Bioinsp. Biomim. **2** (2007) S182–S197 doi:10.1088/1748-3182/2/4/S07

From honeybees to Internet servers: biomimicry for distributed management of Internet hosting centers

Sunil Nakrani[1,2] **and Craig Tovey**[2]

[1] Computing Laboratory, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK
[2] School of Industrial and Systems Engineering, Georgia Institute of Technology, 765 Ferst Drive, NW, Atlanta, GA 30332-0205, USA

E-mail: Sunil.Nakrani@gatech.edu

Received 9 August 2007
Accepted for publication 10 August 2007
Published 16 October 2007
Online at stacks.iop.org/BB/2/S182

Abstract
An Internet hosting center hosts services on its server ensemble. The center must allocate servers dynamically amongst services to maximize revenue earned from hosting fees. The finite server ensemble, unpredictable request arrival behavior and server reallocation cost make server allocation optimization difficult. Server allocation closely resembles honeybee forager allocation amongst flower patches to optimize nectar influx. The resemblance inspires a honeybee biomimetic algorithm. This paper describes details of the honeybee self-organizing model in terms of information flow and feedback, analyzes the homology between the two problems and derives the resulting biomimetic algorithm for hosting centers. The algorithm is assessed for effectiveness and adaptiveness by comparative testing against benchmark and conventional algorithms. Computational results indicate that the new algorithm is highly adaptive to widely varying external environments and quite competitive against benchmark assessment algorithms. Other swarm intelligence applications are briefly surveyed, and some general speculations are offered regarding their various degrees of success.

1. Introduction

The unrelenting growth of myriad Internet services coupled with the ubiquitous Internet characterized by its open and dynamic environment presents key challenges for designing and managing Internet computing infrastructure on which such services are hosted. A typical centralized Internet service hosting center contains an ensemble of commodity servers on which Internet services are hosted for a fee. Faced with unpredictable Internet request traffic, the hosting center must orchestrate its server ensemble amongst co-hosted services so as to maximize revenue earned from hosting fees.

A honeybee colony faces a similar problem. Nectar collection is an essential foraging activity, requiring typically 50–70 kg of nectar during summer and approximately 30–50 kg reserve for winter survival. A honeybee colony must orchestrate its foragers amongst flower patches (nectar sources) so as to maximize nectar intake. However, as in the case of a hosting center, forager orchestration must deal with unpredictability of nectar supply and an unknown scale of fluctuation from dearth to abundance.

We have proposed a biomimetic server orchestration algorithm which performs well on a set of test cases [1]. The inspiration for biomimicry was the remarkable resemblance between the honeybee colony's problem of allocating foragers amongst flower patches to maximize nectar influx and the host center's problem of allocating servers amongst host customers to maximize revenue. In this paper, we give details of the server and forager orchestration problems and describe the self-organizing honeybee solution. Next, we give details of the two problems' many similarities and probe the extent to which the homology holds. Thence, we describe the self-organizing biomimetic algorithm and highlight its information flows and feedback mechanisms.

Biomimetic solutions should be assessed in comparison with existing conventional solutions. We employ three comparative benchmark algorithms along with simulation and

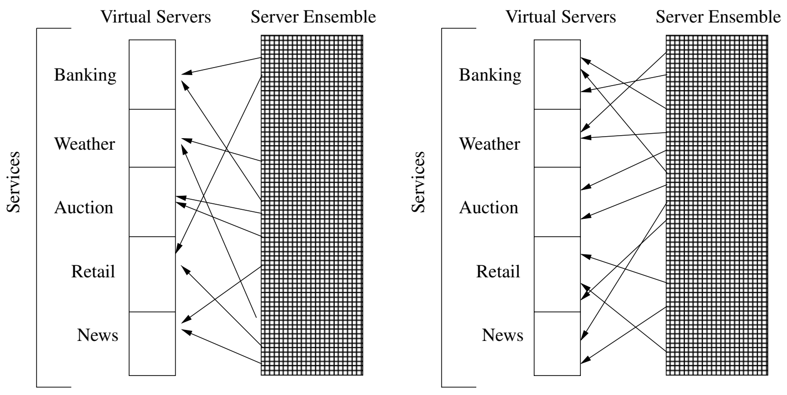

Figure 1. Server ensemble orchestration at two distinct time instants.

request arrival models for computational testing. We assess the solution quality over a considerably broader suite of problems than considered previously [1]. Further, we test the adaptiveness of the algorithm compared with conventional methods. The biomimetic algorithm competes well in quality and with distinction in adaptiveness. We conclude with a brief survey of other swarm intelligence applications and some general speculations regarding their various degrees of success.

2. Internet hosting center orchestration

2.1. Internet host centers

A burgeoning World Wide Web (WWW) user population depends upon the Internet for business-critical and day-to-day activities. Internet traffic variability creates unusually demanding scalability and availability requirements. Service request traffic often exhibits bursts above the mean for lengthy periods over a wide range of timescales, and varies in terms of peak to trough by factors of more than 100 in less than an hour [2]. As a consequence, Internet computing infrastructure often entails centralized Internet hosting centers composed of commodity server appliances, managing content delivery of multiple co-hosted services to global users for a hosting fee [3–5]. Given that the hosting fee may be negotiated on the basis of requests served, maximization of the total requests' revenue becomes the prime incentive of the hosting center.

2.2. Server orchestration problem

Server ensemble management, as depicted in figure 1, involves orchestrating servers in the ensemble to serve requests for a given co-hosted service. A group of servers serving requests for a given co-hosted service make a virtual server which can grow and shrink in capacity. For the entire time horizon, orchestration decisions have to be made as to which servers in the ensemble are part of which virtual servers. At any moment, any server may be orchestrated to remain with its current virtual server or to join another virtual server. If a server is orchestrated to leave a given virtual server and join another, then the server in question suffers a downtime during which it earns no revenue. All these orchestration decisions have to be made in the face of unknown and highly variable future request arrivals for each of its co-hosted services. The average time to serve a request and fee paid per request may differ for each service. If, for a given service, the number of servers necessary to serve the present demand is not orchestrated, then requests in the queue will spend time waiting to be served. This will increase the average service response time and may influence whether a customer is willing to wait for a response. A revenue opportunity is lost if a customer balks.

Suppose an Internet hosting center with N servers co-hosts M Internet services. There are M virtual servers $V_1, \ldots, V_M$, hosting Internet services with respective service request queues $Q_1, \ldots, Q_M$ buffering streams of arriving requests. Each virtual server V_i is composed of n_i servers, $n_i \geqslant 0$, such that $\sum_{i=1}^{M} n_i = N$. The unit of orchestration is a single server and the cost of orchestration is some fixed time units, i.e. the server becomes unavailable for this fixed time interval as it undergoes the repurposing process. Discretize the time horizon into T time steps indexed by $t = 1, \ldots, T$ and let S_t denote the state of the system at the start of time step t, including the server membership of the virtual servers $V_1, \ldots, V_M$, status of any unavailable servers and residual requests in the queues from the time step $t - 1$. Let $A_t = \{a_1^t, \ldots, a_M^t\}$ denote the request arrivals during a time step t and $\pi^t = \{\pi_1^t, \ldots, \pi_M^t\}$ denote the server ensemble orchestration policy for a time step t. Let $R(\pi^t, S, A)$ denote the revenue earned by an orchestration policy π^t during a time step t with the initial state S and arrivals A, and similarly let $f(\pi^t, S, A)$ denote the state of the system which would eventuate at the start of the $(t+1)$st time step. Thus, the server orchestration problem is

Maximize:

$$\sum_{t=1}^{T} R(\pi^t, S_t, A_t) \quad \text{(total reward)}$$

Subject to:

$$S_{t+1} = f(\pi^t, S_t, A_t) \quad \text{(orchestration, request traffic)},$$

and since A_t are not known in advance, the problem is not deterministic.

