funds for the Central Universities (No. D5000210834).

Appendix 1. Implications of the recency threshold for using different α values

We begin by reviewing the meaning of the more general form of recency threshold $EE_j(\mathbb{Q}, x) \geq Threshold_r(\mathbb{Q})$ of (4) when α is not restricted to $\alpha = 2$. Rewriting (3) with α replacing 2 gives

$$EE_j(\mathbb{Q}, x, \alpha) = \sum_{q=1}^{Q} \alpha^{q-1}\left(1 - \left|x_j - x_j^q\right|\right). \tag{3a}$$

Then the corresponding form the recency threshold of (4) for $r \in \{1, \ldots, Q\}$ becomes

$$Threshold_r(\mathbb{Q}, \alpha) = \sum_{q=1}^{r} \alpha^{Q-q} = \begin{cases} \frac{\alpha^Q - \alpha^{Q-r}}{\alpha - 1} & if\ \alpha \neq 1 \\ r & if\ \alpha = 1 \end{cases} \tag{3.1a}$$

We call the binary vector $V_j(\mathbb{Q}, r, x, \alpha) = (x_j^Q, x_j^{Q-1}, \ldots, x_j^1)$ *acceptable* if it satisfies the recency threshold

$$EE_j(\mathbb{Q}, x, \alpha) \geq Threshold_r(\mathbb{Q}, \alpha). \tag{4a}$$

First observe that $EE_j(\mathbb{Q}, x, \alpha)$ can be expressed as follows

$$EE_j(\mathbb{Q}, x, \alpha) = Threshold_Q(\mathbb{Q}, \alpha) - \sum_{q=1}^{Q} \alpha^{q-1}\left|x_j - x_j^q\right|. \tag{3′a}$$

Moreover, by calculus, we have

$$\begin{aligned} \Delta_r(\mathbb{Q}, \alpha) &= Threshold_Q(\mathbb{Q}, \alpha) - Threshold_r(\mathbb{Q}, \alpha) \\ &= \begin{cases} \frac{\alpha^{Q-r} - 1}{\alpha - 1} & if\ \alpha \neq 1 \\ Q - r & if\ \alpha = 1 \end{cases} \end{aligned}$$

Hence the recency threshold of (4α) can rewritten as

$$\sum_{q=1}^{Q} \alpha^{q-1}\left|x_j - x_j^q\right| \leq \Delta_r(\mathbb{Q}, \alpha). \tag{4*a}$$

Let $x_j^{\#}$ denote the current value for x_j, the following describes the nature of the acceptable vectors depending on the value of α.

Proposition 1.

(a) For $r = Q$ and any α, the vector $V_j(\mathbb{Q}, Q, x, \alpha) = (x_j^{\#}, x_j^{\#}, \ldots, x_j^{\#})$ is the single acceptable vector, i.e. $\sum_{q=1}^{Q} |x_j - x_j^q| = 0$.

(b) For $\alpha = 1$, any binary vector $V_j(\mathbb{Q}, r, x, 1)$ is an acceptable vector, such that

$$\sum_{q=1}^{Q} \left|x_j - x_j^q\right| \leq Q - r \tag{4**a}$$

(c) For $\alpha \neq 1$, any vector $V_j(\mathbb{Q}, r, x, \alpha)$ such that $x_j^{\#} = x_j^q$ for all $q \in \{Q, \ldots, Q - r\}$ is an acceptable vector, i.e. $\sum_{q=r}^{Q} |x_j - x_j^q| = 0$.

(d) For $\alpha \geq 2$, any vector $V_j(\mathbb{Q}, r, x, \alpha)$ such that $x_j^{\#} \neq x_j^s$ for a given $s \in \{Q, \ldots, Q - r\}$ is an unacceptable vector, i.e. $\sum_{q=r}^{Q} |x_j - x_j^q| \geq 1$.

(e) For $0 < \alpha < 2$, there exist vectors $V_j(\mathbb{Q}, r, x, \alpha)$ such that $x_j^{\#} \neq x_j^q$ for a given $q \in \{Q, \ldots, Q - r\}$ which is an acceptable vector, i.e. $\sum_{q=r}^{Q} |x_j - x_j^q| \geq 1$.

