BEECLUST swarm algorithm: analysis and implementation using a Markov chain model

James Hereford

Department of Engineering and Physics,
Murray State University,
Blackburn 131, Murray, KY 42071, USA
E-mail: jhereford@murraystate.edu

Abstract: In this paper, we analyse a new swarm search algorithm based on the behaviour of social insects, specifically honey bees. The new algorithm, called BEECLUST, is unique because it does not require any bot-bot communication and does not require the bots to know their position. In this paper, we describe the BEECLUST algorithm and model the algorithm using a birth and death Markov chain. From the Markov model, we answer two questions:

1 Will the bots (eventually) congregate or cluster near the targets (maxima) in the search space?
2 How long does it take the bots to cluster?

We corroborate the time-to-cluster analysis with laboratory measurements. The experimental results were done with epuck robots in two different search spaces. In general, the BEECLUST algorithm shows promise for using very simple bots in swarm search applications if conditions during the search allow for bot-bot collisions near the target.

Keywords: swarm algorithms; BEECLUST; Markov chain; swarm robotics; innovative applications.

Reference to this paper should be made as follows: Hereford, J. (2013) 'BEECLUST swarm algorithm: analysis and implementation using a Markov chain model', *Int. J. Innovative Computing and Applications*, Vol. 5, No. 2, pp.115–124.

Biographical notes: James Hereford is an Associate Professor at Murray State University, Murray, KY USA. He received his BS in Electrical Engineering from Stanford University and his MSEE and PhD from Georgia Institute of Technology. His research interests are in the areas of signal processing, evolvable systems and swarm-based systems. He is a member of IEEE, Phi Beta Kappa, and Tau Beta Pi.

1 Introduction

For robotic searches, a swarm of small bots has several advantages over using a single, multi-function robot. For instance, several bots provide for a natural 'fault tolerance'. If one or more of the bots is damaged, there are still other functional bots that can continue to search. In addition, small bots will potentially be easier to transport (lighter), quicker to search (more bots can cover more area) and cheaper to make.

Our goal is to develop efficient search algorithms for a swarm of small, simple bots. We envision that the bots will eventually be about the size of a coin, or smaller, and have a sensor or sensors that 'sniff' out the desired target. There will not be a master bot or global processor that will direct the movements of each bot, but the bots work together to find the target. Each bot determines its own movement and does all of its processing locally.

The swarm search algorithm should be scalable to large numbers of bots, fault tolerant, immune to local optima, and require no central coordinating robot or agent. The particle swarm optimisation (PSO) algorithm (Clerc and Kennedy, 2002; Eberhart and Kennedy, 1995; Parrott and Li, 2006) has proven successful at controlling bots in search type applications (Hereford et al., 2007; Hereford and Siebold, 2008, 2010; Hereford, 2011; Pugh and Martinoli, 2007). In our own PSO-based approach, we let each bot in the swarm be a particle in the PSO. Each bot moves based on the PSO update equations and transmits its location and value when it discovers a neighbourhood best or global best (Hereford et al., 2007). This physically-embedded particle swarm optimisation (pePSO) search was capable of guiding a small group of bots to the brightest light position in the search space (Hereford and Siebold, 2008, 2010).

There are two drawbacks that limit algorithms like the PSO in general bot-based search applications. First, each bot must know its position within the search space. For artificial search environments, beacons for triangulation can be deployed. However, beacons are unavailable or impractical for most real-world searches. Other positioning systems such as GPS are expensive to implement or not accurate enough for many applications such as intra-body searches.

The second drawback is that a PSO-based search method requires bot-bot communication. Even though the communications traffic is minimal it still requires additional hardware for the bots and limits the size of the swarm if all bots must communicate with all other bots.

Because of the position and communication requirements, we investigated a new swarm search algorithm called BEECLUST. The BEECLUST algorithm (Kernbach et al., 2009; Schmickl et al., 2008, 2009) is based on the behaviour of day-old honey bees. In this bee-based algorithm, the bots (or, more generally, agents) make sensor measurements when two or more bots are 'in contact'. After contact, the bots remain stationary for a certain time that is proportional to the measurement value. Bots eventually cluster in areas of the search space that have high fitness/measurement values. Each bot is independent (i.e., not reliant on neighbour bot measurements) but must have contact with the other bots to find the peaks.

The BEECLUST algorithm has several advantages over other swarm-based search techniques. First, no bot-bot communication is required. Thus, there is no concern with communication radius, protocol, or bandwidth. Unlike classic swarm-based techniques, the bots do not have to be arranged in topologies to communicate personal or global best information to any neighbours.

Second, the bots do not have to know their position. If positioning information is available, then position information can be communicated at the end of the search. In addition, the bots do not need to move in a particular direction or along a particular trajectory. During the search, the bot moves randomly except when it contacts another bot, stops, takes a measurement, and waits. At the end of the search, the cluster locations can be determined.

Third, no on-board processing or memory is required – the bot does not even have to do relatively simple update equations like those required of the PSO. The bot moves at random, detects a collision, takes a measurement and does a multiplication. The simple processing shrinks the required hardware of the bot, thereby shrinking the overall bot size and power requirements.

In this paper, we use a birth/death Markov chain to model the BEECLUST algorithm. We use the model to address two questions:

1 Do we expect at least a subset of the bots to congregate near the target (maximum) in the search space?

2 How long does it take for a significant subset of bots to congregate near the target?

The first question provides confidence that the algorithm will eventually work; that is, it will find the target(s) during the search. The second question is important because it gives the user information about how long to wait for a search to end. For example, if there are no targets in the search area, then the user needs to know how long to wait to conclude that the bots are not clustering and thus stop the search. The analysis will also help estimate how many bots are needed for a timely search. In other words, if the swarm of bots is used to search a room for contraband such as a bomb, how many bots are needed so that the search is completed within 5 minutes? 15 minutes? One hour?

This paper is organised as follows: Section 2 gives a detailed explanation of BEECLUST and work related to this paper. Section 3 describes the Markov model that we use to analyse the BEECLUST algorithm and uses the Markov model to determine the expected steady-state distribution of the bots. Section 4 applies the Markov model to the problem of how long it takes for the bots to cluster near the peaks in the search space. Some sample numbers are computed using realistic bot parameters and those numbers are compared with the experimental results in Section 5. Section 6 gives concluding comments. This is the first journal paper with the complete presentation of the Markov model plus the steady-state distribution calculations plus the passage-time calculations. The time-to-cluster, or passage time, calculations are verified with experimental results that are reported here for the first time.