2.3. Related work

The problem of dynamically orchestrating resources as unpredictable requests arrive has been studied in a general

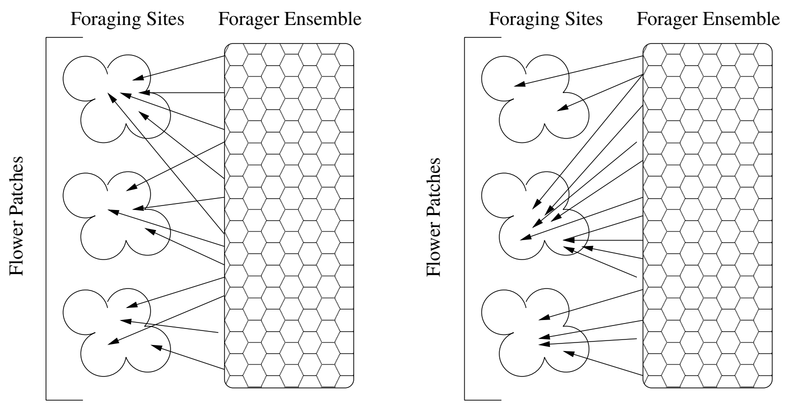

Figure 2. Foragers' orchestration at two distinct time instants.

abstract context as the *K*-server problem [6]. In this formulation, we are given a metric space *S* and *K* mobile servers that reside at points in *S*. When a request arrives at point $x \in S$ it must be processed by moving one of the *K* servers to *x*, at a cost equal to the distance moved in *S*. The objective is to minimize cost in serving a sequence of requests.

Dynamic resource orchestration has been well studied in the context of effectively exploiting available resources to realize certain performance goals. Several papers address the problem of sharing processor resources between competing applications to improve throughput and/or real-time response [7–13]. Numerous papers focus on sharing processors in distributed systems of networked machines for throughput [14–22]. However, these research efforts focus on sharing CPU cycles amongst competing applications as opposed to a whole server being shared in a serial manner and, moreover, they do not consider orchestration downtime cost.

A smaller number of papers have considered the problem in a context similar to that described in this paper. Wolf and Yu [23] propose a server orchestration algorithm for the case where a single server can handle requests for multiple services, and the performance objective is to minimize response time by load balancing. Farias *et al* [24] formulate the server orchestration problem in a dynamic programming framework, but propose an approximate linear programming solution given that the dynamic programming approach is computationally infeasible. It computes an orchestration policy based on a specific arrival rate model. In contrast, we do not make any *a priori* assumptions regarding request arrivals to make orchestration decisions. Palmer and Mitrani [25] examine the server orchestration problem and propose non-optimal heuristics that are easily implementable. However, only a simple case of hosting two services with a total of nine servers is considered. Chase *et al* [3] propose an economic approach to server allocation for cases where energy conservation is the primary goal. Appleby *et al* [26] propose a service level agreement (SLA) based approach, where monitoring agents provide load and performance feedback to a central resource director which makes orchestration decisions. In our approach, feedback is provided by all servers in the hosting center but individual servers are responsible for their own orchestration decisions.

Finally, Jayram *et al* at IBM [5] prove that no finite competitive ratio guarantee is possible for any orchestration algorithm which has no knowledge of the future. The authors give a theoretical algorithm which, like our optimal omniscient algorithm, described in section 7.1, requires knowledge of the future and hence cannot be used in practice. They also describe a related heuristic which does not require such knowledge but apparently was not implemented or tested at the time. Their heuristic turns out to be of precisely the same type as our greedy orchestration algorithm, described in section 7.1, since it employs a forecasting module, details of which are not described. This confirms that our greedy algorithm provides a good comparative base for classical optimization methods.

3. Honeybee (*Apis mellifera*) colonies: forager orchestration problem

A typical honeybee colony contains approximately 20 000–50 000 bees with one queen, a few drones and the rest workers. Nectar collection is a principal foraging activity given that during summer the colony consumes approximately 70 kg of nectar, and approximately 50 kg of additional nectar is needed as a reserve for winter survival without which the queen freezes and the colony dies.

In colder climate zones such as Northern Europe, nectar is effectively available only in the summer for about 12 weeks each year. During this critical time period, a honeybee colony must orchestrate its foragers among the blooming flower patches in the surrounding countryside to collect the nectar it requires. However, the availability of nectar is unpredictable and the scale of fluctuation from dearth to abundance is unknown. From hour to hour and day to day, the availability and quality of nectar vary with micro-climatic conditions, sun position, blooming/withering cycle and exploitation. It is not uncommon for a colony's nectar intake rate to fluctuate by a factor of more than 100:1 within a day or so [27]. As illustrated in figure 2, given the volatility of nectar supply, the colony must make time-varying orchestration decisions as to the numbers of foragers to be deployed at each known flower patch. A forager bee can transfer from one flower patch to another, but the forager in question requires time to learn the

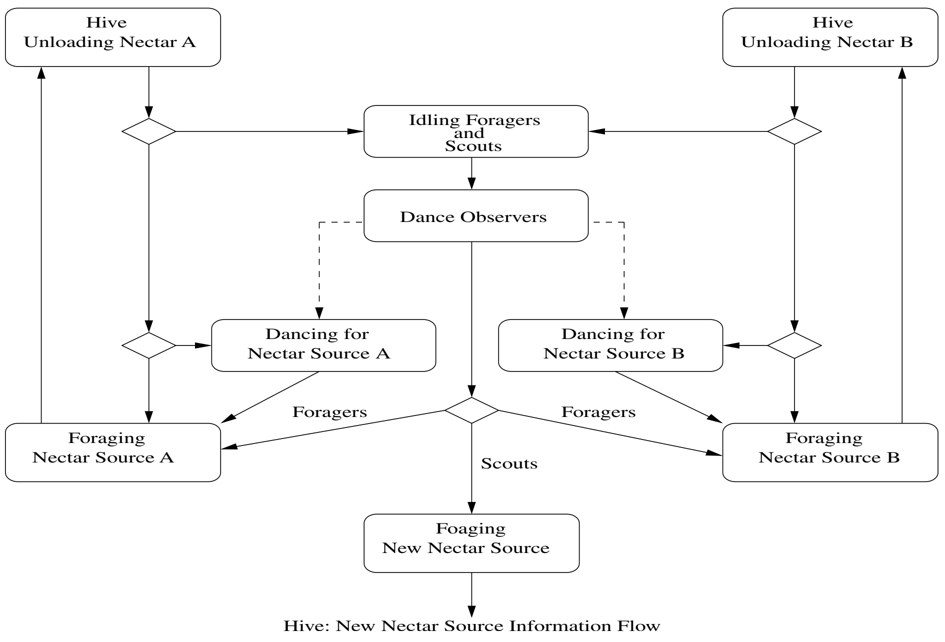
Hive
Unloading Nectar A
Hive
Unloading Nectar B
Idling Foragers and Scouts
Dance Observers
Dancing for Nectar Source A
Dancing for Nectar Source B
Foragers
Foragers
Foraging Nectar Source A
Foraging Nectar Source B
Scouts
Foaging New Nectar Source
Hive: New Nectar Source Information Flow

Figure 3. The information flow and decision processes in a honeybee colony (adapted from Seeley [27]).

location of the new flower patch during which she makes no contribution to colony's nectar collection endeavors.