Proof. The statements a) and b) is trivial and are deduced directly from ($4^*\alpha$). It is easy to see from ($4^*\alpha$) that if $x_j^{\#} = x_j^q$ for all

$q \in \{Q, \ldots, Q-r\}$ in a vector $V_j(\mathbb{Q}, x, \alpha)$ then $\sum_{q=1}^{Q} \alpha^{q-1}|x_j - x_j^q| = \sum_{q=1}^{Q-r} \alpha^{q-1}|x_j - x_j^q| \le \sum_{q=1}^{Q-r} \alpha^{q-1} = \frac{\alpha^{Q-r}-1}{\alpha-1}$. This validates the statement c). Now assume $\alpha \ge 2$ and there exists $s \in \{Q, \ldots, Q-r\}$ such that $x_j^{\#} \ne x_j^s$. Then an acceptable vector $V_j(\mathbb{Q}, x, \alpha)$ must satisfy

$$\alpha^{Q-s} \le \sum_{q=1}^{Q} \alpha^{q-1}\left|x_j - x_j^q\right| \le \frac{\alpha^{Q-r}-1}{\alpha-1}$$

This is equivalent to the inequality

$$\alpha^{Q-r}\left(1 - \alpha^{r-s}(\alpha-1)\right) \ge 1$$

which is impossible for all $\alpha \ge 2$ and $s \in \{Q, \ldots, Q-r\}$ since $\alpha^{r-s}(\alpha-1) \ge 2$. This completes the proof of the statement d). The following example will show the validity of statement e). □

Let $a_{\alpha r}$ denote the number of acceptable vectors and $a^+_{\alpha r}$ denote the number of acceptable vectors such that $\sum_{q=r}^{Q}|x_j - x_j^q| \ge 1$. Observe that $a_{\alpha r} - a^+_{\alpha r} = 2^r$ which corresponds to the number of acceptable vectors such that $\sum_{q=r}^{Q}|x_j - x_j^q| = 0$. For $Q = 7$, the following table shows the values of $a_{\alpha r}$ and $a^+_{\alpha r}$ where $\alpha \in \{2.0, 1.9, \ldots, 0.2, 0.1\}$ and $r \in \{1, 2, \ldots, Q-1\}$.

α	$a_{\alpha 1}$	$a^+_{\alpha 1}$	$a_{\alpha 2}$	$a^+_{\alpha 2}$	$a_{\alpha 3}$	$a^+_{\alpha 3}$	$a_{\alpha 4}$	$a^+_{\alpha 4}$	$a_{\alpha 5}$	$a^+_{\alpha 5}$	$a_{\alpha 6}$	$a^+_{\alpha 6}$
2.0	64	0	32	0	16	0	8	0	4	0	2	0
1.9	69	5	34	2	17	1	8	0	4	0	2	0
1.8	74	10	37	5	18	2	9	1	4	0	2	0
1.7	79	15	39	7	19	3	9	1	4	0	2	0
1.6	85	21	42	10	21	5	10	2	5	1	2	0
1.5	91	27	47	15	22	6	10	2	5	1	2	0
1.4	100	36	56	24	26	10	12	4	5	1	2	0
1.3	109	45	69	37	34	18	15	7	6	2	2	0
1.2	116	52	82	50	44	28	19	11	7	3	2	0
1.1	121	57	99	67	63	47	29	21	9	5	2	0
1.0	127	63	120	88	99	83	64	56	29	25	8	6
0.9	127	63	120	88	101	85	68	60	31	27	8	6
0.8	127	63	123	91	113	97	91	83	55	51	17	15
0.7	127	63	124	92	117	101	103	95	74	70	31	29
0.6	127	63	125	93	120	104	110	102	89	85	48	46
0.5	127	63	125	93	121	105	113	105	97	93	65	63
0.4	127	63	125	93	121	105	113	105	97	93	65	63
0.3	127	63	125	93	121	105	113	105	97	93	65	63
0.2	127	63	125	93	121	105	113	105	97	93	65	63
0.1	128	64	127	95	123	107	115	107	99	95	67	65

The following table gives weights α^q for $q \in \{Q-1=6, \ldots, 0\}$ and $\alpha \in \{2.0, 1.9, \ldots, 0.2, 0.1\}$. The last two columns correspond to $\Delta_r(\mathbb{Q}, \alpha)$ and $Threshold_r(\mathbb{Q}, \alpha)$ respectively for fixed $r = 3$.

α	6	5	4	3	2	1	0	$\Delta_3(\mathbb{Q}, \alpha)$	$Threshold_3(\mathbb{Q}, \alpha)$
2	64	32	16	8	4	2	1	15	112
1.9	47.045881	24.76099	13.032	6.859	3.61	1.9	1	13.369	84.83897
1.8	34.012224	18.89568	10.498	5.832	3.24	1.8	1	11.872	63.4055
1.7	24.137569	14.19857	8.3521	4.913	2.89	1.7	1	10.503	46.68824
1.6	16.777216	10.48576	6.5536	4.096	2.56	1.6	1	9.256	33.81658
1.5	11.390625	7.59375	5.0625	3.375	2.25	1.5	1	8.125	24.04688
1.4	7.529536	5.37824	3.8416	2.744	1.96	1.4	1	7.104	16.74938
1.3	4.826809	3.71293	2.8561	2.197	1.69	1.3	1	6.187	11.39584
1.2	2.985984	2.48832	2.0736	1.728	1.44	1.2	1	5.368	7.547904
1.1	1.771561	1.61051	1.4641	1.331	1.21	1.1	1	4.641	4.846171
1	1	1	1	1	1	1	1	4	3
0.9	0.531441	0.59049	0.6561	0.729	0.81	0.9	1	3.439	1.778031
0.8	0.262144	0.32768	0.4096	0.512	0.64	0.8	1	2.952	0.999424
0.7	0.117649	0.16807	0.2401	0.343	0.49	0.7	1	2.533	0.525819
0.6	0.046656	0.07776	0.1296	0.216	0.36	0.6	1	2.176	0.254016
0.5	0.015625	0.03125	0.0625	0.125	0.25	0.5	1	1.875	0.109375
0.4	0.004096	0.01024	0.0256	0.064	0.16	0.4	1	1.624	0.039936
0.3	0.000729	0.00243	0.0081	0.027	0.09	0.3	1	1.417	0.011259
0.2	6.4E-05	0.00032	0.0016	0.008	0.04	0.2	1	1.248	0.001984
0.1	1E-06	1E-05	1E-04	1E-03	0.01	0.1	1	1.111	0.000111