2 BEECLUST algorithm

2.1 Algorithm overview

The BEECLUST algorithm was inspired by the collective behaviour of young honey bees in the presence of a temperature gradient (Kernbach et al., 2009; Schmickl and Hamann, 2011). In a natural bee hive, the central broodnest areas are at a relatively high temperature while honeycomb areas have significantly lower temperatures. The researchers observed that bees move randomly, unless they are in a cluster. When a bee is in a cluster with other bees, it remains in the cluster for an amount of time based on the temperature at that location. The general rule is "The warmer it is, the longer bees stay in a cluster" (Schmickl and Hamann, 2011). In this way, young bees stay in the broodnest area longer, which affects the growth and development of the bees.

Schmickl et al. (Schmickl et al., 2008, 2009; Schmickl and Hamann, 2011) derived a clustering/peak finding algorithm based on the young bees behaviour. Their BEECLUST algorithm has four basic steps:

Step 1 Bots (or software agents) start randomly throughout the search space and then move in a random direction through the search space.

Step 2 If a bot encounters an obstacle or wall, it turns randomly and continues random movement. If a bot intersects or collides with another bot, then it stops.

Step 3 After stopping, the bot measures the 'fitness' or function value at that point in space using a sensor. (The sensor could be a light sensor to measure luminance or a chemical sensor to sniff out a bomb.) It then waits at that point for a prescribed time based on the measurement. The higher the measurement value, the longer the wait time.

Step 4 After the waiting time has elapsed, the bot turns and continues to move at random through the search space (back to Step 2).

For Step 1, it is straightforward to randomly initialise software agents within the search boundaries. For a hardware scheme (where the agents are small bots), a dispersion algorithm (Siebold and Hereford, 2008; Spears et al., 2006) can be used to randomly distribute the bots. For random movement, we pick a direction for each bot and then have the bot move in a straight line in that direction until it encounters an obstacle, boundary or other bot.

In Step 2, bot collisions are detected by determining whether bots are within a certain distance of each other. In software, this is easily done after each time step. In a hardware implementation of the BEECLUST algorithm, one needs to distinguish between collisions with obstacles and collisions with other bots. Collisions with obstacles and walls cause the bot to reorient and move in a new direction; they do not lead to a stop/measure/wait sequence. One way to distinguish between obstacles and other bots is to have the bots signal after a collision, so a bot will know if it has encountered another bot.

Once a bot is stopped, it takes a sensor reading. Since the bots are nearly co-located, the sensor reading will be nearly the same for both bots. The wait time is a multiple of the measured value so the bot(s) will wait longer in areas of high fitness relative to areas with low function values. Other bots may also 'collide' with the stopped and waiting bots but that will not reset the wait times for the stopped bots.

In general, the algorithm will not reach a point where all of the bots are in a cluster. In fact, it is a good idea for some of the bots to be moving to serve as 'scouts' to find new peak locations or respond to changing environmental conditions (Schmickl et al., 2008). At the end of the search, there is a need to identify the clusters and thus the peak locations. For a hardware search, the cluster locations can be determined by a camera, a special-purpose robot, or surveillance by humans. In a software version of BEECLUST, a clustering algorithm such as K-means can be used to determine the location of the clusters.

2.2 Related work

The distinguishing feature of the BEECLUST algorithm over other search scenarios is that the individual bots do not need to know their position in the search space. Each bot only needs to know whether it is next to (has collided with) another bot. This feature potentially makes the search bots very small since no position-determination mechanism is required. It also makes the algorithm useful when the position information is hard or impossible to obtain.

For example, getting position information for each bot in a collapsed building is difficult. Beacons to provide triangulation would be expensive and difficult to install and signals from satellites such as global positioning system (GPS) can be blocked by metal beams and girders in the structure. Or the GPS signal may be degraded due to receiver quality or atmospheric effects.

An example where position information is impossible to get is intracorporeal (within body) searches. New drug delivery and cancer treatment methods target specific body structures for treatment. Having a swarm of bots that can travel through the arteries and deliver the medicine directly to the site could increase treatment effectiveness.

The BEECLUST algorithm was developed by Thomas Schmickl and others in his research group (Bodi et al., 2011; Schmickl et al., 2008, 2009; Schmickl and Hamann, 2011). Schmickl et al. utilise two different methods to model the BEECLUST algorithm: a stock-and-flow compartment model and a space-continuous model. In the compartment model, they track the movement of the bots like movement of material. They partition the search space into four zones or compartments and use flow rates to track the number of bots in each compartment. The flow rates are expressed using ordinary differential equations.

In the space-continuous model each bot is modelled as a Brownian motion particle. The movement of each bot is within the search space is modelled by a partial differential equation (PDE) as is the clustering or aggregation of the bots. Schmickl and Hamann (2011) simulate their models and compare the simulation outputs of bot locations to the bot locations from experimental results. In this paper, we model the BEECLUST algorithm using a Markov model and are able to estimate how long it takes for bots to cluster.

3 BEECLUST model

3.1 Basic model for analysis of BEECLUST algorithm

The intent of our mathematical analysis is to show whether the bots (agents) will cluster or congregate near the peaks of the measurement function even though the individual bots do not have position information. The same model will be used in the next section to answer the question of how long it takes a subset of the bots to cluster together.

The analysis models the search space as a two dimensional grid of individual cells. For simplicity, we assume that the cells are squares though other shapes can be used as long as adjacent cells do not overlap. The bots start at random locations in the search space and then move in random directions. At one cell in the search space, the bots will randomly arrive, which we call a birth. The bots move independently of each other so the time between successive arrivals, or births, is independent of previous arrivals. The time between arrivals thus has an exponential distribution with an average inter-arrival time of $1/\lambda$ (Hereford, 2010).