Consider a honeybee colony with N nectar foragers and M flower patches. Let function $f_i(x_i)$ denote the total value returned from the ith flower patch by x_i foragers collecting nectar at the patch. Provided the value functions $f_i()$ do not change over time, the resulting *static* forager orchestration problem is

$$\begin{array}{lll} \text{Maximize:} & \sum_{i=1}^{M} f_i(x_i) & \text{(value function)} \\ \text{Subject to:} & x_i \geqslant 0 & \text{(orchestration)} \\ & \sum_{i=1}^{M} x_i \leqslant N & \text{(foragers).} \end{array}$$

Applying standard calculus yields the optimal *static* solution conditions $f_i'(x_i) = \lambda \, \forall x_i > 0$, $f_i'(0) \leqslant \lambda \, \forall x_i = 0$. That is, the marginal contributions are equal at each flower patch that is actively being foraged. The Lagrange multiplier λ is the value of these marginal contributions, whilst the the marginal contributions at inactive patches are $\leqslant\lambda$. This is intuitive since if the marginal contribution $f_i'(x_i)$ at the flower patch i exceeds the marginal contribution $f_j'(x_j)$ at an active flower patch j, then more nectar could be obtained by orchestrating a forager from the flower patch j to i. The intuitive explanation is similar to that of the so-called marginal value theorem (MVT) [28]. The MVT is usually cast in terms of a single forager deciding when to select a different patch, although that casting neglects the cost of travel between patches. Instead, to see the parallel, think of a single forager deciding how to allocate its time among patches as being analogous to the hive 'deciding' how to allocate its forager bees among patches. Note that the value functions $f_i(x_i)$ do incorporate the cost of travel between hive and patch.

The problem faced by the honeybee colony is more challenging than the static version outlined above, and is akin to the Internet host problem described previously. Flower patch availabilities and profitabilities are highly variable, but responsiveness, i.e. the rate at which foragers are recruited to another patch, must be traded off against the temporary loss of the forager's productivity when she is recruited to a different patch.

4. Self-organizing honeybee forager orchestration

Colonies of social insect possess what has been classed as *swarm intelligence* [29]. A broad definition of the term implies a sophisticated collective behavior borne out of primitive interactions among members of the group to solve problems beyond the capability of individual members. Such colonies are characterized by (i) self-organization: decentralized and unsupervised coordination of activities, (ii) adaptiveness: response to dynamically varying environments and (iii) robustness: accomplishing the group's objective even if some members of the group are unsuccessful.

A model of self-organizing behavior that takes place within a colony of honeybees has been presented by Seeley [27]. The model describes interactions between members of the colony and the environment that leads to dynamic distribution of foragers to efficiently collect nectar from an array of flower patches that are capricious in terms of their value to the colony. The honeybee colony, as depicted in figure 3, acquires information with respect to foraging prospects in the surrounding countryside continuously and acts upon this information in concert with nectar requirements of the colony to orchestrate foragers amongst prospective flower patches. The self-organizing behavior model comprises the following key elements.

Information procurement. A honeybee colony continually refreshes information relating to known (currently being foraged) and any new enticing flower patches. A forager bee collecting nectar at a flower patch returns with information about the patch, while a scout discovers a new flower patch and returns with information.

Information type. On each visit to a flower patch, the forager bee returns to the hive with collected nectar as well as value (based on variables such as nectar sugar concentration, nectar bounty, distance from the hive and an internal scale of quality) and location (geographical location of the flower patch) information.

Information filtering. With a large volume of information influx, a colony selects the information pertaining to high value rated patches. The selection occurs as a consequence of interaction between a retuning forager and a receiver bee which sets filtering threshold.

Information broadcast. If filtering threshold permits, each forager will broadcast information about her patch by performing waggle dance on the dance floor inside the beehive.

Information emphasis. A forager bee combines the decision to waggle dance with a value rating to present the information that conveys the degree of goodness and location of her patch. Goodness of a patch is exhibited by a greater number of waggle runs, while the location is exhibited by the orientation of the waggle run and the number of waggle turns in a run.

Information audience. The information is consumed by idling foragers who are potential candidates for deployment to flower patches. Scouts and an active forager do not use this information.

Information utilization. An idling forager randomly selects a dance, follows it to learn the location of the patch and leaves the hive to collect nectar.

Fundamentally, the self-organizing forager orchestration in a honeybee colony emerges due to information flow into the colony and its response to this information. In summary, foraging bees visiting flower patches return to the hive with nectar and with a value rating (function of nectar quality, nectar bounty and distance from the hive) of respective flower patches. At the hive, forager bees interact with receiver bees to offload collected nectar. This interaction provides feedback on the current status of nectar flow into the hive. This feedback mechanism sets a response threshold for an enlisting signal. An amalgamation of response threshold and value rating influences the length of the enlisting signal known as the waggle dance. The waggle dance is performed on the dance floor where inactive foragers can observe and follow. Effectively, each active forager bee provides feedback on her local flower patch while observing bees have access to the set of attractive food sources being capitalized by the colony. However, individual foragers do not acquire the full set of global knowledge but rather randomly select a dance to observe from which they can learn the location of the flower patch and leave the hive to forage. The self-organizing proportionate orchestration pattern, derived from multiple and proportionate feedback on goodness of food sources, is described by Seeley *et al* [30] and validated by experimental study on a natural honeybee colony. The model given by Bartholdi *et al* [31] predicts a steady state pattern of forager orchestration where the rate of value accumulation equalizes among forage sites being exploited, i.e. self-organizing orchestration results in equal average nectar return $\frac{f_i(x_i)}{x_i} = \mu \; \forall i$, where $f_i(x_i)$ denotes the value returned from x_i foragers collecting nectar at the ith forage site.

5. Homology: an Internet hosting center and honeybee colony

A cursory glance at the server and forager orchestration problems, as depicted in figure 4, would suggest a similarity between the two problems, at least on a superficial level. The server ensemble in an Internet hosting center is analogous to forager bees in a honeybee colony, whilst the service request queues pertaining to co-hosted Internet services are analogous to sources of nectar to be exploited profitably.

An in-depth examination at finer granularity revealed a remarkably close mapping between the two problems, as illustrated in table 1. The strong homology between the server and the forager orchestration problems motivated our biomimetic approach. One common characteristic is that in each problem, the supply—service requests and nectar—changes unpredictably over time. Request streams are variable due to fluctuating request arrival rates and balking behavior of users waiting for service, whilst flower patches are variable due to fluctuations in micro-climate, quality, density and nectar replenishment rates.