To describe the next sets of acceptable vectors as a function of the value α, we associate to an acceptable vector $V_j(\mathbb{Q}, r, x, \alpha)$ the following binary vector of dimension Q

$$\tilde{V}_j(\mathbb{Q}, r, x, \alpha) = \left(\left|x_j^{\#} - x_j^Q\right|, \left|x_j^{\#} - x_j^{Q-1}\right|, \ldots, \left|x_j^{\#} - x_j^1\right|\right).$$

In other terms, each binary component q of $\tilde{V}_j(\mathbb{Q}, x)$ is

$$\tilde{V}_j^q(\mathbb{Q}, r, x, \alpha) = \begin{cases} 0 & if\ x_j^{\#} = x_j^q \\ 1 & if\ x_j^{\#} \ne x_j^q \end{cases}$$

This establishes a one-one correspondence between vectors $V_j(\mathbb{Q}, r, x, \alpha)$ and $\tilde{V}_j(\mathbb{Q}, r, x, \alpha)$. For a given vector $\tilde{V}_j(\mathbb{Q}, r, x, \alpha)$, we obtain a vector $V_j(\mathbb{Q}, r, x, \alpha)$ such that

$$V_j^q(\mathbb{Q}, r, x, \alpha) = \begin{cases} x_j^{\#} & if\ \tilde{V}_j^q(\mathbb{Q}, r, x, \alpha) = 0 \\ 1 - x_j^{\#} & if\ \tilde{V}_j^q(\mathbb{Q}, r, x, \alpha) = 1 \end{cases}$$

Hence, we call the binary vector $\tilde{V}_j(\mathbb{Q}, r, x, \alpha)$ *acceptable* if it satisfies the recency threshold

$$\sum_{q=1}^{Q} \alpha^{q-1}\tilde{V}_j^q(\mathbb{Q}, r, x, \alpha) \le \begin{cases} \frac{\alpha^{Q-r}-1}{\alpha-1} & if\ \alpha \ne 1 \\ Q - r & if\ \alpha = 1 \end{cases}. \quad (4^{**}\text{a})$$

Let $A_j(\mathbb{Q}, r, x, \alpha)$ be the set of acceptable vectors $\tilde{V}_j(\mathbb{Q}, r, x, \alpha)$. We observe for $Q = 7$, and $r = 3$ that the sets $A_j(\mathbb{Q}, r, x, \alpha)$ are nested, i.e. for $0 < \alpha < \alpha + \epsilon \le 2$, we have

$$A_j(\mathbb{Q}, r, x, \alpha) \subseteq A_j(\mathbb{Q}, r, x, \alpha + \epsilon)$$

Therefore, let # denote the option of either #=0 or 1. The following table provides the difference set

$$D_j(\mathbb{Q}, r, x, \alpha) = A_j(\mathbb{Q}, r, x, \alpha + \epsilon) - A_j(\mathbb{Q}, r, x, \alpha)$$

with $\epsilon = 0.1$. To reduce the size of the table, we use the following notation

$$E_{r,k} = \left\{ y \in \{0,1\}^Q : \sum_{i=r}^{Q} y_i = k \right\}.$$

Note that $A_j(\mathbb{Q}, r, x, 2) = E_{r,0}$, hence $A_j(\mathbb{Q}, r, x, 1.9) = (0,0,0,0,1,0,0) + A_j(\mathbb{Q}, r, x, 2)$.

