Honey Bee Mating Optimization for generating personalized food menu recommendations

Contents

[2 Representation of a Solution (bee) 1](#_Toc420411468)

[3 The Fitness Function 2](#_Toc420411469)

[3.1 Definition of the fitness function 2](#_Toc420411470)

[3.2 The “ideal” functions for components 3](#_Toc420411471)

[3.2.1 Computation of the ideal values for nutrients 3](#_Toc420411472)

[3.2.2 Computation of the ideal values for food items 7](#_Toc420411473)

[3.2.3 Ideal values for cost and delivery time 7](#_Toc420411474)

[3.3 The weights for components 8](#_Toc420411475)

[3.3.1 Weights for nutrients 8](#_Toc420411476)

[3.3.2 Weights for food items 8](#_Toc420411477)

[3.3.3 Weights for cost and deliver time 8](#_Toc420411478)

[3.4 The Error Function 9](#_Toc420411479)

[3.4.1 Level 1 Error Function 9](#_Toc420411480)

[3.4.2 Level 2 Error Function 11](#_Toc420411481)

[3.4.3 Level 3 Error Function 12](#_Toc420411482)

[3.5 The fitness function template 14](#_Toc420411483)

[3.5.1 Definition 14](#_Toc420411484)

[3.5.2 Level 1 fitness function: nutrients 14](#_Toc420411485)

[3.5.3 Level 2 fitness function: ingredients 15](#_Toc420411486)

[3.5.4 Level 3 Fitness Function: Cost and Delivery time 15](#_Toc420411487)

[3.5.5 The top level fitness function 15](#_Toc420411488)

[4 Hybridization points 15](#_Toc420411489)

[4.1 Brood creation strategies 16](#_Toc420411490)

[4.1.1 The Crossover-based broods creation strategy 16](#_Toc420411491)

[4.1.2 Path-Relinking-based broods creation strategy 17](#_Toc420411492)

[4.2 The worker modification strategy 18](#_Toc420411493)

[4.2.1 The Random Mutation algorithm 20](#_Toc420411494)

[4.2.2 Neighborhood of a solution 21](#_Toc420411495)

[4.2.3 Hill Climbing 22](#_Toc420411496)

[4.2.4 Simulated Annealing 22](#_Toc420411497)

[4.2.5 Tabu Search 24](#_Toc420411498)

[5 Bibliography 25](#_Toc420411499)

# Representation of a Solution (bee)

The solution is a meal for an entire day. In principle, it contains breakfast, lunch, dinner and two snacks meals which are the five main components of a solution.

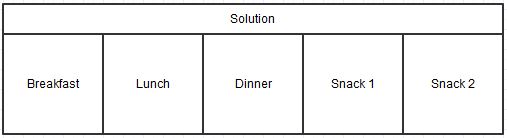


Figure 1 The main components of a solution

Each of these can have a starter, main course and/or desert dish. The following illustration shows the relationships between the meals and the dishes.

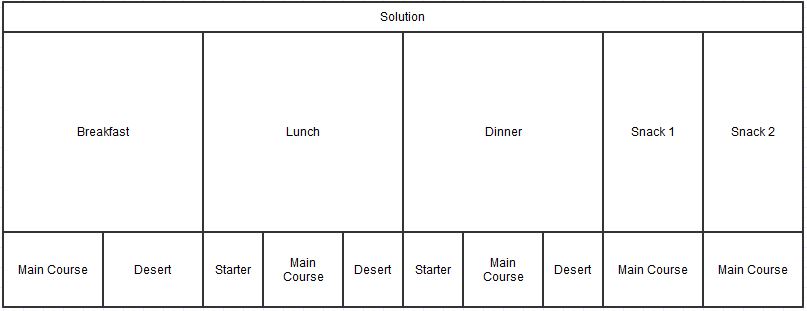


Figure 2 Relationships meals-dishes

Each components contains nutritional information (number of calories, carbohydrates, fats etc.).

# The Fitness Function

## Definition of the fitness function

The fitness function is an evaluation function that rates each state or solution. Usually, it returns a higher value for better states, and of course, lower for worse states.

For the diet of a person, the fitness function models the quality of the menu, in terms of following factors:

1. Kilocalories;
2. Macro-nutrients:
   * Carbohydrates;
   * Proteins;
   * Fats;
3. Micro-nutrients:
   * Vitamins A, B, C, D;
   * Calcium;
   * Iron;
   * Sodium;
4. Deviation of the quantities of the previously mentioned items from the desired ones;
5. Doctor's prescription;
6. Patient's preferences;
7. Cost and delivery time of the food package.