Another characteristic that is shared by two problems relates to over-allocation and degradation in revenue/nectar collection rate. The amount of time required by a server to earn revenue from a hosted Internet service will depend on the service characteristics as well as the total number of servers servicing a given request queue. If more servers are orchestrated than necessary to a service, the revenue rate will degrade due to each server having to wait longer to find a request to serve. Similarly, the amount of time needed by a forager to return with a nectar payload depends on characteristics of the forage site as well as on the number of foragers collecting nectar there. The rate of nectar payload being returned to the colony will degrade if overly many foragers are orchestrated to the site since each forager will take longer to find flowers whose nectar has been replenished but not yet harvested. There is an additional lovely parallel between the two problems. If a honeybee makes repeated visits to a flower patch, she improves her ability to extract nectar from that particular kind of flower [27]. If a server repeatedly serves customers of the same type, its service speed for that type of customer tends to increase, because of the storage of data in its caches [32].

Let us explore the accuracy of this aspect of the homology. In the case of a honeybee colony, let $f_i(x_i)$ denote the number of nectar loads returned per minute by a total of x_i bees devoted to foraging at the ith flower patch. For simplicity, we suppress the index i and normalize the nectar quality. Let T denote

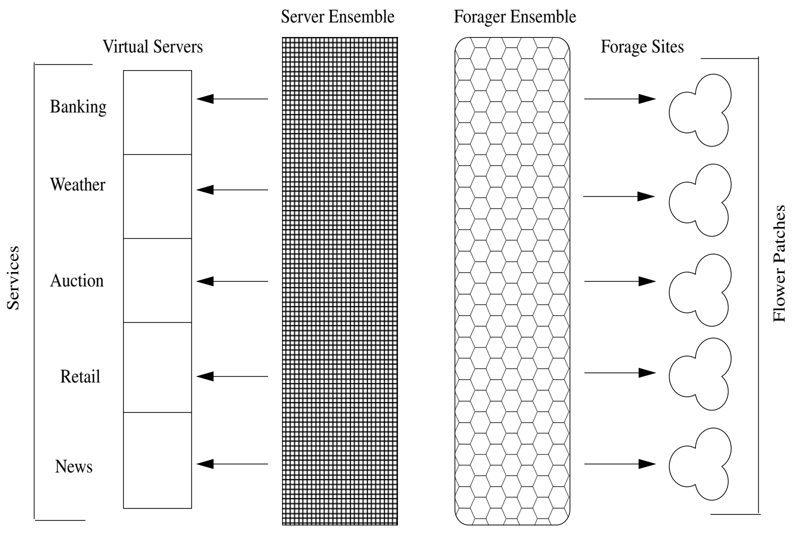

Figure 4. Server and forager orchestration problems.

Table 1. Parallels between server and forager orchestration problems.

Internet hosting center	Honeybee colony
An ensemble of *N* servers	An ensemble of *N* nectar foragers
Unit of orchestration: single server	Unit of orchestration: single nectar forager
Co-host *M* Internet services to be accessed by requests	*M* distinct flower patches in the surrounding countryside
A group of servers (virtual server) with the same Internet service application serving requests from a specific service queue	A group of nectar foragers collecting nectar at a specific flower patch
Time to service a request will be dependent on the Internet service application	Time to travel will be dependent on the location of the flower patch
Time taken by a server to find a request to serve at an Internet service application increases if the number of servers in the group increases	Time taken by a forager to collect nectar at a flower patch increases if the number of foragers in the group increases
A server orchestrated to another Internet service incurs downtime due to purging of current and installation of new Internet service application/data	A forager orchestrated to another flower patch incurs downtime due to time spent learning the location and successful discovery of the patch
An Internet service application and its requests have a *value-per-request*	A flower patch has the quality of nectar (sugar concentration)
Variable rates of requests arrival and balking behavior	Variable rates of flower patch quality, density and replenishment
A server repeatedly serving requests of the same type results in improved response time	A forager repeatedly foraging on flowers of the same type results in improvement in its ability to extract nectar

the travel time from the patch to the hive and back (including nectar unload time). Hence, of the $x/f(x)$ minutes required by a single bee to collect and deliver a nectar load, the fraction $1 - Tf(x)/x$ is spent collecting at the patch. Therefore (on average) $x - Tf(x)$ bees are at the flower patch at any time, and the forage time of a single bee is $x/f(x) = T + w(x - Tf(x))$ minutes, where $w(y)$ denotes the (steady state) time to collect a nectar load if y bees are actively foraging there. One may recover $f(x)$ from $w(y)$ via

$$f(x) = f\left(y + \frac{yT}{w(y)}\right) = \frac{y}{w(y)}. \tag{1}$$

In the case of an Internet hosting center, $f(x)$ denotes the revenue rate per minute of x servers in the virtual server V serving requests from service queue Q and T denotes the service processing time for V. In this case equation (1) represents the revenue-earnings time of a single server, where

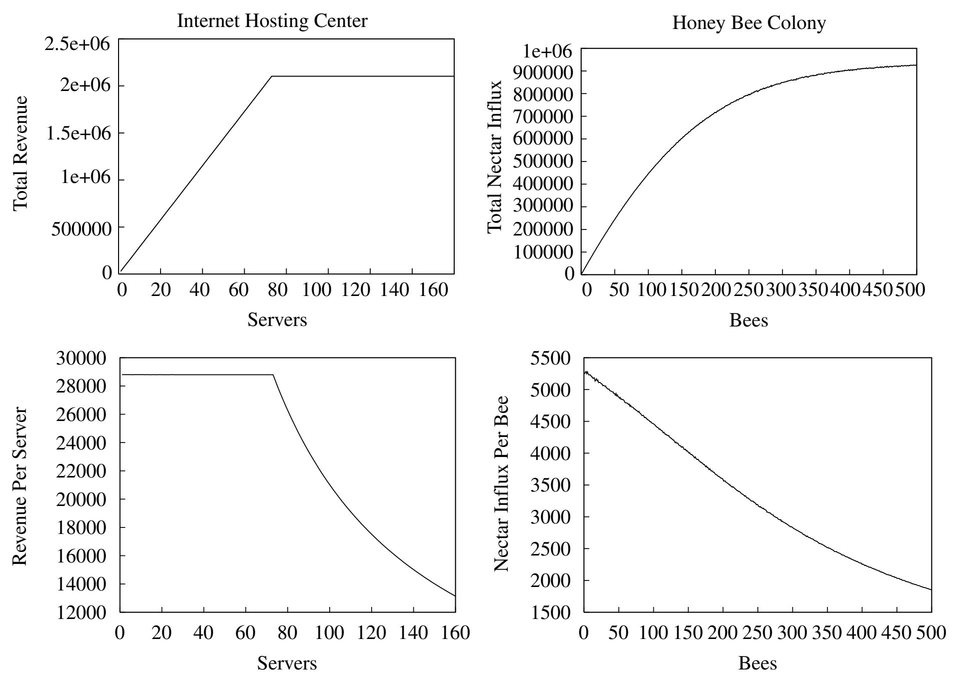

Figure 5. Revenue and nectar influx functions of a honeybee colony foraging at a single flower patch and an Internet hosting center serving requests of the same type at a hosted service.

$w(y)$ denotes the (average) waiting time for one server amongst y to acquire a customer from queue Q.