α	$\lvert D_j(\mathbb{Q}, r, x, \alpha)\rvert$	$D_j(\mathbb{Q}, r, x, \alpha)$
1.9	1	(0,0,0,0,1,0,0)
1.8	1	(1,0,0,0,1,0,0)
1.7	1	(0,1,0,0,1,0,0)
1.6	2	(0,0,#,0,1,0,0)
1.5	1	(0,0,0,0,0,1,0)
1.4	4	(0,0,0,1,1,0,0) + (#,1-#,0,0,0,1,0) + (1,0,1,0,1,0,0)
1.3	8	(#,0,0,0,0,0,1) + (0,0,#,1-#,0,1,0) + (0,1,0,0,0,0,1) + (0,1,1,0,1,0,0) + (1,0,0,1,1,0,0) + (1,1,0,0,0,1,0)
1.2	10	(0,0,0,0,1,#,1-#) + (0,0,#,1-#,0,0,1) + (0,#,1-#,1,1,0,0) + (0,1,1,0,0,1,0) + (1,0,#,1-#,0,1,0) + (1,1,0,0,0,0,1)
1.0	19	(0,0,0,0,0,1,1) + (0,0,0,1,1,#,1-#) + (#,1-#,1,0,0,0,1) + ($E_{7,3} \cap (E_{3,1}$ − (0,#,1-#,1,1,0,0)))
0.9	36	(0,0,0,#,1-#,1,1) + $E_{7,4}$ − (1,1,1,1,0,0,0)
0.8	2	(0,#,1-#,1,1,1,1)
0.7	12	(0,1,1,1,1,1,1) + $E_{7,5} - E_{3,1} - E_{3,3}$ − (1,1,0,1,1,#,1-#)
0.6	4	(1,0,1,1,1,1,1) + (1,1,0,1,1,#,1-#) + (1,1,1,0,0,1,1)
0.5	3	(1,1,0,1,1,1,1) + (1,1,1,0,1,#,1-#)
0.4–2		∅
0.1	2	(1,1,1,1,0,#,1)

First, note that the only acceptable $V_j(\mathbb{Q}, r, x, \alpha)$ vectors for $\alpha = 2$ have the form

$$V_j(\mathbb{Q}, r, x, 2) = \left(x_j^{\#}, x_j^{\#}, x_j^{\#}, \#, \#, \#, \#\right)$$

This means that $x_j = x_j^{\#}$ in each of the 3 most recent local optima x^Q, x^{Q-1} and x^{Q-1} (i.e., $x_j^Q = x_j^{\#}, x_j^{Q-1} = x_j^{\#}$ and $x_j^{Q-2} = x_j^{\#}$), while $x_j = x_j^{\#}$ and $x_j = 1 - x_j^{\#}$ are both possible in earlier local optima x^{Q-3} to x^1. Requiring $EE_j(\mathbb{Q}, x, \alpha) \ge Threshold_r(\mathbb{Q}, \alpha)$ therefore compels $x_j = x_j^{\#}$ in the 3 most recent local optima when $\alpha = 2$. When $\alpha < 2$, other $V_j(\mathbb{Q}, r, x, \alpha)$ vectors in addition to $V_j(\mathbb{Q}, r, x, 2) = (x_j^{\#}, x_j^{\#}, x_j^{\#}, \#, \#, \#, \#)$ can satisfy the recency threshold. Consequently, in some cases $x_j = x_j^{\#}$ may not be required for each of the 3 most recent local optima.

Acceptable $V_j(\mathbb{Q}, r, x, \alpha)$ vectors for $\alpha = 2, 1.7$ and 1.5

For $\alpha = 2$: $(x_j^\#, x_j^\#, x_j^\#, \#, \#, \#, \#)$

For $\alpha = 1.7$: $(x_j^\#, x_j^\#, x_j^\#, \#, \#, \#, \#)$,
$(x_j^\#, x_j^\#, 1 - x_j^\#, x_j^\#, x_j^\#, x_j^\#, \#)$,$(x_j^\#, x_j^\#, 1 - x_j^\#, x_j^\#, x_j^\#, 1 - x_j^\#, x_j^\#)$,
(3 more options than for $\alpha = 2$, accounting for $\# = 0$ or 1)

For $\alpha = 1.5$: $(x_j^\#, x_j^\#, x_j^\#, \#, \#, \#, \#)$, $(x_j^\#, x_j^\#, 1 - x_j^\#, x_j^\#, x_j^\#, \#, \#)$,
$(x_j^\#, x_j^\#, 1 - x_j^\#, x_j^\#, 1 - x_j^\#, 1 - x_j^\#, 1 - x_j^\#)$,
$(x_j^\#, 1 - x_j^\#, x_j^\#, x_j^\#, x_j^\#, x_j^\#, x_j^\#)$, (6 more options than for $\alpha = 2$, accounting for $\# = 0$ or 1)

To further see the relevance of these differences, recall that Strategy $S^=$ uses the recency threshold $EE_j(\mathbb{Q}, x, \alpha) \geq Threshold_r(\mathbb{Q}, \alpha)$ when the $x_j = x_j^\#$ in the most recent local optimum ($x_j^Q = x_j^\#$ in x^Q), and we want to decide whether to change x_j to give $x_j = 1 - x_j^\#$ (under conditions where this change is evaluated to improve the current solution). As previously emphasized, when $\alpha = 2$, changing x_j to give $x_j = 1 - x_j^\#$ causes x_j to take a different value than in the 3 most recent local optima, and hence we will not duplicate any of these local optima as long as x_j retains its new value of $1 - x_j^\#$.