As the doctor or the patient wishes to control the meals on different levels, a fitness function for every level is computed. The evaluation of a solution, which in our case is a meal for a day is done on three different levels and then combined to form the final mark.

The first is the nutrients' and kilocalories' level and deals with evaluating the meals from these points of view. If the number of calories and the quantities of nutrients that they contain are close to the ideal (desired) ones, then this function will have a great value. The ideal values are given are computed based on the user’s profile (gender, age, physical activity etc.) and can be overwritten by the doctor.

The second level deals with food items. It takes into consideration inclusion or exclusion of items to meet the requirements or preferences of the doctor or patient. If some undesired food item exist in a food package, then it is marked lower.

Next, the third level evaluates a solution based on the cost and delivery time of the food package by the food provider.

Finally, the three functions will be combined, in order to give the final mark to the solution. In order to properly define all the mentioned functions, the following aspects are explained first:

* Ideal value
* Weight
* Error function

The following table summarizes what each level deals with.

|  |  |
| --- | --- |
| Level | Evaluation by |
| 1 | Deviation of kilocalories, macro and micro-nutrients from the desired values |
| 2 | Existence/absence of required/restricted or (un)desired food items |
| 3 | Cost and delivery time |

Table 1: Subjects of each level of the fitness function

## The “ideal” functions for components

The evaluation of a component of a solution (e.g. a nutrient or a food item) will require two pieces of information:

1. The real quantity of the component; this is what one solution proposes. For example, if the meal contains 1700 kilocalories, it is said that the “quantity” of the component “kilocalories” is 1700.
2. The desired quantity of the component; this what the quantity of one component is compared to and should be as closest to. This is called the “ideal” value (or quantity) of one component. For example, for a female adult, the desired quantity of kilocalories is 2000, thus, it is said that the “ideal function” of “kilocalories” is 2000.

The solutions that contain components in quantities closer to the “ideal” values are marked higher than the ones that contain components in quantities further from the “ideal” values. The function *error margin* evaluates each component from this point of view (deviation on the real value from the ideal value) and marks it by a number between 0 (smallest) and 1 (greatest). In the following paragraphs, the procedures for computing the ideal values are described.

There are separate procedures to compute the ideal values for nutrients (level 1), food items (level2) and for cost and delivery time (level 3).

### Computation of the ideal values for nutrients

The user has a profile that contains the following pieces of information about him or her that are used for computing the ideal nutritional values:

* Gender;
* Weight;
* Height;
* Age;
* PAF (physical activity factor);
* Optional desire for the user to gain or lose weight.

The ideal value can be either:

* A fixed value, i.e. one single number is considered the only good value
* An interval, i.e. every number in that interval is accepted; in this case the lower and upper limits of the interval are specified.

The following paragraphs present the formulas that are used for computing the ideal values for nutrients from the bachelor’s thesis belonging to Dalma Racz.

#### Kilocalories

For computing the ideal values for kilocalories, the [Basal Metabolic Rate](http://en.wikipedia.org/wiki/Basal_metabolic_rate) and PAF have to be known, as the [Harris-Benedict equation](http://en.wikipedia.org/wiki/Harris%E2%80%93Benedict_equation) states.

|  |  |
| --- | --- |
| English BMR Formula | |
| Men |  |
| Women |  |
| Metric BMR Formula | |
| Men |  |
| Women |  |

Finally,

#### Carbohydrates

|  |  |
| --- | --- |
| Value | Formula |
| Lower daily carbohydrate intake limit |  |
| Upper daily carbohydrate intake limit |  |
| Upper daily carbohydrate intake limit for weight loss |  |

#### Proteins

|  |  |
| --- | --- |
| Value | Formula |
| Lower daily protein intake limit |  |
| Upper daily protein intake limit |  |
| Estimated average daily protein intake value |  |

#### Fats

|  |  |
| --- | --- |
| Value | Formula |
| Lower daily fats intake limit |  |
| Upper daily fats intake limit |  |

#### Vitamin A

|  |  |
| --- | --- |
| Recommended Daily Allowance (RDA)/Adequate Intake (AI) | |
| Male | 900 micrograms (0.9 milligrams) |
| Female | 700 micrograms (0.7 milligrams) |

#### Vitamin B

|  |
| --- |
| Recommended Daily Allowance (RDA)/Adequate Intake (AI) |
| 1.3 milligrams |