Since the inter-arrival times are independent, exponential random variables, the arrival (or birth) process can be modelled by a Poisson process (Allen, 1978) with a birth rate of λ. Thus, the probability that M bots arrive in t seconds is given by

$$p(m = M) = \frac{e^{-\lambda t}(\lambda t)^M}{M!}; n \geq 0; t \geq 0 \quad (1)$$

Another way to think of the Poisson process is that the probability of one birth within a short time segment h is given by

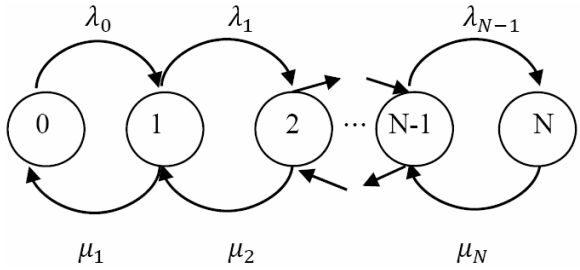
$$\lambda h e^{-\lambda h} = \lambda h + o(h) \tag{2}$$

where $\lim_{h\to 0} \frac{o(h)}{h} = 0.$

Once a bot arrives at a cell it will either pass through the cell or collide with another bot and stay for a prescribed time. Bots that leave the cell reduce the number of bots at that location and are referred to as deaths. The death rate is denoted by μ.

The number of bots at a location can be modelled by the state diagram shown in Figure 1, where the state number corresponds to the number of bots in the cell. Our model assumes that there can be only single births or single deaths at any one moment in time, so the number of bots in the cell can only traverse to adjacent states and the arrival rate can be modelled as a Poisson process. The state diagram stops at N, since there are only N bots in the swarm.

Figure 1 State diagram of number of bots in cell



In the state diagram model of the cell, the state of the cell is the number bots within that cell at a particular time. We assume that the state of neighbouring cells do not affect each other so there is no correlation between cells.

Since there can only be single births or deaths, the future state is based only on the present state of the cell. Therefore, the cell can be modelled as a birth/death Markov chain (Allen, 1978, 2003). A Markov chain is a stochastic process where the future of the process is only dependent upon the present state and the states are discrete.

To determine how many bots will be in a cell, we define the probability that there are m bots in the cell. Let $p_0 = \mathrm{P}[m = 0]$ be the probability that there are zero bots in the cell. Likewise, $p_1 = \mathrm{P}[m = 1]$ and, in general,

$$p_n = \mathrm{P}[m = n] \tag{3}$$

The probabilities must sum to 1 for each cell, so

$$\sum_{n=0}^{N} p_n = p_0 + p_1 + p_2 + \cdots + p_N = 1 \tag{4}$$

The probabilities, p_n, can be written as a function of the birth and death rates, λ and μ. However, the birth and death rates are a function of the state number which makes finding the p_n for all time and state number difficult.

3.2 Steady state distribution

In this section, we want to determine the expected final bot distribution, so we will look at the steady state relationship of the birth/death chain. In steady state, the average outgoing rate from the state will equal the average ingoing rate; that is, that the $p_n(t)$ approach a constant value for all n as $t \to \infty$.

Consider the transition between state 0 and state 1. During steady state, the average rate out of state 0 ($\lambda_0 p_0$) must equal the average rate into state 1 ($\mu_1 p_1$). This leads to the balance equation $\lambda_0 p_0 = \mu_1 p_1$, so

$$p_1 = \frac{\lambda_0}{\mu_1} p_0. \tag{5}$$

Repeating the balance equation for each state transition leads to

$$\lambda_{n-1} p_{n-1} = \mu_n p_n \Rightarrow p_n = \frac{\lambda_{n-1}}{\mu_n} p_{n-1}. \tag{6}$$

Recursively substituting for successive p_{n-1} allows us to write p_n in terms of p_o:

$$p_n = \frac{\lambda_{n-1} \cdot \lambda_{n-2} \cdots \lambda_0}{\mu_n \cdot \mu_{n-1} \cdots \mu_1} p_0 \tag{7}$$

Inserting the p_n probabilities into (4) yields

$$p_0 \left(1 + \frac{\lambda_0}{\mu_1} + \frac{\lambda_1 \lambda_0}{\mu_2 \mu_1} + \cdots + \frac{\lambda_{N-1} \lambda_{N-2} \cdots \lambda_0}{\mu_N \mu_{N-1} \cdots \mu_1} \right) = 1. \tag{8}$$

The next step is to determine the birth and death rates for each state n. Once the λ_i and μ_i are determined, the previous equation can be solved for p_0 and then all the p_n can be found from equation (7).

The bots arrive at each cell in a Poisson arrival pattern. If there are no bots in the cell ($m = 0$), then potentially any of the bots can arrive at the cell. Since each bot must be in one of the cells, we know that

λ_0 = (total number of bots) / (number of cells in space).

The number of cells in the search space is

$$\text{Number of cells} = (\text{size of space}) / \text{size of cell} = S / \delta \tag{9}$$

where S is the size of the search space and δ is the area of a cell. The cell size will generally be the area that a single bot can scan and detect collisions.

The birth rate when there are no bots in the cell ($n = 0$) is thus

$$\lambda_0 = \frac{N}{S/\delta} = \frac{\delta \cdot N}{S} \tag{10}$$

If there is one bot in the cell, then there are only $(N - 1)$ remaining bots that can enter that cell. That reduces the arrival rate to $(N - 1)$ / (number of cells). By extrapolation, if there are k bots clustered together in the same cell, the arrival rate will be reduced by k, so

$$\lambda_k = \frac{\delta}{S}(N-k) \tag{11}$$

Since the arrival rate is a non-constant value, the Markov chain is a non-homogeneous process.

There are two death rates. The first death rate, μ_1, is when the bot is alone in the cell and has not collided with another bot. For modelling purposes, we will assume $\mu_1 = \beta$, where β is a constant. β is based on the speed of the bots because it is inversely related to how fast a bot travels through the cell.

Once a bot collides with another bot, the bot will wait a prescribed amount of time, s. From birth/death queuing theory, the death rate is $\mu = 1 / E[s]$, where $E[s]$ is the expected time in the cell (Allen, 1978). For BEECLUST, the time spent in the cell is the same for all bots that have collided so $E[s] = s$, which is dependent upon the measured or sensed value at the cell location. We can apply a scaling or weight function to the measured value so we set $s = g(f(x, y))$, where $f(x, y)$ is the measured value at location (x, y) and $g()$ is the scaling applied to the measured value.

