Carleton University Animal Care System (cuACS)

Algorithm Design Document

Team: JSB
Justin Ward
Sabina Bialic
Brandon Ward

Submitted To:

Dr. Christine Laurendeau
COMP 3004 Object-Oriented Software Engineering
School of Computer Science
Carleton University

March-05-2019

Contents

Purpose of the Document 3
Animal Attributes 3
Physical Animal Attributes 3
Non-Physical Animal Attributes 4
Description of the Algorithm 5
Cases to Consider 5
Case 1: “Hit or Miss” 6
Case 2: Client Extremes 6
Case 3: Middle 7
Matching Rules 7
Values and Weights in the Animal-Client Matching Algorithm 10
Special Cases 11
Pseudocode 12

Figures

Figure 1 - ACM Flowchart 12

Tables

Table 1 - Physical Animal Attributes 4
Table 2 - Non-Physical Animal Attributes 5
Table 3 - Animal Attributes, Client Selections, Weights, and Case Association 9

Purpose of the Document

The purpose of the document is to describe the cuACS animal-client matching (ACM) algorithm, and the attributes that it considers when matching clients with animals in the system. The ACM allows for the clients to be matched with an animal in the system that best matches their matching criteria; this matching criteria is entered by each user on their profile. Presented in this document are all the animal attributes that are taken into consideration when finding the most optimal client-animal pairs, which includes physical and non-physical attributes. Each of the attributes are given a weight relative to how important they are when calculating the optimal animal-client match. As well, a detailed description of the algorithm is provided with specific matching rules and all the cases that must be considered when running the ACM algorithm and matching clients with animals; these cases are based off of the animal's attribute. Each case has a case equation, for which a score is calculated based on the weights of the animal's attribute and the values associated with that attribute. In the end, the summation of all the results from each case equation is taken into consideration for one client and all the animals in the system; the animal-client pair that returns the highest score is to be considered the best match. Finally, pseudocode is provided to give an idea regarding how the ACM algorithm will be implemented into the cuACS system to compute the optimal set of animal-client matches.

Animal Attributes

Every animal has different attributes, both physical and non-physical, that must be entered for each animal. In some cases, attributes may be consistent between animal type, however in other cases the staff member must enter these attributes into the system themselves. In the ACM, the values of each animal's attributes are compared to what the client has entered into their matching criteria on their profile for the corresponding attribute.

Physical Animal Attributes

There are several physical attributes for animals that must be entered by a staff member every time an animal is added into the system. All animal physical attributes are presented in table 1, along with the range of values that the attribute accepts.

Table 1 - Physical Animal Attributes

Physical Attribute	Range of Values
Type of Animal	(1) Dog (2) Cat (3) Bird (4) Small Animal
Sex	(1) Male (2) Female
Size	(1) Small (2) Medium (3) Large

Colour	(1) Brown (2) Black (3) Golden (4) Orange (5) White (6) Grey (7) Multi
Age	Any positive integer
Lifespan	Any positive integer

Non-Physical Animal Attributes

There are several non-physical attributes for animals that must be entered by a staff member every time an animal is added into the system. All animal non-physical attributes are presented in table 2, along with the range of values that the attribute accepts.

Table 2 - Non-Physical Animal Attributes

Non-Physical Attribute	Description	Range of Values
Difficulty of Care	How difficult it is, on average, to care for this type of animal	(1) Easy (2) Intermediate (3) Difficult
Affection Needed	How much affection/attention the pet needs from its owner	(1) Little (2) Moderate (3) Lots
Cost of Care	Average cost of caring for the animal, per year	Any positive integer value
Time Commitment	Average hours per day that is required to care for the animal	Any positive float value
Space Needed	How much space is needed to care for the animal	(1) Small space (2) Moderate space (3) Large space
Loudness	How much noise the animal makes	(1) None (2) Little (3) Moderate (4) Lots
Activeness	How much activity the animal needs (ie. if it needs to be played with, if it needs to be taken on walks, etc.)	(1) None (2) Little (3) Moderate (4) Lots
Integration with Dogs	How well the animal interacts	(0) Dog Friendly

	with other Dogs	(1) Not Dog Friendly
Integration with Cats	How well the animal interacts with other Cats.	(0) Cat Friendly (1) Not Cat Friendly
Integration with Children	How well the animal interacts with children	(0) Child Friendly (1) Not Child Friendly
Obedience Level	How obedient the animal is already	(1) Untrained (2) Somewhat Trained (3) Trained
Shedding	How much the animal sheds	(1) None (2) Little (3) Moderate (4) Lots

Description of the Algorithm

The algorithm takes into consideration what the client is looking for in an animal based on the matching information that they have added onto their profile and tries to find an animal in the system that best satisfies their criteria. There are several cases the algorithm considers, based on the animal attribute and the clients matching information that corresponds to that attribute. Each case has a corresponding case equation that takes a specific animal attribute and compares its value with the client's selection for the value, as well as the weight of the specific attribute; the weight signifies how important the attribute is to the matching process. There are certain attributes that weigh a significant amount more than others because they are much more important to consider when creating an optimal animal-client match.