When $\alpha = 1.7$ above, the solutions $(x_j^\#, x_j^\#, 1 - x_j^\#, x_j^\#, x_j^\#, x_j^\#, \#)$ and $(x_j^\#, x_j^\#, 1 - x_j^\#, x_j^\#, x_j^\#, 1 - x_j^\#, x_j^\#)$ show that changing $x_j = x_j^\#$ to $x_j = 1 - x_j^\#$ would cause the new solution to have a different value than in the two most recent local optima (where $x_j^Q = x_j^{Q-1} = x_j^\#$), but there are three cases where changing $x_j = x_j^\#$ to $x_j = 1 - x_j^\#$ would yield the same x_j value as in the third most recent local optimum (where $x_j^{Q-2} = 1 - x_j^\#$ in these solutions). Consequently, there would be a possibility that changing $x_j = x_j^\#$ to $x_j = 1 - x_j^\#$ would permit the third most recent local optimum to be revisited. This possibility might not be large, considering that most of the local optima avoided by $\alpha = 1.7$ are represented by the solutions $(x_j^\#, x_j^\#, x_j^\#, \#, \#, \#, \#)$. The risk of revisiting the r^{th} most recent solution would also clearly have a smaller impact if r is somewhat greater than 3. The risk would further be diminished if other variables x_j likewise satisfied the recency threshold, since each of these instances would mostly avoid the solutions represented by $(x_j^\#, x_j^\#, x_j^\#, \#, \#, \#, \#)$.

The case for $\alpha = 1.5$ shows this smaller α value poses additional risks beyond $\alpha = 1.7$ of revisiting solutions other than $(x_j^\#, x_j^\#, x_j^\#, \#, \#, \#, \#)$. One of these involves a risk of duplicating the second most recent local optimum. (Since this solution is the one indexed x^{r-1}, the significance of this risk is not very great as r becomes larger.)

In all of these cases, the risk may be additionally reduced as the number of moves away from the most recent local optimum increases, since this produces a chance that the ascent to a new local optimum would be launched from a point farther away from previous local optima. However, greater assurance would be provided by the trigger threshold that postpones the Ascent Phase until an increased number of different x_k variables are identified by Strategies $S^=$ and $S^{\neq}$ whose $V_k(\mathbb{Q}, r, x, \alpha)$ vectors satisfy $EE_k(\mathbb{Q}, x, \alpha) \geq Threshold_r(\mathbb{Q}, \alpha)$.

As in the case of $\alpha = 2$, it is not necessary to record these $V_k(\mathbb{Q}, r, x, \alpha)$ vectors, since the simple update of $EE_j^1(\mathbb{Q})$ for all j can be used with the general form of (5) where α replaces 2; i.e.,

$$EE_j^1(\mathbb{Q}) \;=\; \alpha^{Q-1}x_j^Q + EE_j^1(\mathbb{Q})/\alpha$$

and

$$EEbase(\mathbb{Q}) = \alpha^{Q-1} + EEbase(\mathbb{Q})/\alpha.$$

By these observations it is clear that there may be merit in exploring the use of α values other than $\alpha = 2$ when exponential extrapolation is embedded in an adaptive memory strategy. For example, the preceding examples show that smaller α values can avoid revisiting some local optima beyond the first r, and this might be additionally exploited by choosing larger r values for smaller α values. The chief appeal of using an α value less than 2 is that it allows greater latitude in the choice of variables that qualify for launching a new Ascent Phase by Strategy $S^=$ or $S^{\neq}$.

A Mixed α Strategy

When selecting an α value less than 2, it is desirable to use a "mixed α strategy" where the first term of the sequence

$$EE_j^1(\mathbb{Q}) = \sum_{q=1}^{Q} \alpha^{q-1}x_j^q$$

replaces $\alpha^{q-1}x_j^q$ by $2^{q-1}x_j^q$ to give the mixed sequence

$$EE_j^1(\mathbb{Q}) = \sum_{q=1}^{Q} 2^{q-1}x_j^q$$

which similarly gives

$$EEbase(\mathbb{Q}) = \sum_{q=1}^{Q} 2^{q-1}$$

$$Threshold_r(\mathbb{Q}) = \sum_{q=1}^{r} 2^{Q-q}.$$

The reason for making the last coefficient in this sequence 2^{Q-1} instead of α^{Q-1} is to assure that satisfying the recency threshold will always imply that a variable cannot duplicate its value in the most recent solution x^Q and additionally yield $2^{Q-1} > \sum_{q=1}^{Q-1} \alpha^{q-1}$, as in the case where $\alpha = 2$. This latter outcome allows us to update $EE_j(\mathbb{Q}, x)$, $EEbase(\mathbb{Q})$ and $Threshold_r$ by a slight generalization of the rule for the case where $\alpha = 2$, without having to save the value x_j^Q. We won't go through the full algebraic derivation but identify the key changes in the formulas (5.23) (5.24 and (5.3) for updating $EE_j(\mathbb{Q}, x)$ and $EEbase(\mathbb{Q})$, which give the following formulas. Define $A = 2^{Q-1}$ and $B = \alpha^{Q-1}$ once a first local optimum is identified, and before that, initialize $A = B = 0$, just as we initialize $EEbase(\mathbb{Q}) = 0$ and $EE_j(\mathbb{Q}, x) = 0$ for all j. Then the new formulas become