If there are two or more bots in a cell, the death rate is given by

$$\mu = \frac{1}{E[s]} = \frac{1}{s} = \frac{1}{g\left(f(x, y)\right)}. \tag{12}$$

Letting $\mu_2 = \mu_3 = \ldots = \mu_N = \mu = 1 / g(f(x, y))$ and plugging into (8) yields

$$p_0\left(1+\frac{\frac{\delta N}{S}}{\beta}+\frac{\frac{\delta N}{S}}{\beta}\left(\frac{\frac{\delta}{S}(N-1)}{\frac{1}{g\left(f(x, y)\right)}}+\frac{\frac{\delta}{S}(N-1)(N-2)}{\left(\frac{1}{g\left(f(x, y)\right)}\right)^2}+\cdots+\frac{\frac{\delta}{S}(N-1)(N-2)\cdots 1}{\left(\frac{1}{g\left(f(x, y)\right)}\right)^{N-1}}\right)\right) \tag{13}$$

The parameters δ, N, S, β, $g()$, and $f(x, y)$ are determined by the search scenario (e.g., the number of bots, the weight function (g), speed of bots) and the search space characteristics (e.g., size of the search space, distribution of contaminant, cell size). Once the search characteristics are known, equation (13) can be solved for p_0, then p_0 is used to solved for the other probabilities p_n using (7).

Once the steady state probabilities are known, one can calculate the expected number of bots in the cell during steady state, L.

$$\begin{aligned} L &= \text{expected number bots in cell} = E[m] \\ &= \sum_{n=0}^{N} n \cdot p_n. \end{aligned} \tag{14}$$

For the simulations in Section 3.3, we calculated L for each cell in the search space and then normalised the results for each cell so that the total number of bots in all of the cells was N.

3.3 Visualisation

We applied the Markov steady state analysis from the previous section to two 2D functions: the Himmelblau function and the Rastrigin function. We modified both functions so that:

a the goal is to find the maxima, not the minima

b the maximum function value in the search range is 1

c the minimum function value in the search range is 0.

The Himmelblau function is given by

$$f(x, y) = \left(200 - \left(x^2 + y - 11\right)^2 - \left(x + y^2 - 7\right)^2\right) \Big/ 200 \tag{15}$$

within the range of $-6 \le (x, y) \le +6$. It has four global optima within the search range. The Rastrigin function is given by

$$\begin{aligned} f(x, y) = -\big(&x^2 - 10\cos(\pi / 2x + 10) + y^2 \\ &-10\cos(\pi / 2y + 10)\big) \end{aligned} \tag{16}$$

within the range of $-5.12 \le (x, y) \le +5.12$. It has nine optima though only one is a true global maximum. For the initial simulations, we used the following parameter values:

N number of bots = 60

δ size of cell = 0.4 × 0.4

S search space area = 12 × 12 (Himmelblau), 10.24 × 10.24 (Rastrigin)

$g()$ sigmond function with weight 8

$$g\left(f(x, y)\right) = \frac{8 * f(x, y)^2}{f(x, y)^2 + 1}$$

β 'death rate' for state 1 = $\mu_1 = 4$.

We investigated different weight functions (g), which affect the death rate μ for states 2, 3, 4, etc. Schmickl et al. (2009) use the sigmoid weight function to determine the waiting time for the bots, so we slightly modified their function to fit with our scaled 2D functions ($f_{\min} = 0, f_{\max} = 1$). The wait factor of 8 ensures that bots near peaks in the search space wait for a relatively long time. Note that for the numerical simulations (results shown below in Figures 2 and 3), death rates for individual bots are not calculated. Instead, the death rates for the different states are used in equations (7) and (13) to calculate the steady state probability for how many bots are in each cell.

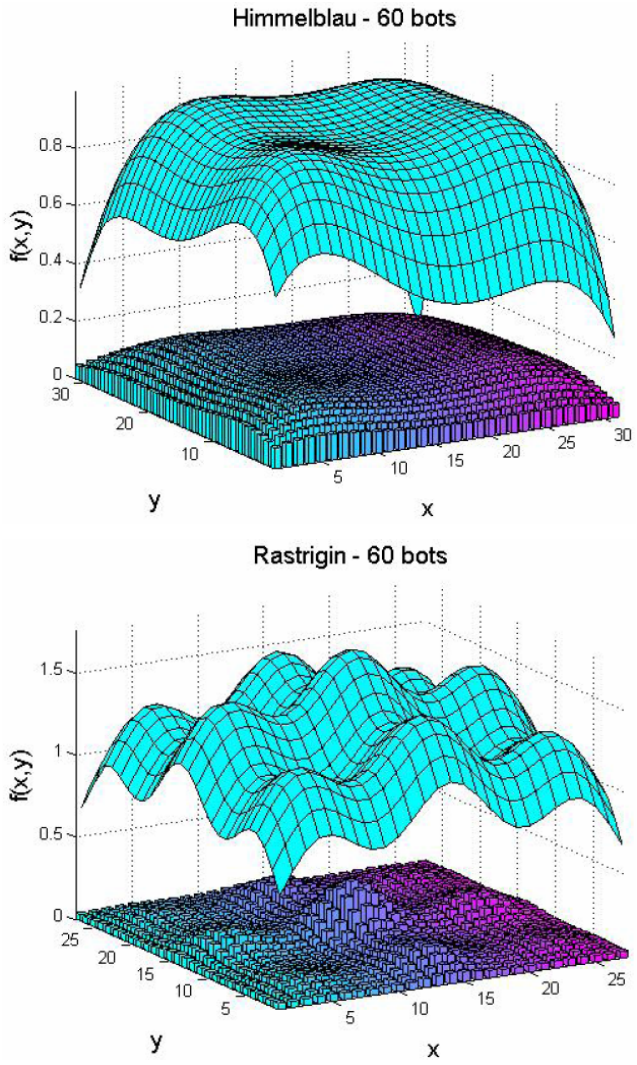
We show the Markov analysis results for searches assuming 60 bots for the Himmelblau and Rastrigin functions in Figure 2. The top part of the figures show $f(x, y)$ with the plot of the Rastrigin function raised by 0.5 to avoid obscuring the 3D bar plot. The bottom part of the figures show a 3D bar plot of the expected number of bots in each cell.