Thus, the cycle times of both bee and server have the same functional description. We now show that the functions $w()$ and hence $f()$ have similar qualitative behavior in the two cases. For the honeybee colony, $w(1)$ is the nectar load collection time from the patch with one bee collecting nectar. Now, if two bees are collecting nectar from the flower patch, then there is a slight possibility of interference due to the activity of each bee. One bee will collect nectar from a specific flower whose nectar will be replenished with some delay. Soon afterwards, the other bee may attempt to collect nectar from the said flower with an outcome that it will fail to find any nectar and will lose time to try another flower. Therefore, $w(2) \gtrapprox w(1)$ because of the slight interference effect. If a third bee were collecting nectar as well, the other bees would experience an additional interference effect because she would deplete additional flowers (we are neglecting the second-order effect of interference reducing interference rates.) Thus, $w(3) \gtrapprox w(2)$. By the same reasoning, for small values of y (relative to total flower patch nectar replenishment rate $\mathcal{N}$ loads/min), we have $w'(y) \gtrapprox 0$.

In the case of two servers at a virtual server of the Internet host center, each takes an available request from the service queue, but a slight interference effect can be triggered when Q contains exactly one request. Thus, $w(2) \gtrapprox w(1)$. More generally, for small values of y the interference effect increases in y but is not large since it can only occur if at least one but fewer than y requests are present in the service queue. Thus, as with the honeybees, we have $w'(y) \gtrapprox 0$. At the other extreme, if $\frac{y}{w(1)} \gg \mathcal{N}$, the foraging capacity will essentially saturate the patch's nectar production. Thus, $w(y) \approx \frac{y}{\mathcal{N}}$. In the host center, if $\frac{y}{w(1)} \gg \mathcal{N}$ the virtual server will essentially lose zero requests to balking, and thus again $w(y) \approx \frac{y}{\mathcal{N}}$. At intermediate values $y \gtrapprox w(1)\mathcal{N}$, it is not so clear how the function behaves. Interference effects force $w(y) > w(1)$, but nectar harvesting (respectively customer service) should be close enough to saturation that one additional forager (respectively server) increases nectar (respectively revenue) collection only slightly; thus $\left[\frac{y}{w(y)}\right]' \gtrapprox 0$ whence $w'(y) \lessapprox \frac{w(y)}{y}$.

We developed a simulation, since we could not obtain empirical data, to test bees foraging at a flower patch. Flowers were modeled as points in a grid, which replenish nectar at some rate; bees were modeled as traveling randomly from the grid cell to the neighboring cell, until a full load is collected. Other model parameters included number of flowers in the patch, nectar extraction time and round trip time. We performed a simulation run for each increment in allocation of a bee to the flower patch over a 24 h time horizon and collected performance data. The plot of total nectar influx and nectar influx per bee for varying allocations of bees to the flower patch is depicted in figure 5 (honeybee colony).

For the case of the Internet host, we ran the hosting center simulation model [33] over a 24 h time horizon with one virtual server and an inhomogeneous Poisson (IP) request stream distribution, for each increment in server allocation to the virtual server. Figure 5 (Internet hosting center) illustrates the total revenue and revenue per server for varying allocation of servers. The simulations confirm our analysis. The qualitative behaviors of the functions are similar in the two cases. However, in the Internet hosting center data the functions appear nearly piecewise linear with a single break point, whereas the slopes in the honeybee data do not change rapidly. In other words, the second derivative spikes much

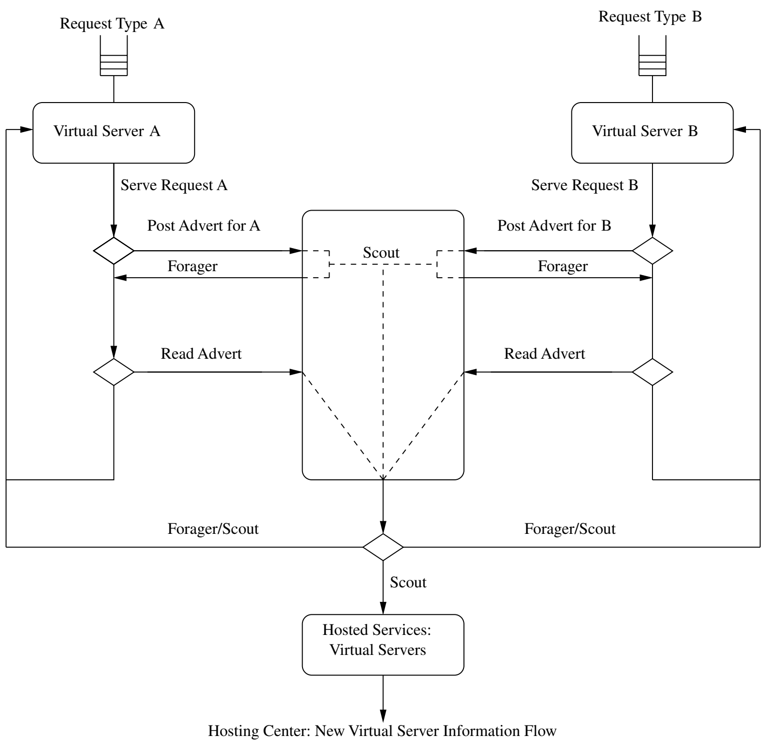

Figure 6. The information flow and decision processes in an Internet hosting center.

more in the Internet center case. The homology fails at this level, although admittedly past the scope of our empirical observation.

The break point for the Internet center occurs roughly at the point at which the traffic intensity (the ratio of customer demand to server capacity) equals 1. Past that break point, a negligible fraction of customers are not served, and the downward slope of the revenue per server simply reflects an increasing number of servers sharing a very nearly fixed customer pool. The same qualitative behavior occurs with the honeybee simulation data, for the same reason. The break point is smoothed out because the interference effects are greater. The difference in interference effects is due to the search time expended by the honeybee to identify and travel to a nectar-laden flower; in an Internet server, the process of matching an available server to an available customer is almost instantaneous.

Thus, there are many close parallels between the problems. Based on our simulation tests, the homology begins to lose accuracy at the level of the nature of the interference effects. The net revenue and net nectar influx functions do have the same rough qualitative behavior, increasing fairly rapidly and then flattening out. However, the nectar influx flattens out considerably more gradually. The functions and their first derivatives are not greatly different, but the second derivatives are.

6. Biomimetic server ensemble orchestration algorithm

A self-organizing server ensemble orchestration algorithm for Internet hosting centers follows readily from the homology that we have exposed. As in the case of the forager orchestration in a honeybee colony, the biomimetic server ensemble orchestration algorithm must acquire information on an ongoing basis in respect of prospective co-hosted Internet services and request load on their service queues as illustrated in figure 6. To gather relevant information, we mimic signals, cues and other mechanisms of a honeybee colony. Each server in a hosting center is either a forager or a scout server. The dance floor is represented by an advertisement board. A waggle dance of some duration is mimicked by an advertisement whose posting appears on the advertisement board for some time length. Furthermore, a flower patch location is represented by a virtual server (service) identifier whilst waggle dancing and following a waggle dance are mimicked by advertisement posting and advertisement reading respectively. The biomimetic server orchestration in an Internet hosting center relies on each server generating information about its own service queue and making it globally available, whilst all servers in the ensemble have the opportunity to view this information and act upon it. Basically, a server services requests and posts advertisements of length varying with revenue potential. It regularly checks its own revenue rate against the hosting center's and adjusts its probabilities of advertising or reorchestration. To reorchestrate, a forager server will randomly select an advertisement and migrate to the corresponding virtual server, whilst a scout server will migrate to a random virtual server.