#### Vitamin C

|  |  |
| --- | --- |
| Recommended Daily Allowance (RDA)/Adequate Intake (AI) | |
| Male | 90 milligrams |
| Female | 75 milligrams |
| Tolerable Upper Intake Levels (UL) | |
| 2000 milligrams | |

#### Vitamin D

|  |  |
| --- | --- |
| Age (years) | Value (milligrams) |
| < 1 | 0.01 |
| 1 – 70 | 0.015 |
| > 70 | 0.02 |

#### Calcium

|  |  |
| --- | --- |
| Recommended Daily Allowance (RDA)/Adequate Intake (AI) | |
| Age < 50 | 1000 milligrams |
| Age 50 and above | 1200 milligrams |

#### Iron

|  |  |
| --- | --- |
| Age (years) | Value (milligrams) |
| <1 | 11 |
| 1 – 3 | 7 |
| 4 – 8 | 10 |
| 9 – 13 | 8 |
| 14 – 18 | Male: 11; Female: 15 |
| 19 – 50 | Male: 8; Female: 18 |
| > 50 | 8 |

#### Sodium

|  |  |
| --- | --- |
| Recommended Daily Allowance (RDA)/Adequate Intake (AI) | |
| Age < 51 | 2300 milligrams |
| Age 51 and above | 1500 milligrams |

#### Data model for the nutrients

The following image is an extract from the Java class that models the previous information: a mapping from each nutrient to either a fixed value, an interval or both.

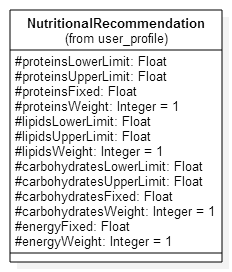


Figure 3: Java class that models ideal nutritional values

In other words, every nutrient is an arithmetic function that depends on the user’s profile.

### Computation of the ideal values for food items

The user has the possibility to specify a list of food items that he or she wishes to be present in his or her recommended menu. Therefore, the user’s profile contains:

* A “like” list – the list of ingredients that the user likes; the ideal value for these is 1;
* A “dislike” list – the list of ingredients that the user does not like; the ideal value for these is 0.

Equation 1 ideal values for food items

An important property of the two sets (like and dislike list) is that they have a void intersection, because the user is able to specify only one from the values (like or dislike):

Equation 2

### Ideal values for cost and delivery time

The cost and delivery time is specified by the user and they default to 0 in order to decrease the cost and time.

## The weights for components

Each component in one solution can be more “important” than other by assigning it a weight greater than the other. The motivation for this is giving priority to the user’s predilections for some component and also to doctor’s prescription.

The weights are level-dependent, i.e. they are meant to classify components within same level. For example, it does not make sense to say that the weight for calcium is greater that the weight of the cheese cake. However, it does make sense that the weight for calcium is greater that the weight of the kilocalories.

### Weights for nutrients

The weights for nutrients are taken into consideration in a hierarchical manner. This is achieved by amplifying the nutrient's requirements coming from a doctor more than those coming from the patient. Nevertheless, there are also default general recommendations for every nutrient, which are the least level on importance.

|  |  |
| --- | --- |
| Source | Weight |
| Doctor’s prescription | 100 |
| Personal preference | 50 |
| Default | 1 |

Table 2: Example of weights

For example, if *n* is a nutrient for which the doctor specified its value, then *w(n)* = 100. The values in this table are subject to be changed, in order to improve the results.

### Weights for food items

The user has the possibility to specify that he or she wishes a specific food item in his or her meal but also *not* to specify anything about certain items. In the former case, to model the fact that the user specified his or her preference, the item is put into the *specified list* and a weight value of 1 is assigned to the corresponding food item whereas in the latter case, when nothing is specified for a certain item, the weight value 0 is assigned.

Equation 3

An important aspect is that the *specified list* is actual the union of the *like list* and the *dislike list*, because the user can only specify one from the 2 values (like or dislike). In other words, the *like list* and *dislike list* form a partition of the *specified list* (see also Equation 4):

Equation 4 The specified list

### Weights for cost and deliver time

The weights for cost and delivery time do not have any significant impact on the fitness function, so they are set to value 1.

## The Error Function

So far, the quantities of a particular nutrient have been deduced, but achieving these exact numbers is practically impossible. There is the need to analyze and penalize the deviation of the real value from the ideal value. Because of the different criteria that the solution is evaluated on, the functions differ in their implementation, but the output is standardized.