In the plots in Figure 2, there is no clear bunching of the bots near the peaks. In the Himmelblau function especially, the estimated distribution of bots (bar plot at the bottom) is relatively flat, which means the bots are not clustering near

the peaks. With the Rastrigin function, one cluster of bots has formed but the side peaks are indistinct from the background distribution.

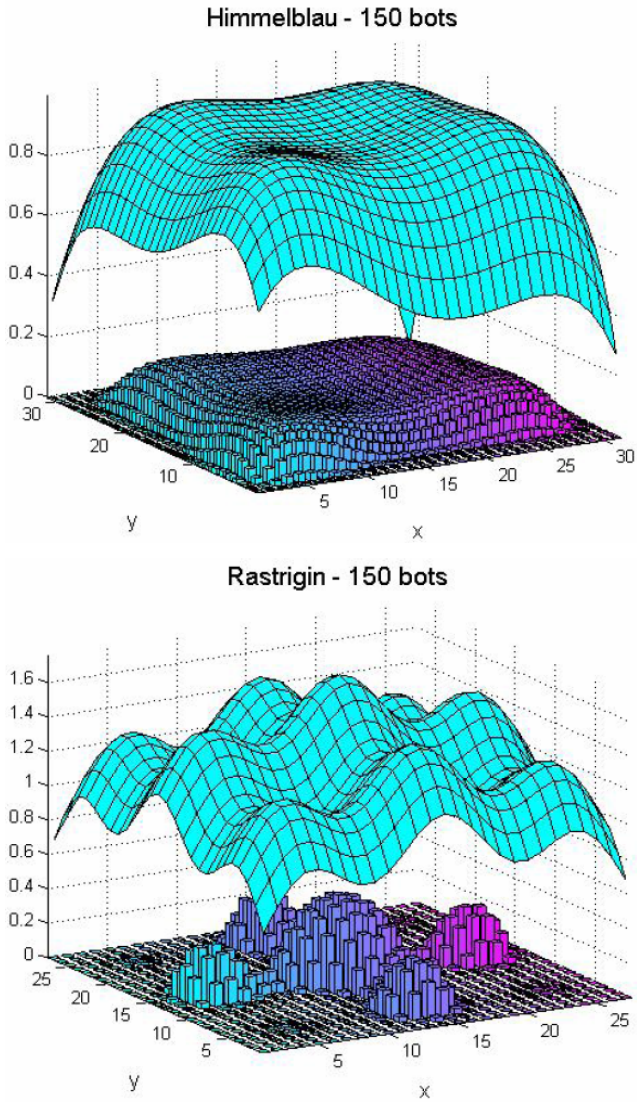
To improve the BEECLUST results, we need to improve the birth rates. One way to increase the birth rate λ is to increase the size of the cell, δ. However, we assume that a bot in the cell can detect another bot in the cell so the maximum cell size is limited by the detection range of the bots.

Figure 2 Himmelblau and Rastrigin functions with 60 bots in search and the expected number of bots in each cell based on steady state Markov chain analysis shown as 3D bar chart below (see online version for colours)



Another way to increase the birth rate is to increase N, the number of bots in the search. Figure 3 shows the results for $N = 150$. There is now clearer variation in the distribution of bots with the Himmelblau function and discernible clusters of bots with the Rastrigin function. In the patch of cells underneath the Rastrigin maxima, there is an expected cluster of 40 bots (out of 150) compared to an expected cluster of only 6 bots (out of 60) for the previous results.

Figure 3 Rastrigin function with 150 bots in search and the expected number of bots in each cell shown as 3D bar chart below (see online version for colours)



Increasing the birth rate by increasing the number of bots, N, leads to more collisions and larger relative clusters of bots near the peaks. As mentioned previously, increasing the bot detection radius and hence the cell size will also increase birth rates and thus increase bot clustering.

4 Timing analysis

4.1 Theory

In this section, we use the Markov model to determine how long it takes for the bots to begin to cluster near the target. From queueing theory, the time for a Markov chain to transverse from one state to another state is called the passage time (Allen, 2003). Suppose that there are a bots in a cell and we want to determine how long it takes till there are b bots in the cell. We will denote the time it takes to go from state a to state b as $T_{b,a}$ and call it the passage time from a to b. For a birth/death process $T_{b,a}$ can be broken down to

$$T_{b,a} = T_{a+1,a} + T_{a+2,a+1} + \cdots + T_{b,b-1} \tag{17}$$

assuming that a is less than b. The mean first passage time for state b is the expected value of $T_{b,a}$ or

$$E\left[T_{b,a}\right] = E\left[T_{a+1,a}\right] + E\left[T_{a+2,a+1}\right] + \ldots + E\left[T_{b,b-1}\right]. \tag{18}$$

Before we derive the mean first passage time for the BEECLUST algorithm, we note that the process will change state (i.e., bot will enter or leave a cell) at a random time. The state change will be either a birth or death. Assuming the inter-event times are independent, they can be modelled by an exponential probability density function (PDF)

$$f_i(t) = \left(\lambda_i + \mu_i\right) e^{-(\lambda_i + \mu_i)t} \tag{19}$$

where

$$E\left[f_i(t)\right] = \frac{1}{\left(\lambda_i + \mu_i\right)} \tag{20}$$

The f_i gives the PDF for the inter-event time when in state i. The expected time to leave state i is the mean of the exponential distribution or $1 / (\lambda_i + \mu_i)$.