The behavior of any server in the hosting center is controlled by the pseudocode of figure 7. A server $s_i \in V_j$, on completion of each request from Q_j, will attempt with probability p to post an advert on the advertboard with

```
[A] Initialization– $s_i \in V_j$ serving $Q_j$, Advert posting probability $p$,
    Advert reading probability $r_i$, Revenue rate interval $T_{pr}$,
    Advert reading interval $T_r$
[B] forever
[C]   while $Q_j \neq$ Empty do
          serve request;
          if $T_{pr}$ expired then
              compute revenue rate;
              adjust $r_i$ from lookup table;
          if Flip(p) == TRUE then Post Advert;
          if $T_r$ expired && Read($r_i$)==TRUE then
              /* randomly select an advert or a virtual server */
              Select advert( if forager) or Virtual Server $V_k$(if scout);
              Read advert id $V_k$( if forager);
              if $V_k \neq V_j$ then Switch($V_k$);
      endwhile
    endforever
```

Figure 7. Behavior of a server in the biomimetic orchestration algorithm.

Table 2. Rules for adjusting the probability of reading the advertboard.

Revenue rate	P[Read] r_i
$P_i \leqslant 0.5P_{\text{hosting}}$	0.60
$0.5P_{\text{hosting}} < P_i \leqslant 0.85P_{\text{hosting}}$	0.20
$0.85P_{\text{hosting}} < P_i \leqslant 1.15P_{\text{hosting}}$	0.02
$1.15P_{\text{hosting}} < P_i$	0.00

duration $D = v_j A$, where A denotes the advert scaling factor. Also, it will attempt with probability r_i to read a randomly selected advert from the advertboard if it is a forager or randomly select $V_j, j : 0 \ldots (M-1)$, if it is a scout. The probability r_i is dynamic and changes as a function of the forager/scout server's own revenue rate and the hosting center's overall revenue rate. The revenue rate, P_i, for a server s_i is given by $P_i = \frac{v_j R_i}{T_i}$, where R_i is the total number of requests served by a given server in the time interval T_i. The hosting center's overall revenue rate, P_{hosting}, is given by $P_{\text{hosting}} = \frac{1}{T_{\text{hosting}}} \sum_{j=0}^{(M-1)} v_j R_j$, where R_j denotes the total number of requests served by V_j in the time interval T_{hosting}. A server $s_i \in V_j$ serving queue Q_j determines its profitability by comparing the revenue rate P_i with P_{hosting}. It updates r_i according to the rules shown in table 2. These rules quantitatively model forager behavior in real honeybee colonies, given that waggle dancing foragers never get recruited to another patch and foragers use global information (hive's nectar intake rate cued by time to find a receiver bee) to decide whether to dance or not. Thus, bees at a profitable patch have a decreased chance of following another waggle dance.

7. Assessing self-organizing server orchestration

In this section, we describe three additional orchestration algorithms utilized for assessing comparative performance of the self-organizing orchestration algorithm along with the simulation model of an Internet hosting center, using all four orchestration algorithms. We also describe a suite of request arrival patterns utilized to simulate demands for co-hosted Internet services. In our previous work, we have described the assessment orchestration algorithms, the simulation model and the request arrival suite in detail [33]. Below, we summarize the assessment orchestration algorithms and simulation model whilst expanding on the request arrival suite to encompass a wider range of request patterns and variability in demand intensity.

7.1. Assessment orchestration algorithms

The optimal omniscient algorithm provides an upper bound on possible revenue. It computes, by dynamic programming, the optimal server orchestration given complete knowledge of future request arrivals. Assume that the time horizon is split into $t = 1, \ldots, T$ time steps and let S_t denote the system state at the start of step t, including the orchestration of servers amongst services, and residual requests from time step $t-1$. Let A_t denote the request arrivals during time step t and π denote an orchestration decision for a time step. Let $R(\pi, S, A)$ denote the revenue earned by π during a time step with the initial state S and arrival pattern A. Furthermore, let $f(\pi, S, A)$ denote the system state which would eventuate at the start of the next time step, if the current step starts in state S, allocation π is used and the arrival pattern is A. Define $v^{T+1}(S_{T+1}) = 0$, i.e. there is no state-dependent salvage value at the end of the time horizon. Then the value function for the omniscient orchestration policy is $v^t(S_t) = \max_\pi \{R(\pi, S_t, A_t) + v^{t+1}(f(\pi, S_t, A_t))\}$. The corresponding optimal orchestration policy is given by $\pi_*^t(S_t) = \arg\max_\pi \{R(\pi, S_t, A_t) + v^{t+1}(f(\pi, S_t, A_t))\}$.

The greedy orchestration algorithm represents a conventional heuristic approach to the problem. In particular, the greedy policy chooses orchestration π_{G}^t for time step t as $\pi_{\text{G}}^t = \arg\max_\pi R(\pi, S_t, A_{t-1})$. The state is then updated according to $S_{t+1} = f\left(\pi_{\text{G}}^t, S_t, A_t\right)$. Thus, the algorithm chooses, for time step t, an allocation policy that would maximize revenue if actual arrivals A_t were the same as the previous arrivals A_{t-1}.

The optimal-static omniscient orchestration algorithm omnisciently chooses the best from amongst all static orchestrations. This is an upper bound on the current level of revenue of many hosting centers, which often maintain their orchestration decision for a month or so. Their revenue will be lower since they do not know the coming month's request arrivals in advance. The policy is computed by $\pi_{\text{static}} = \arg\max_\pi \sum_{t=1}^{T} R(\pi, S_t, A_t)$. Thus, the same orchestration π is used in every time step. The total revenue is $\sum_{t=1}^{T} P(\pi_{\text{static}}, S_t, A_t)$.

7.2. Simulation model

We have developed the discrete event simulation model of an Internet hosting center along with the four server orchestration algorithms—self-organizing, optimal omniscient, greedy and optimal-static. The simulation model is implemented in C++SIM [34] on an IBM XSeries platform with a Linux operating system. The following assumptions are common to all orchestration algorithms: all servers are homogeneous in terms of processing capacity and employ the first-come-first-serve (FCFS) scheduling policy. The time to serve a request is exponentially distributed with a mean service time depending on the request type. Each server is paid a fixed revenue per request served, the amount depending on the request type. An orchestrated server becomes unavailable for a fixed downtime [26]. Each service has an independent request stream, which is held in its own queue. Each request has a waiting threshold to receive service and on crossing this threshold, a request randomly chooses to wait further or balk. The request demand intensity offered to the hosting center ranges from low (0.25), medium (0.7), high (1.0) to super high (1.6). The request demand intensity is defined to be $\frac{1}{\text{Servers}_{\text{center}}}\sum_{i=1}^{VS_M}\frac{\text{Mean Service Time}_i}{\text{Mean Interarrival Time}_i}$. Therefore, the mean demand intensity of a given request arrival pattern over a 24 h period and the total number of servers in the hosting center determine exponentially distributed mean time to serve a request.