Irrespective of the criterion (level) that the error function is applied to, the co-domain is the interval [0, 1], where 1 indicates that the evaluated quantity is the ideal one whereas 0 indicates an unacceptable quantity. Therefore,and if grows, the error function should decrease.

### Level 1 Error Function

The level 1 Error Function analyses one single nutrient, more specifically, a mark based on the deviation of the real value from the ideal one. As mentioned in the previous chapters, it must be present in the meal in quantities close to the ideal ones (see 3.1 Computation of the ideal values for nutrients). There are two possibilities to store the ideal value, either as a single fixed value or as an interval. For each of these, different but similar error functions will be used.

#### Fixed value

The Gauss function

Equation 5 The Gauss function

meets the previously mentioned requirements and could be used as an excellent error function for this particular purpose. The b parameter is our ideal value for a specific nutrient and the final function is the following:

Equation 6 the error function for fixed value

C is an additional parameter with the default value 1.

Equation 7 default value for c is 1

For example, if for some nutrient the ideal value is 5 mg per day, the error function is the following

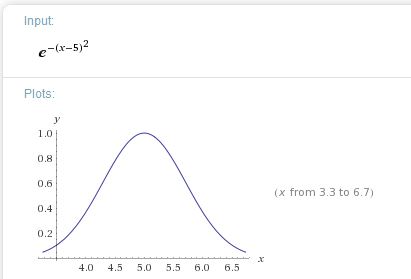


Figure 4 Error function for ideal value 5

In any case,, and, as seen in Figure 2 Error function for ideal value 5, the more distant x from the ideal value is, the lower value the function E takes.

#### Interval

The ideal values can belong to an interval, in this case the previous function is adapted to have the value 1 in that interval. For example, the carbohydrates can range from ~200 to ~400 grams, so any value in that interval must be evaluated to 1 (highest).

Suppose that the ideal interval is [a, b], then:

Figure 5 Error margin if the ideal value belongs to an interval

For example, if a = 5 and b = 10, then the plot of the function is in the following figure.

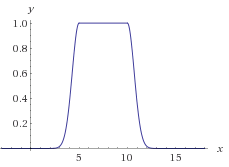


Figure 6 Example plot of the previous function for a = 5 and b = 10

#### Scaling the values

The Gauss function takes into consideration the absolute difference between the real and the ideal value whereas the correct criterion is the ratio between the two.

For example, let us take a difference of 2:

It is obvious, that a difference of 2 is much worse in the case of vitamin A as in the case of calories. To avoid this issue, a small adjustment has to be made; the values are scaled to a standard ideal value.

Equation 8 Mapping from unscaled to scaled values

Thus, in principle, the final mark is given by:

For the previous example, the values are scaled in the following manner:

* Let ;

Now, the evaluations seem to be much more valid than in the previous case, the calories have a good mark whereas the vitamin A has a very weak one.

### Level 2 Error Function

The level 2 Error Function evaluates a food item for which the preference of being present or absent in the meal has been specified. In principle, it is the same as the level 1 fitness function, but instead of analyzing nutrients, it analyzes real food items and assesses a fitness value based on the presence (or absence) of ingredients that are on the preference list of the patient in menu.

In this case, the set *C* of components is the set *F* of food items present the solution meal union the *specified list* (see Equation 6). There four possible cases:

1. The user does specifically not want a food item (ideal = 0) and the item is not present in the meal (real = 0). This is marked with 1, because it is as desired.
2. The user wants a food item (ideal = 1) and the item is not present (real = 0). This is marked with a number, because it is not desired to be marked with 1, but also not very critical to be marked with 0. This particular use case is of a lower importance. For example, is a good approximation of “how important” it is.
3. The user does specifically not want a food item (ideal = 0) and the item is present in the meal (real = 1). This is marked with 0, because the meal should avoid that item.
4. The user wants a food item (ideal = 1) and the item is present in the meal (real = 1). This is marked with 1 because it is as desired.

|  |  |  |
| --- | --- | --- |
| Real | Ideal | E(real, ideal) |
| 0 | 0 | 1 |
| 0 | 1 |  |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Table 3 Equation 9 The level 2 Error function

### Level 3 Error Function

The level 3 Error Function evaluates a food package from the cost’s and delivery time’s points of view. In this case, a real quantity less than the ideal one is a positive aspect (e.g. a food package that costs 10$ less than expected), and the error function takes the value 1. If the real quantity is greater than the ideal one, the Gauss function from Equation 8 is applied.