The algorithm goes through all the clients in the system and one by one, computes a matching score for each animal in the system. The score is the summation of all of the results from each case equations for an animal-client pairing, which are all stored in order to later identify an optimal set of matches. After all the clients have been compared to all the animals in the system, the algorithm then looks at all the scores to decide which animal should be paired with which client. There are also several cases when coming up with the optimal set of matches, however generally the animal that has the highest score for a given client is the optimal match.

This section of the document will further explain the cases to consider when first matching clients with animals, as well as the equations associated with each case. As well, the process of finding an optimal set of matches will be further explained, going into details about specific cases for when an animal-client pair is deemed to be an optimal match. Finally, presented in this section is pseudocode pertaining to how the ACM algorithm will be implemented into the cuACS system.

Cases to Consider

There are several cases to consider when matching clients with animals, because sometimes an animal trait may be a "hit or miss" situation, while other times there is room for leeway. Several cases are described below, for which a numerical value is generated to represent the value of each case's equation. Animals with the highest score at the end of the ACM process are deemed to be the

best matches for the client in question; the score refers to the summation of all the values for each case equation for a particular client-animal pairing.

Note: In the various equations, 'Client' and 'Animal' refer to the values of the traits for the client and animal we're looking at matching.

Case 1: "Hit or Miss"

In this case, a range of values from -5 to 5 is generated, representing if the animal is a "hit or miss" with the client's matching criteria, since there are some animal attributes that clients are set on, and if not a hit then the animal can't be considered a possible match for the client. The animal attributes to consider are as follows...

- **Type of Animal:** If a client is set on adopting a dog, for example, they will not want to settle for a bird.
- **Size:** If a client wants a small animal for example, it will be hard to get them agree to a large size animal.
- **Sex:** If a client wants a female pet, they will have a harder time accepting a male pet and vice versa.
- **Integration with Dog:** If the client doesn't need an animal to integrate well with other dogs, the animal must be able to integrate well with dogs.
- **Integration with Cat:** If the client doesn't need an animal to integrate well with other cats, the animal must be able to integrate well with cats.
- **Integration with Children:** If the client needs an animal to integrate well with children, the animal must be able to integrate well with children.

Equation (programmatic) =
```
if (hit && value != 0) {
        return 1*Weight
else if (value != 0){
        return -1*Weight
else {
       return 0;
}
```

Case 2: Client Extremes

In this case, a range of values from -40 to 5 are generated, and the match between a client and an animal is better when client's value for a specific trait is higher than animal's value for that trait. The animal attributes to consider are as follows…

- **Difficulty of Care:** If a client is very experienced with taking care of animals, they will be able to care for an easy animal, or a difficult animal.
- **Cost of Care:** The client knows best how much money they are able to spend to care for an animal each year, so it's best when the average cost of caring for that animal is less than or equal to their budget.
- **Affection Needed:** If a client has lots of affection/attention to give to their pet, they would also be able to care for a pet that requires little affection/attention.
- **Time Commitment:** If a client has lots of free time, they are able to take care of animals with a low time commitment, or animals with very high time commitments.

- **Space Needed:** If a client has lots of space available to care for an animal, they will be able to accommodate animals that require little space, as well as animals that require lots of space.

  Equation = $\frac{Client-Animal}{Client} * Weight$

Case 3: Middle

In this case, a range of values from 0.5 to 5 are generated, and the match between a client and an animal is best when the animal traits match as closely as possible to the client's selection criteria. These traits could be considered as "lifestyle" traits and therefore have some leeway, but not much; it's best to have a client with an animal that matches their lifestyle, and vice-versa. The animal attributes to consider are as follows…