The response delay experienced by Internet service users in receiving service plays a fundamental role in their behavior. It is not uncommon for a user to balk or click away as a result of excessive response delay. The balk manager simulates user behavior for each co-hosted service in the hosting center by continuously scanning the respective service queue at regular intervals and randomly deciding with some probability to expunge requests if waiting time in the queue has equaled or exceeded some waiting threshold. The waiting threshold represents user patience for receiving a response to a submitted request whilst the request scanning interval represents how often service users think about balking, having waited for at least the waiting threshold. The balking probability represents the chance that a user may then balk by canceling the request. Nielsen [35] suggests that a delay greater than 10 s causes users to become distracted and makes them more likely to click away. The parameter balk rate represents how often users, having waited more than 10 s for service, think about balking and the parameter balk probability represents the likelihood of balking on each occasion. Given that we have chosen a balk rate of 1.01 s for each waiting user, a low balk chance of 4% is chosen based on appropriate scaling.

Servers can be orchestrated at any time for the self-organizing algorithm. One server per virtual server is a scout and the rest are foragers, reflecting the low proportion of scouts in real honeybee colonies [27]. A scout server randomly orchestrates itself to one of the virtual servers in the hosting center, whilst a forager server randomly orchestrates itself in response to an advertisement read for a particular virtual server. Each server in the hosting center advertises its own virtual server by placing an advertisement on the advertisement board with a given time duration. The advertisement board is kept up to date by purging advertisements with expired display time duration.

Servers using the optimal omniscient and greedy orchestration algorithms are orchestrated at the beginning of each time step and do not change until the beginning of the next time step. In the case of the optimal-static omniscient orchestration algorithm, a single orchestration decision is taken at the beginning of time step 1 and this decision remains unchanged for all other time steps. Consequently, servers do not suffer any downtime.

The self-organizing and the greedy orchestration algorithms compute orchestration decisions online, whilst the optimal omniscient and the optimal-static omniscient orchestration algorithms require orchestration decision to be computed offline using trace data from request arrival patterns. The common simulation parameters for the hosting center are as follows. servers: 50; request waiting threshold: 10 s; balk rate: 1.01 s; balk probability: 0.04; server downtime: 300 s; time step length: 1800 s; revenue (cent): two services (0.1, 0.155), three services (0.1, 0.141 66, 0.1833), four services (0.1, 0.133, 0.166, 0.2); the exponentially distributed mean service time is a function of the request arrival pattern and its demand intensity. Specific parameters for the self-organizing algorithm are as follows: advertisement posting probability: 0.1, scouts per virtual server: 1, and the advertisement board reading probability is given in table 2.

7.3. *Request arrival suite*

We have described a limited suite of request arrival patterns in our previous work [33] that are representative of the Internet request traffic characteristics. Here, we build on the existing suite and give a expanded suite of request arrival patterns encompassing different request behavior characteristics. The expanded suite comprises five request arrival patterns of which three are synthetically generated, one is generated from trace data obtained from an international content provider offering Internet services (a confidentiality agreement restricts us from disclosing its name) and the last is generated from synthetic as well as trace data patterns.

The real request arrival pattern is generated using logs of request activity recorded over a 24 h period on servers hosting Internet services. The simulated request arrival patterns are drawn from the inhomogeneous Poisson process [2, 36] and heavy tail distributions [37, 38]. The step request arrival pattern is a reduced form of the inhomogeneous Poisson process in which the process is either at the maximum or at the minimum level of mean intensity controlled by a ratio parameter which determines the peak-to-trough variation. The heavy request arrival pattern is generated from Cauchy distribution where the mean intensity is controlled by a scaling parameter to limit its range to a manageable level of 0–45 000 requests per second given that the distribution has undefined mean and standard deviation. The inhomogeneous Poisson (IP) request arrival pattern is a true inhomogeneous process where the mean request intensity can vary from 10–10 000 requests per second. In all three simulated request arrival patterns, the mean request intensity is randomly chosen from the allowable intensity range with a frequency of change governed by a parameter that randomly selects an interval from a defined range and upon expiry of this interval a change in the mean request intensity is guaranteed to take place. Finally, the hybrid request arrival pattern, as the name suggests, is generated by combining real and simulated patterns where a particular pattern is randomly chosen from step, heavy, real or IP with a frequency of change governed by a parameter that determines when a new pattern is to be selected, whilst the behavior of the individual request pattern is as described above.

8. Efficacy of the self-organizing algorithm

The principal aim of this section is to demonstrate the adaptiveness of the self-organizing server orchestration algorithm. Adaptiveness generally denotes attributes which change depending on external circumstances in order to remain effective in attaining an objective. In the context of an Internet hosting center, adaptiveness denotes the selection of server orchestrations over a lengthy time horizon so as to achieve high revenue earning in response to demands from a wide variety of traffic stream behaviors.

In order to demonstrate adaptiveness, we can vary the external circumstances (traffic streams) of an orchestration algorithm and observe the resulting revenue stream over a lengthy time horizon. We assess the self-organizing algorithm by comparing its revenue streams to theoretical and classical (conventional) orchestration algorithms' revenue streams in response to demands from a wide variety of traffic stream behaviors, intensity, co-hosting, service level agreements (SLAs). We class the optimal omniscient and optimal-static omniscient orchestration algorithms as theoretical since they require knowledge of the future, whilst the greedy orchestration algorithm is classed as conventional since it requires knowledge only of past demands to select appropriate orchestrations. The methodology entails designing and conducting a series of experiments on the hosting center's simulation framework followed by reasoning about its characteristics through comparative and statistical analyses.

8.1. *Experimental methodology*

The experimental methodology employs a multi-pronged approach to evaluate the adaptiveness property of the self-organizing algorithm. It considers simulation runs with single unique seed values as well as multiple random seed values across different orchestration algorithms and external environments (traffic streams). The results are analyzed, and a comparative revenue stream performance is given in terms of other assessment algorithms. The statistical significance of adaptiveness is analyzed using a nonparametric technique known as the Wilcoxon matched-pairs signed-ranks test [39]. The technique takes into account the size and direction of difference between matched pairs and tests the null hypothesis that the difference between each pair has a median value of zero. Briefly, to perform the test, we calculate differences of each matched pair and rank order the difference by their absolute value, i.e. giving one to the smallest difference, two to the next smallest and so on with zero differences ignored and all tied differences getting an equal rank order. Next, we sum the ranks of the positive difference (W^+) and sum the ranks of the negative differences (W^-). The test statistics is the lesser of the two sums and if the null hypothesis were true and there was no difference, then we would expect the rank sums for positive and negative ranks to be identically distributed. All experiments consider the case of co-hosting two, three and four services with demand for service being driven by a combination of five traffic stream types (step, heavy, real, IP and hybrid), four levels of demand intensity (low, medium, high and super high) and differentiated SLA fees amongst services co-hosted.

Table 3. Normalized revenue stream performance of the self-organizing algorithm aggregated across traffic streams (step, heavy, real, inhomogeneous Poisson and hybrid) and demand intensities (low, medium, high and super high) where for each co-hosting instance (two, three and four services), $n = 20\,(5 \times 4)$ and for overall, $n = 60\,(5 \times 4 \times 3)$.