However, at this point, the additional constant *c* (see Equation 9) for the Gauss function is used to decrease the value more dramatically if the real value exceeds the ideal one. In other words, the constant *c* controls the slope of the curve: a smaller value means a more dramatic change.

The following two illustrations show the influence of the constant *c*.

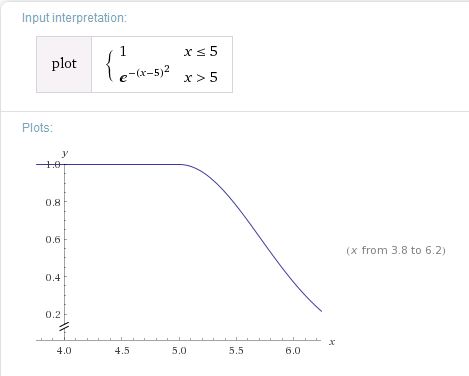


Figure 7Level 3 Error function with c = 1

It can be seen that the slope is not dramatic.

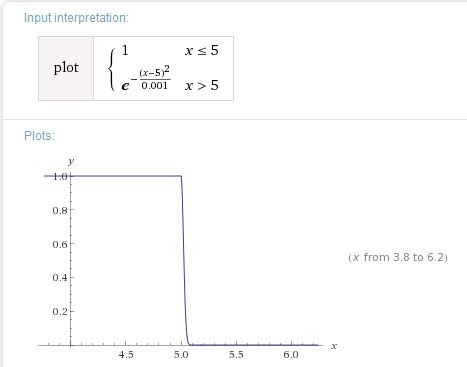


Figure 8 Level 3 Error function with c = 0.001

Hence, the level 3 Error function is:

Equation 10 Level 3 Error function

## The fitness function template

Because the fitness value of a solution depends on more than one criterion, i.e. on different levels (see Table 1: Subjects of each level of the fitness function), the fitness function template will be applied for each criterion in particular; there are 3 criteria (levels) hence applied 3 times and then combined. The template function consists of a weighted average between all the components of a level. The ideal functions and weights for each level have been described in the previous chapters (3. The “ideal” functions for components and 4. The weights for components) and are applied in this chapter.

### Definition

Let *c* be a component from the set *C* of components, *w(c)* the weight of component *c*, real*(c)* the quantity of *c*, *ideal(c)* the ideal quantity of *c*, *E(q(c), ideal(c))* the error function which penalizes any deviation of *q(c)* from the ideal value of *c* and *X* the set of solutions, then:

Equation 11 The fitness function template

Since, because is basically just a weighted average.

### Level 1 fitness function: nutrients

The level 1 fitness function takes a solution as input and provides a value between 0 and 1 to evaluate it from the nutrients’ point of view. It assures that each nutrient comes in the appropriate quantity. The previously mentioned set *C* of components is in this case the set of nutrients, therefore if *N* is the set of nutrients, then:

Equation 12 Fitness function level 1

Where the function *w* is described in chapter 3.1, the function *ideal* in chapter 4.1 and in chapter 5.1.

The generated meals have to contain quantities of nutrients as close as possible to the desired ones, otherwise the evaluation of the bad nutrients will be marked lower, and as consequence of the weighted average, the whole fitness to be marked lower.

Two examples are explained below:

1. A meal containing twice the recommended fat will be scored lower because:

* function E(x, 2x) will have a low value

2. A meal containing sugar for a diabetic patient will also be scored very low because:

* the weight for sugar will be very high for diabetic patients –500;
* the ideal value for sugar is very low (0.001 g) which causes a high level of sugar in the meal the evaluation (the E function) of the sugar level to be very low;
* the weight of the sugar level will highly affect the weighted average in a negative manner;

### Level 2 fitness function: ingredients

The level 2 fitness function takes a solution as input and provides a value between 0 and 1 to evaluate it from the ingredients’ point of view. It checks if each ingredient in the *like* *list* is present in the meal and each ingredient in the *dislike list* is not present in the meal (see Equation 6 The specified list). In this case, the set *C* of components is the set *F* of food items present in the solution meal union the *specified list* (see Equation 6).

Equation 13 Fitness function level 2

### Level 3 Fitness Function: Cost and Delivery time

The Level 3 Fitness Function takes into consideration only the two aspects: cost and delivery time. Thus, the set of components.

Equation 14 Fitness function level 3

### The top level fitness function

The top level fitness function is the one that gives the final mark to a solution by combining the previously presented smaller-level fitness functions. The fitness function template is here also applied, therefore, it is just a weighted average between level 1, 2 and 3 fitness functions.