- **Lifespan:** If a client wants to keep an animal for say, 5 years, it's best if they match with an animal whose lifespan is in the same range. An animal who can live up to 20 years would not be a great match because it is too much of a time commitment for the client, however an animal whose lifespan is 7 years would still be considerable for the client.
- **Activeness:** Some animals require lots of activity, and it's best to have a client that can tolerate the same amount. Someone who is extremely active would not necessarily want an animal that does absolutely nothing, but could still care for an animal that does moderate activity.
- **Obedience Level:** Sometimes a client will want an animal that is untrained so that they can spend the time to train it, but would still accept an animal that is somewhat trained.
- **Shedding:** There is not much room to negotiate this trait; some people have allergies and are not able to tolerate much shedding, therefore they may not be willing to have an animal that sheds more than they can handle.
- **Age:** If the client is set on adopting a young animal, the animal's age will need to be younger. If the client is willing to adopt an older pet, it would be better to give them an older pet as they're less likely to find homes.
- **Loudness:** If an animal is very loud and the client lives in an apartment, this would not be a good match. A louder animal should go to a client with a house in a less crowded neighbourhood. Alternatively, quiet animals are perfect for apartments as they won't disturb the very close neighbours.

  Equation = $\frac{Weight}{ABS(Animal-Client)+1}$

Matching Rules

Given the above cases, presented below is a table depicting all the animal attributes and the question in the client's profile that they must answer that corresponds with the animal attribute. For each attribute-client selection pair, there is an assigned weight deeming how important it is for the matching process, as well as a corresponding case in the algorithm; the higher the weight, the more important the attribute is when it comes to matching a client with an animal. This table serves as the matching rules for the algorithm, that is, which animal attribute to match with each of the client's selection in their matching preferences on their profile.

Table 3 - Animal Attributes, Client Selections, Weights, and Case Association

Animal Attribute	Client Selections	Weight	Case
Type of Animal	What type of animal are you looking to adopt? (1) Dog (2) Cat (3) Bird (4) Small Animal	5	1
Sex	What animal sex would you prefer? (1) Male (2) Female	2	1
Size	What size of animal are you looking to adopt? (1) Small (2) Medium (3) Large	4	1
Colour	What is your colour preference for an animal? (1) Brown (2) Black (3) Golden (4) Orange (5) White (6) Grey (7) Multi	1	1
Difficulty of Care	How experienced are you with owning [type of animal]? (1) Inexperienced (2) Moderately Experienced (3) Well Experienced	5	2
Affection Needed	How much affection/attention do you have to give to your pet? (1) Little (2) Moderate (3) Lots	3	2
Cost of Care	How much money are you willing to spend to care for your pet? (1) $200 or less per year (2) $200-$500 per year (3) $500-$750 per year (4)$750-$1000 per year (5)$1000 or more per year	5	2
Time Commitment	How much time do you have to care for your pet, per day? (1) Less than 1 hour (2) 1-2 hours	4	2

	(3) 2-3 hours (4) 3+ hours		
Age	Would you prefer an older or younger pet? (2) Younger (8) Older If (2) : Would you be willing to adopt an older pet? (+4) Yes (-1) No	4	3
Lifespan	How long do you hope to have your pet for? (1) 1-2 years (2) 3-5 years (3) 6-10 years (4) 10+ years	4	3
Space Needed	How much space do you have in your home to accomodate for your pet? (1) Small space (2) Moderate space (3) Large space	5	2
Loudness	How loud can your pet be, without disturbing you or others around you? (2) Noise must be minimal (3) Noise doesn't bother me Do you live in an apartment or a house? (-1) Apartment (+1) House	3	2
Activeness	How active are you? (1) Not at all (2) Somewhat active (3) Moderately active (4) Heavily active	3	3
Integration with Dog	Do you have any dogs? (0) Yes If no, do you want to get a dog later on? If yes ? 0 : 1	5	1
Integration with Cat	Do you have any cats? (0) Yes If no, do you want to get a cat later on? If yes ? 0 : 1	5	1
Integration with Children	Do you have children? (0) Yes If no, do you want to have children? If yes ? 0 : 1	5	1

Obedience Level	How obedient do you want your pet to be? (1) Untrained (2) Somewhat trained (3) Trained	3	3
Shedding	How much shedding can you tolerate? (1) None (2) Little (3) Moderate (4) Lots	4	3