$$EE_j(\mathbb{Q}, x) = \begin{cases} 2^{Q-1} \;+\; \big(EE_j(\mathbb{Q}, x) + B - A\big)/\alpha & \text{if } EE_j(\mathbb{Q}, x) \leq 2^{Q-1} \\ 2^{Q-1} \;+\; \big(EEbase(\mathbb{Q}) - EE_j(\mathbb{Q}, x)\big)/\alpha & \text{if } EE_j(\mathbb{Q}, x) < \; 2^{Q-1} \end{cases}$$

These are executed sequentially each time a local optimum is found, followed by setting

$$EEbase(\mathbb{Q}) = 2^{Q-1} + (EEbase(\mathbb{Q}) + B - A)/\alpha$$

$$A = 2^{Q-1} \text{ and } B = \alpha^{Q-1}$$

It is easy to confirm that these formulas reduce to the formulas given in Section 6 when $\alpha = 2$.

The modifications of the pseudocode are correspondingly straightforward. The Preliminary Initialization adds the following two instructions:

- Choose a value for α (e.g., 1.5, 1.7 or 2)
- $A = B = 0$.

Then in the procedure $UpdateEE()$ is replaced by the following procedure:

Procedure UpdateMixedEE(){
For $j = 1$ to n **do**
If $EE_j \geq 2^{Q-1}$ **then** $EE_j = 2^{Q-1} + (EE_j + B - A)/\alpha$
Else $EE_j = 2^{Q-1} + (EEbase - EE_j)/\alpha$
Endfor
$A = 2^{Q-1}$ and $B = \alpha^{Q-1}$
$EEbase = 2^{Q-1} + 2^{-1}EEbase$
$Threshold_r = \min(EEbase, 2^{Q-r}(2^r - 1))$
} // **End** ***UpdateMixedEE***

It is easy to show the following properties:

- $EE_j(\mathbb{Q}, x) = EEbase(\mathbb{Q}) - \sum_{q=1}^{Q} 2^{q-1}|x_j - x_j^q|$
- $EE_j(\mathbb{Q}, x) = x_j EE_j^1(\mathbb{Q}) + (1 - x_j)EE_j^0(\mathbb{Q})$
- $EE_j(\mathbb{Q}, x) = (2x_j - 1)EE_j^1(\mathbb{Q}) + (1 - x_j)EEbase(\mathbb{Q})$
- $EE_j(\mathbb{Q}, x) = \begin{cases} EE_j^1(\mathbb{Q}) & if\ x_j = 1 \\ EEbase(\mathbb{Q}) - EE_j^1(\mathbb{Q}) & if\ x_j = 0 \end{cases}$
- $EE_j^1(\mathbb{Q}) = (2x_j - 1)EE_j(\mathbb{Q}, x) + (1 - x_j)EEbase(\mathbb{Q})$
- $EE_j^1(\mathbb{Q}) = \begin{cases} EE_j(\mathbb{Q}, x) & if\ x_j = 1 \\ EEbase(\mathbb{Q}) - EE_j(\mathbb{Q}, x) & if\ x_j = 0 \end{cases}$
- $EE_j^1(\mathbb{Q}) = \begin{cases} EE_j(\mathbb{Q}, x) & if\ x_j = 1 \\ EEbase(\mathbb{Q}) - EE_j(\mathbb{Q}, x) & if\ x_j = 0 \end{cases}$

Appendix 2. Tradeoff Relationships

A refers to a current evaluation and B refers to a previous evaluation, such as the best before now. A_1 and B_1 refer to the first type of evaluation and A_2 and B_2 refer to the second type of evaluation. We assume the second type of evaluation, A_2 and B_2, is always nonnegative (as in the case of $EE_j(\mathbb{Q}, x)$), but the first type, A_1 and B_1, can sometimes be negative (as in the case of $Eval_j(x)$). The current evaluation will dominate the previous evaluation if $A \geq B$; i.e.

$A_1 \geq B_1$ and $A_2 \geq B_2$.

Assume dominance does not occur. Then we have two possibilities.