# Hybridization points

Honey Bee Mating Optimization is a meta-heuristic that uses some heuristics at different points in its algorithm. The points where different heuristics can be applied are called *Hybridization Points*. There are two such points:

* The brood creation strategy
* The worker modification strategy

## Brood creation strategies

A brood creation process takes place between a queen and a drone and will generate a set of broods. More precisely, the procedure takes two solutions as input and returns a set of solutions.

Procedure Create-Broods(queen, drone) returns Set of broods

Set\_of\_broods := Broods-Creation-Strategy(queen, drone)

Return Set\_of\_broods

Algorithm 1 High level procedure for creating broods

Two strategies are implemented in the algorithm:

1. Crossover-based
2. Path-Relinking-based

### The Crossover-based broods creation strategy

The Crossover-based broods creation strategy applies the crossover operation between the queen and the drone at randomly selected crossover points. A crossover point may be one of the main components of the solution: breakfast, lunch, dinner, snack 1 or snack 2. Thus, a new solution (brood) is created by randomly selecting components between the queen and the drone. The process is repeated multiple times to provide a set of broods.

proceudure Crossover-Based-Broods-Creation-Strategy (queen, drone): returns Set of broods

Broods = ∅

Repeat numberOfBroods times

brood =new solution;

foreach meal in {Breakfast, Lunch, Dinner, Snack1, Snack2}

if( newRandomBoolean() == true)

brood.setMeal(meal, queen.meal)

else

brood.setMeal(meal, drone.meal)

Broods.add(brood)

return Broods

Algorithm 2 Crossover-based Broods Creation Strategy

The following image illustrates the process: the green arrows are the randomly selected crossover points, therefore, the lunch and the dinner have been interchanged the resulting “queen”, which is now a brood, is returned.

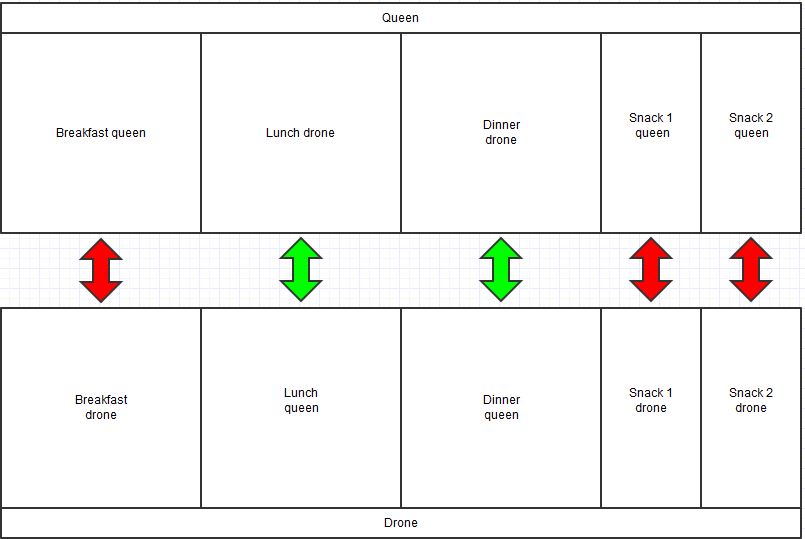


Figure 9 Illustration of Crossover-based broods creation strategy: red arrows are randomly selected crossover points

### Path-Relinking-based broods creation strategy

Path-Relinking is a strategy that generates solutions by iteratively combing a single component of a bad solution with the one of a good solution until a better one is found, or until all components of the bad solution have been replaced.

In our case, a drone firstly takes one random component (breakfast or lunch etc.) from the queen and gives birth to a brood. If the brood is not better than the queen, the new brood takes further a random component from the queen. This process is repeated until a solution at least as good as the queen is found. All the intermediary broods are stored in a set and returned by the procedure.

proceudure Path-Relinking-Broods-Creation-Strategy (queen, drone): returns set of broods

Broods = ∅

brood = drone

while brood.fitness < queen.fitness do

brood.replaceRandomMeal(queen)

Broods.add(brood)

return Broods

Algorithm 3 Path-Relinking Broods Creation Strategy

The following image exemplifies how Path Relinking *may* find a better solution given two solutions A and B. Suppose that the lower (on OY) a node in this plot is, the higher it is marked. There are two paths shown from A from B. The dotted path contains an intermediary node that is better than both A and B.