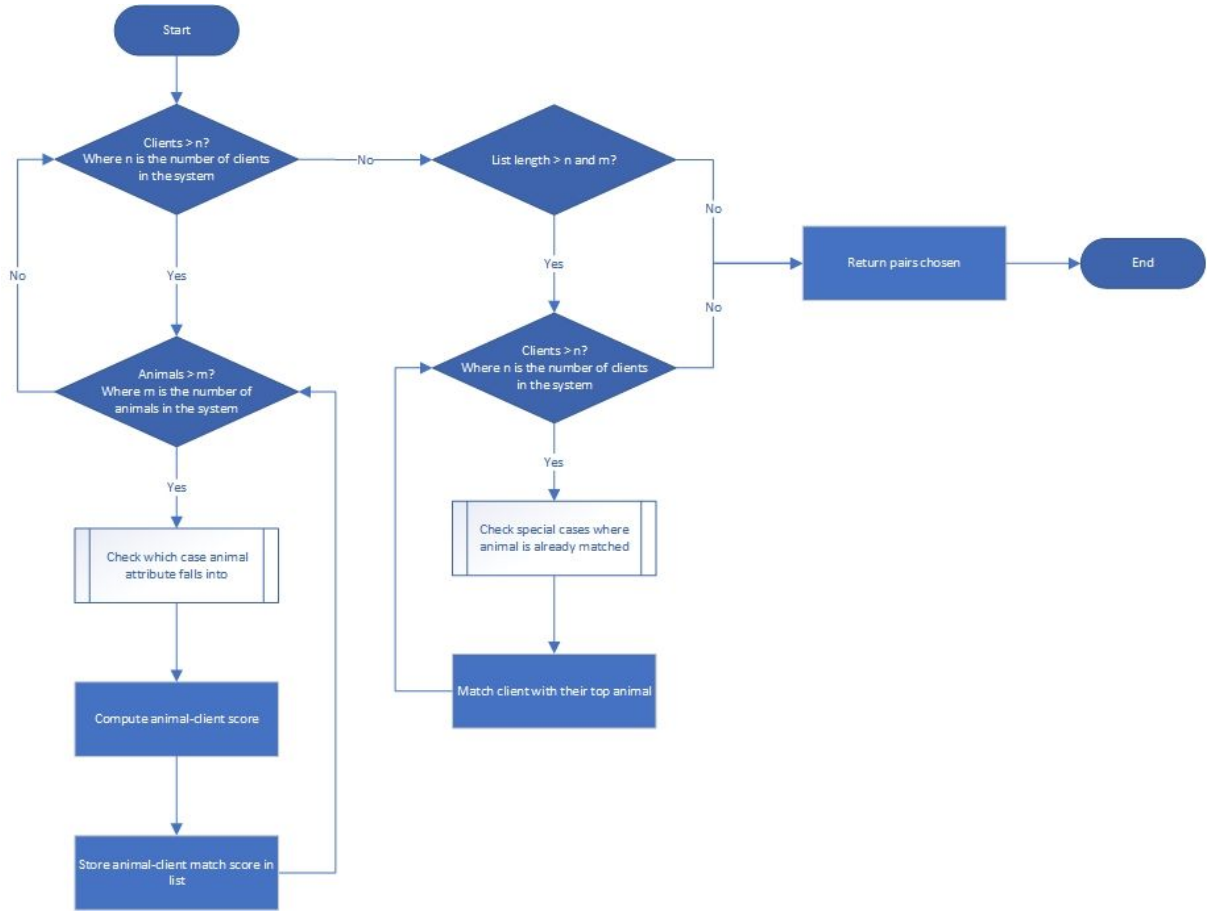
Values and Weights in the Animal-Client Matching Algorithm

The following section highlights the values and weights associated with each animal attribute will be implemented into the ACM algorithm in order to match the shelter's clients with the best possible animal, based on their matching information and the animal's attributes. The weight of each animal attribute is highlighted in table 3, along with the case it falls into in our algorithm; the steps to find the best match given these weights and cases are as follows:

1. The ACM runs on each pair of clients and animals, checking the case and calculating the match from the case and weight.
2. Save the top n animals for every client, where n is the number of clients being matched. If there are more animals than clients, stop at the end of the list of animals.
3. While the length of the list of pairs is less than both the lengths of the client and animal lists, iterate over the clients and match them with their top animal. If their top animal is already chosen, check which client was a better match for the animal and remove the other from the match.
4. Once the pair length is equal to either the number of clients or number of animals, return the pairs chosen.

In order to fully understand the sequence of events that will take place when the ACM algorithm is run, a flowchart illustrating the algorithm can be examined in figure 1, below. The flowchart goes over the steps involved from when the ACM is run, to when the optimal set of matches is computed to serve as a solution model to the problem.