Case 1. $A_1 > B_1$ and $A_2 < B_2$

Case 2. $A_1 < B_1$ and $A_2 > B_2$

Consider these two cases in the context of conditions satisfied by moves in $N^1(x)$ and $N^2(x)$, which we write as follows:

Condition 1. $A_1, B_1 \geq 0$. $A_2, B_2 \geq 0$

Condition 2. $A_1, B_1 \leq 0$. $A_2, B_2 \geq 0$.

These conditions correspond to conditions defined by reference to the sets $N^1(x)$ and $N^2(x)$ of Section 3.2 where A_1 and B_1 refer to $Eval_j(x)$ and A_2 and B_2 refer to $EE_j(\mathbb{Q}, x)$. However, the conditions here are less stringent than those of Section 3.2, since they do not include reference to Tabu restrictions or the recency threshold or the $S^=$ status of variables. In addition, moves in $N^1(x)$ would imply $A_1, B_1 > 0$ rather than $A_1, B_1 \geq 0$. We note, however, that we can translate every case for Condition 2 into Condition 1 by identifying a lower bound LB for all instances A_1 and B_1 such that $A_1, B_1 \geq LB$, and redefining

$A_1 = A_1 - LB;\ \ B_1 = B_1 - LB$.

Without identifying LB, we consider Conditions 1 and 2 separately. For each combination of conditions and cases, we identify the max and min values of the A and B components.

Condition 1 & Case 1: The combination of Condition 1 and Case 1 yields $A_1 > B_1 \geq 0$, hence $A_1 > 0$, and we seek a nonnegative multiple x so that $A_1 x$ dominates B_1, as given by $A_1 x \geq B_1$. We also have $B_2 > A_2 \geq 0$, hence $B_2 > 0$, and we seek a nonnegative multiple x so that A_2 dominates $B_2 x$, as given by $A_2 \geq B_2 x$. An x that yields dominance in both situations gives $A_2/B_2 \geq x \geq B_1/A_1$ or equivalently

$A_1A_2/A_1B_2 \geq x \geq B_1B_2/A_1B_2$

Hence dominance and strict dominance are respectively achieved by

$A_1A_2 \geq B_1B_2$ and $A_1A_2 > B_1B_2$.

In terms of $Eval_j(x)$ and $EE_j(\mathbb{Q}, x)$ this corresponds to $Eval_j(x) \times EE_j(\mathbb{Q}, x) > EE_j^p(\mathbb{Q}, x) > Eval_j^p(x) \times EE_j^p(\mathbb{Q}, x)$, where the "$p$" exponent represents "previous".

Condition 1 & Case 2: Corresponding analysis gives $A_1 \geq B_1 x$ and $A_2 x \geq B_2$ to yield

$A_1A_2/A_2B_1 \geq x \geq B_1B_2/A_2B_1$

and while the denominator is different, the conclusions for dominance and strict dominance are the same as in Condition 1 & Case 1, i.e. $A_1A_2 \geq B_1B_2$ and $A_1A_2 > B_1B_2$.

Condition 2 & Case 1: We now have $0 \geq A_1 > B_1$, hence $B_1 < 0$, and we seek a nonnegative multiple x so that A_1 dominates $B_1 x$, hence

$A_1 \geq B_1 x$ or $-B_1 x \geq -A_1$.

Likewise, we have $B_2 > A_2 \geq 0$, hence $B_2 > 0$, and we seek a nonnegative multiple x so that A_2 dominates $B_2 x$, hence

$A_2 \geq B_2 x$.

Since $-B_1 > 0$, the two inequalities become

$A_2/B_2 \geq x \geq -A_1/-B_1$.

or equivalently

$-A_2B_1/-B_1B_2 \geq x \geq -A_1B_2/-B_1B_2$

with positive denominators. Hence dominance and strict dominance are achieved by

$-A_2B_1 \geq -A_1B_2$ $(A_1B_2 \geq A_2B_1)$ and $-A_2B_1 > -A_1B_2$ $(A_1B_2 > A_2B_1)$

In terms of $Eval_j(x)$ and $EE_j(\mathbb{Q}, x)$ this corresponds to $Eval_j(x) \times EE_j^p(\mathbb{Q}, x) > Eval_j^p(x) \times EE_j(\mathbb{Q}, x)$, where the "$p$" exponent again represents "previous".

Condition 2 & Case 2: Following the line of argument as in Condition 2 & Case 1, we conclude

$-A_2B_1/A_1A_2 \geq x \geq -A_1B_2/A_1A_2$

which yields the same dominance conclusions as in Condition 2 & Case 1.

We remark that the conclusions in all these cases can also be reached by a more involved derivation using a different definition of dominance, where A dominates B if

$f(A_1, B_1) \geq f(A_2, B_2)$

where for $i = 1, 2$

$f(A_i, B_i) = (Max(A_i, B_i) - Min(A_i, B_i))/(|A_i| + |B_i|)$.