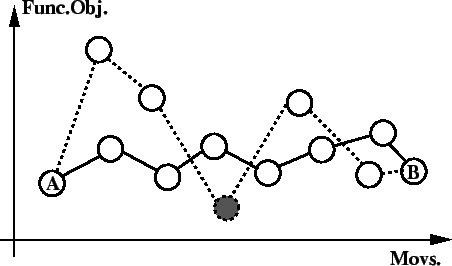


Figure 10 Illustration of Path-Relinking

## The worker modification strategy

Honey Bee Mating Optimization applies different heuristics on broods to improve them. There are three local search heuristics implemented that improve a solution:

1. Hill Climbing
2. Simulated Annealing
3. Tabu Search

Because there are no data dependencies between the broods, the brood improvement process is done in parallel by a certain number of threads. The set of broods is partitioned into the number of threads and each thread executes in parallel the brood improvement processes on its own set of broods. For each brood, a random heuristic is chosen to improve it.

proceudure Improve-Broods-Parallel(Broods): returns set of broods

Broods = ∅

broodsPartionSet = Partition(broods, numberOfThreads)

foreach broodsSet in broodsPartitionSet do

t = new Thread

broodsSet = t.Improve-broods(broodsSet)

Broods ∪= broodsSet

return Broods

procedure Improve-Broods(Broods): returns a set of broods

foreach brood in Broods

heuristic = getRandomHeuristic()

brood = heuristic(brood)

return Broods

procedure getRandomHeuristic(): returns an heuristic

static heuristicsArray[]={HillClimbing, SimulatedAnnealing, Tabu Search}

index=getRandomInteger() modulo sizeof(heuristicsArray)

return heuristicsArray[index]

Algorithm 4 High level procedure for worker modification strategies

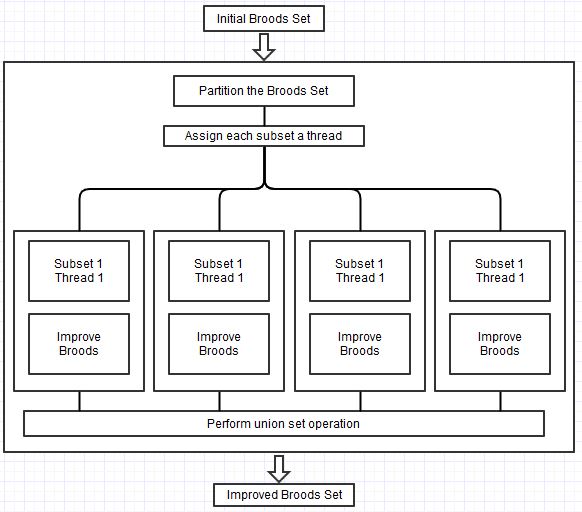


Figure 11Illustration of Parallelizing the Brood Improvement Process

### The Random Mutation algorithm

In order to improve the broods, mutation operations are applied.

A mutation is the replacement of a meal of the solution with another meal (of same type, of course).

A random mutation the replacement of a random meal (one of breakfast, lunch, dinner, snack 1 or snack 2) with a random meal from the available set of meals.

procedure randomMutation(brood): returns a new brood

brood = copy(brood)

mealType = getRandomMealType() // one of Breakfast, Lunch, Dinner, Snack 1 or Snack 2

newMeal = getRandomMeal(mealType) // returns an instance of a meal

brood.replaceMeal(mealType, meal)

return brood

Algorithm 5 Random mutation

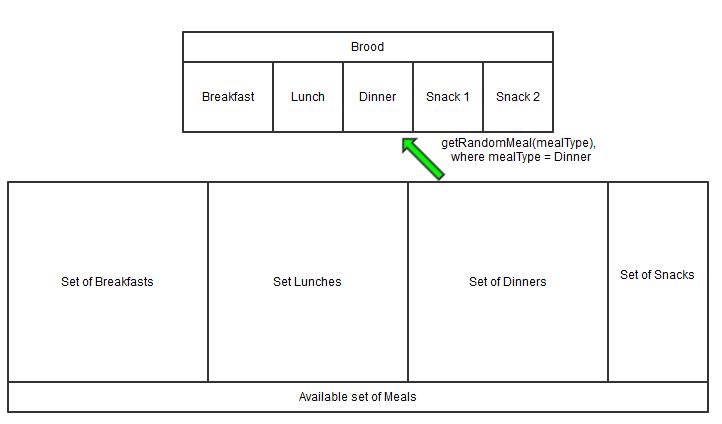


Figure 12 Random Mutation

### Neighborhood of a solution

The heuristics that are used for improving a solution are local search heuristics, i.e. they operate on the *neighborhood* of a solution.