Suppose that there are i bots in a cell. After an exponential amount of time the process jumps from i to $i + 1$ with probability $\lambda_i / (\lambda_i + \mu_i)$ and to state $i - 1$ with probability $\mu_i / (\lambda_i + \mu_i)$. To find the expected time to go from i to $i + 1$ we must take into account the process may jump to $i - 1$ then the expected time it takes to go back to i must be added. Thus,

$$\begin{aligned} E\left[T_{i+1,i}\right] = {} & \frac{\lambda_1}{\lambda_i + \mu_i}\left(\frac{1}{\lambda_i + \mu_i}\right) \\ & + \left(\frac{\mu_i}{\lambda_i + \mu_i}\right)\left(\frac{1}{\lambda_i + \mu_i} + E\left[T_{i,i-1}\right] + E\left[T_{i+i,i}\right]\right) \end{aligned} \tag{21}$$

Simplifying equation (21) and rearranging terms (see Hereford, 2011) leads to

$$E\left[T_{i+1,i}\right] = \frac{1}{\lambda_i} + \frac{\mu_i}{\lambda_i} E\left[T_{i,i-1}\right] \tag{22}$$

The mean (or expected) time for the bots to reach state $i + 1$ can be found recursively using equation (22). Since the state number is how many bots are together in a cell, this gives the mean time for bots to congregate or cluster together. For example, the mean time till two bots are in the cell together is

$$\begin{aligned} E\left[T_{2,0}\right] = {} & E\left[T_{1,0}\right] + E\left[T_{2,1}\right] \\ & + \frac{1}{\lambda_0} + \frac{\mu_0}{\lambda_0} E\left[T_{0,-1}\right] + \frac{1}{\lambda_1} + \frac{\mu_1}{\lambda_1} E\left[T_{1,0}\right] \\ = {} & \frac{1}{\lambda_0} + \frac{1}{\lambda_1} + \frac{\mu_1}{\lambda_1 \lambda_0} \end{aligned} \tag{23}$$

The analysis can be extended to any number of bots in a cell. In the next section, we give sample values for the λ and μ parameters.

4.2 Numerical results

The mean passage time results depend on the mean birth rate (λ) and the mean death rate (μ) for the cell. The birth rate and the death rate depend on the characteristics of the individual search: speed of bot, collision radius of bot, size of search space, strength of measurement signal, etc. In this section, we determine parameter values for BEECLUST and then how many bots are needed for a search to be completed in an efficient manner; that is, determine how many bots are needed if one wants the search to end in T minutes.

We want to compute the expected time for a cluster of 5 bots to form ($E[T_{5,0}]$) using e-puck robots (Mondada et al., 2009) to do the search. For this scenario, we use the following parameter values which assume a swarm of bots searching for one target in a room with a strong signal:

δ size of cell = 12 cm × 12 cm

$g()$ weight function = sigmoid function with $W = 8$

μ_1 death rate for state 1 = 2 sec^{-1}.

We assume that the maximum value of $f(x, y)$ will be 1 and the minimum value will be 0; values near 1 represent a strong signal and/or that the target (destination) is close.

The value for μ is determined by the expected death rate. If there is only one bot in the cell, then the death rate is given by how long it takes the bot to travel through the cell. The maximum velocity of the e-puck is about 16 cm/sec. The time that a bot will be in the cell is approximately (cell width)/velocity = 0.75 sec if the epuck travels through the centre. If it clips just a corner of the cell, then the time in the cell will be less. We chose an average time of $t_0 = 0.5$ sec. The death rate will be reciprocal of the time in the cell so

$$\mu_1 = 1 / E\left[t_0\right] = 2 \text{ sec}^{-1}. \tag{24}$$

We calculate the expected number of bots that would be required to find the target for 2 different room sizes: 5 m × 5 m (office or conference room), and 12 m × 12 m (large conference room or storage room). In these scenarios, we assume that at least 5 bots must be near the target for the target to be 'found'. Table 1 shows the minimum number of bots that are required for expected search times of 5 minutes, 15 minutes, and one hour.

Table 1 Number of bots required to form a cluster of 5 bots for three different room sizes

Expected wait time (min)	*Room size*	
	5 m × 5 m	*12 m × 12 m*
5	375	2,157
15	278	1,594
60 (1 hour)	199	1,141

As expected, fewer bots are needed for the slower searches. But the number of bots does not decrease linearly. For instance, if the expected wait time is increased by a factor of three (5 minutes to 15 minutes), the number of bots only decreases by about 25%. Similarly, as the expected search

time increases from 15 minutes to 60 minutes, the required number of bots decreases by about 28%.

The passage-time calculations give an indication of how long to wait for the search to end. This is important when there is no target or object in the search space. The calculated search times, however, are expected values so the actual passage times may be more. Thus, a factor of 5 or 10 should be added to the times before halting the search.

The search times are affected by the speed of the bot and, even more, by the radius that is covered by the bots' collision detection sensor. To decrease the expected search times, one can increase the cell size but that would also require increasing the collision detection range of the bots. In other words, the further that the bots can 'see' then the fewer bots that are needed.

Table 1 also helps to estimate the cost of implementing the BEECLUST algorithm. If we use 15 minutes as our desired search time, then it takes about 280 bots to search a moderate size room and 1,600 bots to search a large room. The bots required to implement the BEECLUST are fairly simple but they would require a chassis, two motors, power source, at least one sensor, collision detector(s) and a simple microcontroller for coordinating the behaviour. No positioning sensor nor communication transceiver are required, but they can be added if desired plus, there would be the logistical burden of storing, transporting and recharging nearly 1,600 bots. That is potentially a large expense for a relatively slow search.

5 Experimental results

To verify the passage time results from the Markov model, we implemented the BEECLUST algorithm using a small swarm of bots. Our swarm was made up of six e-puck robots (Mondada et al., 2009) that searched for the brightest location in the search space. The search space was a square with no obstacles; we used two different sizes for the search space – 1 m^2 and 2 m^2 – to see how the passage time numbers scale. For each size, we computed how long it took for three bots to cluster underneath the bright light. For each trial run, we randomly placed the bots in the search space. The 1 m^2 search space with one random initial configuration of bots is shown in Figure 4.

5.1 Experimental setup

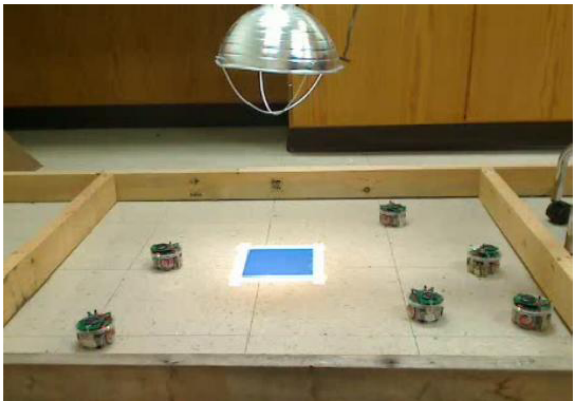
In our laboratory experiments, we used a cell size of δ = 16 cm × 16 cm and N = 6 total bots in the search. Thus, the arrival rate for the smaller search space if zero bots are currently in the cell is $\lambda_0 = 0.15$ and decreases for λ_1, λ_2, etc. See Table 2 for the parameter values for the larger search space.