Figure 1 - ACM Flowchart



Special Cases

There are a few special cases to consider in the ACM algorithm to compute an optimal set of matches for animal-client pairs. Essentially, a client will either be matched with one animal for which they are deemed to be a good match, or they will not be matched with any animal depending on the circumstances. The circumstances for this optimal set of matches are as follows:

1. If any client doesn't match well enough with the remaining animals, the algorithm will check this client's top animals and unpair the original match, adding this client to the exhausted list to indicate they have no other options. The client who lost the original pairing will be re-evaluated to find his next best match.
2. If a client doesn't match with any animals at all, then this client will be considered unable to be matched due to such low matching scores.
3. If two clients match with the same animal, the client who matches best (that is, the client who has the highest score associated with that animal) will be given the animal; the other client will be considered unable to be matched with that animal.

Pseudocode

Documented below is pseudocode that has been derived in order to implement the ACM algorithm as described in detail throughout this document.

```
class ACMAlgorithm {
  Map<String, Int> caseDictionary; //Predefined list of all traits and their case numbers
  Map<String, Int> weightDictionary; //Predefined list of all traits and their weights

  runACMOnPair(Animal animal, Client client){
    int pastMatchScore = 0;
    int matchScore = 0;
    for (trait in caseDictionary){
      clientValue = client.getTraits().at(trait));
      animalValue = animal.getTraits().at(trait);
      pastMatchScore = matchScore;
      if (caseDictionary.at(trait) == 1){
        if (clientValue == animalValue && animalValue != 0){
            matchScore += (1 * weightDictionary.at(trait));
        }else if (animalValue != 0){
            matchScore += (-1 * weightDictionary.at(trait));
        }
      }else if (caseDictionary.get(trait) == 2){
        matchScore += (((clientValue-animalValue)/clientValue)*weightDictionary.at(trait))
      }else if (caseDictionary.get(trait) == 3){
        matchScore += weightDictionary.at(trait)/(ABS(clientValue-animalValue)+1)
      }
      if (pastMatchScore >= (matchScore+5)){
        return null;
        break; //If any trait match value is less than or equal to -5, then disregard this match and
                //break the loop
      }
    }
    return matchScore;
  }

  runACM(vector<Client> clients, vector<Animal> animals){
    Map<Client, Map<int, Animal>> clientMatches;
    for (client in clients){
      for (animal in animals){
        int matchScore = runACMOnPair(animal, client);
        if (numKeys in clientMatches[client] <= clients.length()){
          clientMatches[client].insert([matchScore : animal]);
        }else{
          int worst = 50;
          int pop = 0;
```

```
        for (key in clientMatches[client]){
          if (matchScore >= key && pop == 0){
            pop = 1;
            clientMatches[client].insert([matchScore : animal]);
          }
          if (worst >= key){
            worst = key;
          }
        }
        if (pop == 1){
          clientMatches[client].remove(worst);
        }
      }
    }
  }
  Map<Animal, Client> pairs;
  vector<Client> exhausted; //Clients who don't have enough matches
  while ((pairs.length < clients.length && pairs.length < animals.length) || (pairs.length <
(clients.length + exhausted.length) && pairs.length < (animals.length + exhausted.length))){
    for (client in clients){
      if (pairs !contain client){
        for (key in clientMatches[client]){ //need to iterate in sorted order (decrementing){
          if (clientMatches[client][key] != null){
            Animal animal = clientMatches[client][key];
            if (pairs !contains animal){
              pairs.add([animal : client]);
              break;
            }else{
              if (clientMatches[client] > clientMatches[pairs[animal]] && (exhausted !contain
                  pairs[animal])){
                pairs.remove(animal);
                pairs.add([animal : client]);
                break;
              }
            }
          }
        }
      }
      if (clientMatches[client].length < animals.length && clientMatches[client].length <
          Clients.length && pairs !contain client){
          //Check the matches this client does have to see if they have any good animal
          int matchable = 0;
          for (key in clientMatches[client]){
              Animal animal = clientMatches[client][key];
              if (exhausted !contain pair[animal]){
                  pairs.remove(animal);
```

```
                pairs.add([animal : client]);
                 exhausted.add(client);
                 matchable = 1;
            }else{
                  //Check who's score is better
                if (clientMatches[client] > clientMatches[pair[animal]]){
                     pairs.remove(animal);
                    pairs.add([animal : client]);
                     exhausted.add(client);
                      exhausted.remove(pair[animal]);
                     matchable = 1;
                }
            }
        }
        if (matchable == 0){
            //This client does not match well enough to take an animal home
            exhausted.add(client);
        }
      }
    }
  }
  return pairs;
 }
}
```