The neighborhood of a solution *s* is a set of solutions resulted by applying multiple random mutations on *s*.

proceudre getNeighborhood(brood, size) : returns a set of broods

neighborhood = ∅

repeat size times

newBrood = randomMutation(brood)

neighborhood ∪= newBrood

return neighborhood

Algorithm 6 Neighborhood of a brood

### Hill Climbing

The Hill Climbing heuristic searches for better solutions in the neighborhood of the current best solution. If one is found, it replaces the current best solution. The solutions that are worse than the current one are ignored, as in every greedy approach. This step is repeated for a certain number of times or until a local maximum is found.

Procedure hillClimbing(brood): returns a brood

local: timeStep=0, localMaximumFound=false, bestBrood = brood

local: limit, neighborhoodSize

while(timeStep < timeLimit and localMaximumFound == false)

timeStep += 1

localMaximumFound = true

N = getHeighborhood(bestBrood)

for each brood in N

fitness(brood) > fitness(bestBrood)

bestBrood = brood

localMaximumFound = false

return bestBrood

### Simulated Annealing

The Simulated Annealing heuristic is similar to the Hill Climbing, but instead of accepting only better solutions, it accepts also worse solutions with probabilities based on the current timestamp, from which the temperature is deduced, and the fitness values of the solutions. This is also known as the Annealing Scheduler. When the temperature is high (at the beginning), the worse solutions are accepted more easily than at lower temperatures.

#### The Annealing Scheduler

The Annealing Scheduler controls the temperature and the number of evaluations at the given temperature.

Basically, the temperature is a function of time. “A geometric cooling schedule essentially decreases the temperature by a cooling factor so that is replaced by or

,

Equation 15 The Annealing Schedule for computing the current Temperature

where is the final temperature. After a certain number of iterations *limit*, the temperature is 0.

The probability to accept a worse solution is:

,

Equation 16 Probability to accept a worse solution

Where *T* is the current temperature and is the difference between the fitness values of the current best solution and the new candidate.

Finally, the solution is accepted if the previously computed probability is greater than a random number .

The number of evaluations at a certain timestamp also depends on the current temperature. At the beginning, a greater number of neighbors are visited whereas towards the end, this number decreases.

procedure getTemperature(timeStep) : returns temperature (float)

return

procedure getNumberOfEvaluations(timeStep) : returns a int

return

Algorithm 7 The annealing scheduler

#### The Simulated Annealing Heuristic

procedure simulatedAnnealing(brood): returns a new brood

local timeStep = 0, temp = getTemperature(timeStep), bestBrood = brood

while(temp >= 0)

timeStep++

temp = getTemperature(timeStep)

neighborhoodSize = getNumberOfEvaluations(timeStep)

N = getHeighborhood(bestBrood, neighborhoodSize)

for each brood in N

delta=fitness(brood) – fitness(bestBrood)

r = randomNumber() // between 0 and 1

if (delta > 0 || exp(-delta/temp) > r)

bestBrood = newBrood

return bestBrood

### Tabu Search

A simple version of Tabu Search is implemented, i.e. the heuristic uses only the short-term memory, as opposed of more complicated (and efficient) versions that use intermediate and long-term memories.

This heuristic is also similar to Hill Climbing, but it uses a short-term memory (the tabu list) to avoid visiting some solutions that have already been visited. Hill Climbing may visit the same solutions multiple times, thus being less efficient. The best solution from the neighborhood is added to the tabu list. The candidate solutions for the a iteration is the neighborhood minus the tabu list (solutions in the tabu list are not considered candidates).

procedure tabuSearch(brood): returns brood

tabuList = ∅, timeStep = 0, limit = 100

globalBestBrood = brood

currentBrood = brood

maxTabuListSize = 10

while(timeStep < limit)

timeStep++

N = getNeighborhood(currentBrood, size)

candidateList = differenceSet(N, tabuList)

bestCandidate = max(candidateList)

if( fitness(bestCandidate) > fitness(globalBestBrood))

globalBestBrood = bestCandidate

add(tabuList, bestCandidate)

while(size(tabuList) > maxTabuSize)

remove\_first(tabuList)

return globalBestBrood

Algorithm 8 Tabu Serach

# Bibliography

Yang, X.-S. (2010). *Engineering Optimization - An Introdcution with Metaheuristics Applications.* Hoboken, New Jersey: John Wiley & Sons, Inc.