For the theoretical calculations, we assume that the value for μ_1 is 2 sec^{-1} based on the e-puck speed. If there are two or more bots in the cell, then the death rate will be the reciprocal of the wait time. In our experiments, we set the wait time to be approximately 20 seconds when the bots were in the target cell (near the peak luminance) and approximately 1 second when they were away from the peak luminance. Therefore,

$$\mu_2 = 1 / E[t] = 0.05 \text{ sec}^{-1},$$

for bots near the target. The death rates for 3, 4, or 5 bots in the cell are the same as μ_2.

Figure 4 1 m × 1 m search space with random initial configuration of epuck robots (see online version for colours)



Notes: The blue square denotes the target location of highest luminance. The trial was designated complete when three of the six bots were on the target square.

Table 2 Parameters for experimental results

Parameter	*Description*	*Value – small search space*	*Value – large search space*
S	Size of search space	1 m × 1 m	1.41 m × 1.41 m
δ	Cell size	16 cm × 16 cm	16 cm × 16 cm
N	Number of bots	6	6
$g()$	Wait function	Step function	Step function
λ_0	Birth rate, state 0	0.15 sec^{-1}	0.0769 sec^{-1}
λ_1	Birth rate, state 1	0.125 sec^{-1}	0.0641 sec^{-1}
μ_1	Death rate, state 1	2 sec^{-1}	2 sec^{-1}
μ_2	Death rate, state 2	1/20 sec^{-1} (target area)	1/20 sec^{-1} (target area)

We programmed each bot in the swarm to follow the BEECLUST rules of:

a move in a straight line

b turn randomly if it encounters a wall.

For the experimental results, the weight factor $g()$ was a step function that was tailored to the light sensor on each bot. Each bot would wait for 20 seconds after a bot-bot collision in the bright region and about 1 second after a collision

outside of the bright region. The bright region is approximately 16 cm × 16 cm in the centre of the search space.

Note that the bots would not actually collide but would stop after intersecting another bot that was within its 'collision radius'. The collision radius for the e-puck robot is about 4 cm so the bots would stop before colliding once another bot was within 4 cm. Faster passage times would occur if the collision radius was bigger since then there would be more bot-bot collisions and thus more stop/wait cycles.

The dimensions of the search space and other search parameters are shown in Table 2. From the parameters we calculated numerical values for birth rates (λ) and death rates (μ), which are also shown in Table 2.

In the search space, the light intensity in the peak (target) area was approximately 1,400 lux. In the corners of the search space, the light intensity varied from about 600 to 850 lux.

5.2 Results

Two snapshots from a sample run are shown in Figure 5. Figure 5(a) (top) shows the bot locations after about two minutes of searching and Figure 5(b) (bottom) shows the bot locations after about 5 minutes (end of search). At two minutes there is no discernible clustering of the bots near the target (blue square); there are bot-bot collisions during the search but rarely within the target area.

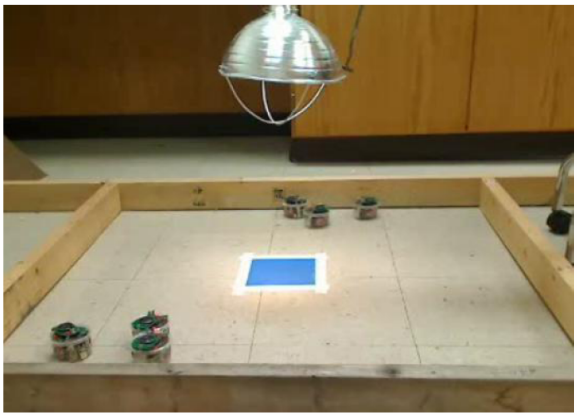
Quantitative results from the lab tests of the BEECLUST algorithm are shown in Table 3. The first row in the table shows the expected time until 3 bots first overlap the centre cell (target cell) for the small (1 m^2) and large (2 m^2) search spaces. The expected times were calculated using the theoretical results from Section 4.

The subsequent rows in Table 3 show the quantitative results from the experimental trials. Table 3 shows the average, standard deviation and 90% confidence interval for ten successful runs using the hardware swarm. We also show the average number of collisions between bots throughout the entire search space and the average number of bot-bot collisions just within the target zone.

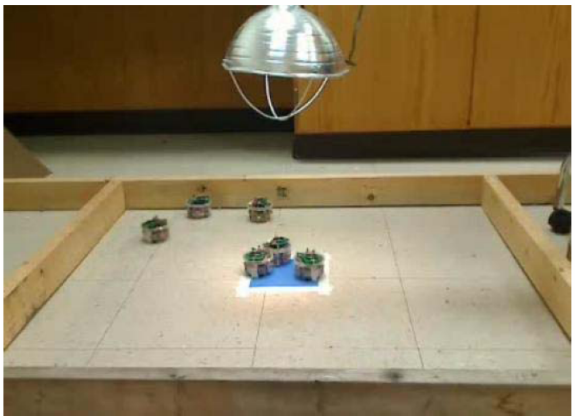
There is very good agreement between the experimental results and the theoretical results. For both the small and large search spaces, the theoretical passage time falls within the 90% confidence interval of the experimental results. This provides confidence that the Markov model is valid for the BEECLUST algorithm.

One thing that the experimental results highlight is the importance of bot-bot collisions within the target area. The last two rows in Table 3 show the average number of collisions between bots for the ten test runs. For both search spaces there is a large standard deviation for the number of collisions in the search space; in other words, there is no correlation with the number of collisions and when the bots will cluster. However, once the bots start to collide in the target area, clustering happens quickly.