Proposition 2. *Let x' the solution obtained from x by flipping the variable x_k,* i.e. $x' = x + (1 - 2x_k)e^k$. *Then the update of the evaluation $Eval_j(x')$ can be computed using the rule*

$$Eval_j(x') = \begin{cases} -Eval_j(x) & if\ j = k \\ Eval_j(x) + a'_{jk} & if\ j \neq k\ and\ x_j = x_k \\ Eval_j(x) - a'_{jk} & if\ j \neq k\ and\ x_j \neq x_k \end{cases}$$

Justification. In case the input matrix A is a lower triangular matrix, let $A' = (a'_{ij})$ denote its associated symmetric matrix defined as follows

$$a'_{ij} = \begin{cases} a_{ij} & if\ i < j \\ a_{ii} & if\ i = j \\ a_{ji} & if\ i > j \end{cases}$$

Consequently, we have $A'_j x = (A^j + A_j)x - a'_{jj}x_j$, hence the initial evaluation $Eval_j(x)$ can be calculated in linear time using the formula $Eval_j(x) = (1 - 2x_j)A'_j x + (1 - x_j)a'_{jj}$. Hence, we have

$$Eval_j(x') = (1 - 2x'_j)A'_j x' + (1 - x'_j)a'_{jj}$$

$$Eval_j(x') = (1 - 2x'_j)A'_j(x + (1 - 2x_k)e^k) + (1 - x'_j)a'_{jj}$$

$$Eval_j(x') = (1 - 2x'_j)A'_j x + (1 - 2x'_j)A'_j(1 - 2x_k)e^k + (1 - x'_j)a'_{jj}$$

$$Eval_j(x') = (1 - 2x'_j)A'_j x + (1 - 2x'_j)(1 - 2x_k)a'_{jk} + (1 - x'_j)a'_{jj}$$

Two cases are considered:

Case 1: $j \neq k \rightarrow x'_j = x_j$

$$Eval_j(x') = (1 - 2x_j)A'_j x + (1 - 2x_j)(1 - 2x_k)a'_{jk} + (1 - x_j)a'_{jj}$$

$$Eval_j(x') = Eval_j(x') + (1 - 2x_j)(1 - 2x_k)a'_{jk}$$

Case 2: $j = k \rightarrow x'_k = 1 - x_k$

$$Eval_k(x') = (2x_k - 1)A'_k x + (2x_k - 1)(1 - 2x_k)a'_{kk} + x_k a'_{kk}$$

$$Eval_k(x') = (2x_k - 1)A'_k x + (x_k - 1)a'_{kk}$$

$$Eval_k(x') = -Eval_k(x). \square$$

References

Barr, R. S., Glover, F., Huskinson, T., & Kochenberger, G. (2021). An extreme–point tabu-search algorithm for fixed-charge network problems. *Networks, 77*(2), 322–340.

Faigle, U., & Kern, W. (1992). Some convergence results for probabilistic Tabu search. *ORSA Journal on Computing, 4*(1), 32–37.

Gendreau, M., & Potvin, J. Y. (2005). Tabu search. *Search methodologies* (pp. 165–186). Boston, MA: Springer.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research, 13*(5), 533–549.

Glover, F. (1989). Tabu search—part I. *ORSA Journal on Computing, 1*(3), 190–206.

Glover, F. (1995). Tabu thresholding: Improved search by nonmonotonic trajectories. *ORSA Journal on Computing, 7*(4), 426–442.

Glover, F. (2020). Exploiting Local Optimality in Metaheuristic Search, arXiv:2010.05394v3 [cs.AI].

Glover, F., & Hanafi, S. (2002). Tabu search and finite convergence. *Discrete Applied Mathematics, 119*(1–2), 3–36.

Glover, F., & Laguna, M. (1997). *Tabu search*. Kluwer Academic Publishers, Springer.

Guemri, O., Nduwayo, P., Todosijević, R., Hanafi, S., & Glover, F. (2019). Probabilistic tabu search for the cross-docking assignment problem. *European Journal of Operational Research, 277*(3), 875–885.

Hanafi, S. (2001). On the convergence of tabu search. *Journal of heuristics, 7*(1), 47–58.

Karamichailidou, D., Kaloutsa, V., & Alexandridis, A. (2021). Wind turbine power curve modeling using radial basis function neural networks and tabu search. *Renewable Energy, 163*, 2137–2152.

Laguna, M., & Glover, F. (1993). Integrating target analysis and tabu search for improved scheduling systems. *Expert Systems with Applications, 6*(3), 287–297.

Qiu, M., Fu, Z., Eglese, R., & Tang, Q. (2018). A Tabu Search algorithm for the vehicle routing problem with discrete split deliveries and pickups. *Computers & Operations Research, 100*, 102–116.

Servranckx, T., & Vanhoucke, M. (2019). A tabu search procedure for the resource–constrained project scheduling problem with alternative subgraphs. *European Journal of Operational Research, 273*(3), 841–860.