Figure 5 1 m × 1 m search space showing location of bots, (a) bot locations after about two minutes of searching (b) three bots within blue target cell thus ending the search after approximately 5 minutes (see online version for colours)



(a)



(b)

Table 3 Results from ten trials each for two different search spaces

Parameter	*Small search space*	*Large search space*
Expected passage time (theory)	189 sec (3 min, 9 sec)	865 sec (14 min, 25 sec)
Experimental average from ten test runs	270 sec (4 min, 30 sec)	822 sec (13 min, 42 sec)
Experimental standard deviation	193 sec	701 sec
90% confidence range	270 ± 100 sec [170, 370]	822 ± 365 sec [457, 1187]
Average number of collisions (entire search space)	34.4 ± 23	63.5 ± 54
Average number of collisions (target cell only)	3.9 ± 2.2	4 ± 2.7

Note: Times are for 3 bots to overlap target cell.

The experimental results confirm that the passage times scale do not scale linearly with the size of the search space. The large search space is two times bigger than the small search space but the average time increased by about a factor of three. We expect the passage time to increase as the square of the search space. We attribute the difference to the large standard deviation in the experimental results.

6 Conclusions

This paper has presented a new way to model the BEECLUST swarm algorithm. The BEECLUST algorithm does not require position information for the swarm agents so it is useful in many scenarios that other swarm search algorithms will not work. We model the clustering aspect of the BEECLUST algorithm using a birth-death Markov chain. The model allows us to determine the expected long-term (steady state) location of the bots and determine how long it takes for the bots to cluster near a peak in the search space.

The Markov model shows that, stochastically, the bots will eventually cluster near the peak(s) in the search space. The model also shows that for reasonably-sized search areas and typical bots, the passage time is rather large. The passage time can be reduced by increasing the number of bots in the swarm or by increasing the collision radius of the bots. Increasing the collision radius will increase the number of bot-bot collisions and hasten the clustering near the peak.

The results of the theoretical passage time calculations were confirmed by experiment. We programmed a small swarm to perform the BEECLUST. For two different search spaces, the theoretical value using the Markov analysis for the time for the bots to cluster was within the 90% confidence interval of the experimental results. In general, the BEECLUST algorithm shows promise for search applications with simple bots.

Acknowledgements

The author would like to acknowledge the financial support from Murray State University through the Committee on Institutional Studies and Research (CISR). He is also grateful for the help of Mr. Kyle Frye and Mr. Michael Siebold, who performed the laboratory measurements.

References

Allen, A.O. (1978) *Probability, Statistics, and Queueing Theory with Computer Science Applications*, Academic Press, New York.

Allen, L.J.S. (2003) *An Introduction to Stochastic Processes with Applications to Biology*, CRC Press, Boca Raton.

Bodi, M., Thenius, R., Schmickl, T. and Crailsheim, K. (2011) 'How two cooperating robot swarms are affected by two conflictive aggregation spots', in Kampis, G. (Ed.): *Advances in Artificial Life: Darwin Meets von Neumann,* Vol. 5778, pp.367–374, Springer, Berlin.

Clerc, M. and Kennedy, J. (2002) 'The particle swarm – explosion, stability, and convergence in a multi-dimensional complex space', *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 1, pp.58–73.

Eberhart, R. and Kennedy, J. (1995) 'A new optimizer using particle swarm theory', *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Japan, pp.39–43.

Hereford, J.M. (2010) 'Analysis of a new swarm search algorithm based on trophallaxis', *2010 IEEE Congress on Evolutionary Computation*, Barcelona, Spain, July, pp.1097–1104.

Hereford, J.M. (2011) 'Analysis of BEECLUST swarm algorithm', *2011 IEEE Swarm Intelligence Symposium*, Paris, France, April, pp.192–198.

Hereford, J.M. and Siebold, M. (2008) 'Multi-robot search using a physically-embedded particle swarm optimization', *International Journal of Computational Intelligence Research*, March, Vol. 4, No. 2, pp.197–209.

Hereford, J.M. and Siebold, M. (2010) 'Bio-inspired search strategies for robot swarms', in Martinez Martin, E. (Ed.): *Swarm Robotics: From Biology to Robotics*, March, pp.1–26 Intech Publishing.

Hereford, J.M., Siebold, M. and Nichols, S. (2007) 'Using the particle swarm optimization algorithm for robotic search applications', *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, Honolulu, HI, April, pp.53–59.

Kernbach, S., Thenius, R., Kernbach, O. and Schmickl, T. (2009) 'Re-embodiment of honeybee aggregation behavior in an artificial micro-robotic system', *Adaptive Behavior*, Vol. 17, No. 3, pp.237–259.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J-C., Floreano, D. and Martinoli, A. (2009) 'The e-puck, a robot designed for education in engineering', *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, Vol. 1, No. 1, pp.59–65.

Parrott, D. and Li, X. (2006) 'Locating and tracking multiple dynamic optima by a particle swarm model using speciation', *IEEE Transactions on Evolutionary Computation*, August, Vol. 10, pp.440–458.

Pugh, J. and Martinoli, A. (2007) 'Inspiring and modeling multi-robot search with particle swarm optimization', *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, Honolulu, HI, April, pp.332–339.

Schmickl, T. and Hamann, H. (2011) 'BEECLUST: a swarm algorithm derived from honeybees. Derivation of the algorithm, analysis by mathematical models, and implementation on a robot swarm', in Yang, X. (Ed.): *Bio-inspired Computing and Networking*, pp.95–137.

Schmickl, T., Hamann, H., Wörn, H. and Crailsheim, K. (2009) 'Two different approaches to a macroscopic model of a bio-inspired robotic swarm', *Robotics and Autonomous Systems*, Vol. 57, No. 9, pp.913–921.

Schmickl, T., Thenius, R., Möslinger, C., Radspieler, G., Kernbach, S., Szymanski, M. and Crailsheim, K. (2008) 'Get in touch – cooperative decision making based on robot-to-robot collisions', *Autonomous Agents and Multi-Agent Systems*, Vol. 18, No. 1, pp.133–155.

Siebold, M. and Hereford, J.M. (2008) 'Easily scalable algorithms for dispersing autonomous robots', *2008 IEEE SoutheastCon*, Huntsville, AL, April, pp.545–550.

Spears, D., Kerr, W. and Spears, W. (2006) 'Physics-based robot swarms for coverage problems', *The International Journal of Intelligent Control and Systems*, September, Vol. 11, No. 3, pp.124–140.