Hosting	$\frac{\sum_{i=1}^{n} \text{Self-org}_i}{\sum_{i=1}^{n} \text{Opt-omni}_i}$	$\frac{\sum_{i=1}^{n} \text{Self-org}_i}{\sum_{i=1}^{n} \text{Greedy}_i}$	$\frac{\sum_{i=1}^{n} \text{Self-org}_i}{\sum_{i=1}^{n} \text{Opt-stat omni}_i}$
Two-services	0.85	1.11	1.00
Three-services	0.80	1.07	0.98
Four-services	0.83	1.02	1.04
Overall	0.83	1.07	1.01

8.2. *Comparative performance*

We begin by considering the adaptive revenue stream of self-organizing against the optimal omniscient, optimal-static omniscient and greedy orchestration algorithms. We ran simulations with the hosting center managed by each orchestration algorithm in turn. The demand for service was driven by step, heavy, real, inhomogeneous Poisson and hybrid streams at four levels of demand intensity (low, medium, high and super high). The simulation runs considered the cases of co-hosting only two, three and four Internet services with revenue stream performance data collected over a 24 h simulation time horizon. Thus, the total number of simulation runs for each algorithm is 60 ($5 \times 4 \times 3$) for each combination of traffic stream, demand intensity and hosting instance. It follows that for each hosting instance, 20 simulation runs are possible (5×4) resulting from each combination of traffic stream and demand intensity. These runs considered single unique seed values for distribution used in traffic stream generation, balk manager and respective orchestration algorithms. In table 3, we give the performance of the self-organizing algorithm normalized against the optimal omniscient, greedy and optimal-static omniscient algorithms. It is evident that the self-organizing algorithm performs well against benchmark and conventional algorithms.

The simulation runs performed above considered only a single initial seed value for distributions used in traffic streams and orchestration algorithms. Therefore, to further our understanding of statistical significance of the above adaptive revenue stream performance, we now extend the experiment where ten simulation runs are performed for each instance of hosting (two, three and four), traffic stream (step, heavy, IP and hybrid) and demand intensity (low, medium, high and super high) with unique random [40] initial seed values for distributions used in traffic streams and orchestration algorithms. The hosting center is managed, in turns, by self-organizing, greedy and optimal-static omniscient algorithms. We did not consider the optimal omniscient algorithm for these runs since it is apparent that it will always outperform other algorithms and to avoid its time-space intensive offline policy computation. Our extended experiment, therefore, entailed 480 ($10 \times 3 \times 4 \times 4$) different seed values for each algorithm with a total of 1440 ($10 \times 3 \times 3 \times 4 \times 4$) simulation runs resulting from each combination of seed value, algorithm, hosting instance, traffic stream and demand intensity. In table 4, we give the normalized revenue stream performance

Table 4. Normalized revenue stream performance of the self-organizing algorithm aggregated across traffic streams (step, heavy, inhomogeneous Poisson and hybrid) and demand intensities (low, medium, high and super high) where for each co-hosting instance (two, three and four services), $n = 160$ (10 × 4 × 4) and for overall, $n = 480$ (10 × 4 × 4 × 3).

Hosting	$\frac{\sum_{i=1}^{n} \text{Self-org}_i}{\sum_{i=1}^{n} \text{Greedy}_i}$	$\frac{\sum_{i=1}^{n} \text{Self-org}_i}{\sum_{i=1}^{n} \text{Opt-stat omni}_i}$
Two-services	1.03	1.01
Three-services	1.12	1.01
Four-services	1.03	0.99
Overall	1.06	1.00

of the self-organizing algorithm against the revenue stream of the greedy and optimal-static omniscient algorithms. We observe that the self-organizing algorithm outperforms the greedy algorithm across each hosting instance and overall whilst being competitive against the optimal-static omniscient algorithm where it outperforms for two and three services co-hosted and underperforms for four services co-hosted. In table 5, we give the statistical significance of the adaptive performance using the Wilcoxon matched-pairs signed-ranks test. We observe that the adaptive revenue stream performance of the self-organizing algorithm compared to the greedy algorithm is highly significant overall. Against the optimal-static omniscient algorithm, statistical significance is better than 95% for two and three services co-hosted in favor of self-organizing but in favor of optimal-static omniscient for four services co-hosted whilst being insignificant overall.

In table 6, we illustrate the normalized adaptive revenue stream performance of the self-organizing algorithm against the greedy and optimal-static omniscient algorithms whilst illustrating the statistical significance of the adaptive performance in table 7. The adaptive performance is an aggregate across the four traffic stream types and grouped in terms of demand intensity (ρ) and services co-hosted at the hosting center. Under this performance partitioning scheme, the self-organizing algorithm consistently outperforms the greedy algorithm in the overall category for all four levels of demand intensity with statistical significance better than 99%. In the hosting category, it performs better than the greedy algorithm for high and super high demand intensities with statistical significance better than 99%, whilst its performance is better than 99% significant for low and medium intensities, except when the demand intensity is low and two or four services are hosted or for medium demand intensity when four services are hosted. Against the optimal-static omniscient algorithm, it performs competitively for low, high and super high demand intensities whilst being outperformed for medium intensity in the overall category. However, performance is not statistically significant for low intensity whilst being better than 95% statistically significant for medium, high and super high intensities. In the hosting category, it is outperformed for medium and high demand intensities when four services are hosted whilst performing competitively for all other instances. However, only on the four hosting instance, the result is at least 95% significant.

Table 6. Normalized revenue stream performance of the self-organizing algorithm aggregated across traffic streams (step, heavy, inhomogeneous Poisson and hybrid) where for each co-hosting instance (two, three and four services), $n = 40$ (10 × 4) and for overall, $n = 120$ (10 × 4 × 3).

ρ	Hosting	$\frac{\sum_{i=1}^{n} \text{Self-org}_i}{\sum_{i=1}^{n} \text{Greedy}_i}$	$\frac{\sum_{i=1}^{n} \text{Self-org}_i}{\sum_{i=1}^{n} \text{Opt-stat omni}_i}$
Low	Two-services	0.99	1.01
Low	Three-services	1.04	1.01
Low	Four-services	1.02	1.00
Low	Overall	1.02	1.01
Medium	Two-services	1.04	1.00
Medium	Three-services	1.16	1.00
Medium	Four-services	1.00	0.95
Medium	Overall	1.07	0.98
High	Two-services	1.05	1.02
High	Three-services	1.21	1.02
High	Four-services	1.04	0.99
High	Overall	1.10	1.01
Super high	Two-services	1.05	1.01
Super high	Three-services	1.20	1.03
Super high	Four-services	1.06	1.02
Super high	Overall	1.10	1.02

We qualitatively observed that the self-organizing algorithm exhibits a highly adaptive performance for most conditions in comparison to the greedy algorithm. We also qualitatively observe that its performance against the optimal-static omniscient algorithm is competitive but mixed. We emphasize that the optimal-static omniscient algorithm is theoretical and cannot be implemented in practice as it requires knowledge of the future. We believe that certain traffic stream behaviors coupled with its omniscient property confer an advantage to the optimal-static omniscient algorithm.

We hypothesize that rapidly variable traffic streams (mean interarrival time changes often in relation to downtime) behave

Table 5. Wilcoxon matched-pairs signed-ranks test for self-organizing against greedy and optimal-static omniscient aggregated across traffic streams (step, heavy, inhomogeneous Poisson and hybrid) and demand intensities (low, medium, high and super high) where for each co-hosting instance (two, three and four services), $n = 160$ (10 × 4 × 4) and for overall, $n = 480$ (10 × 4 × 4 × 3).

Wilcoxon matched-pairs signed-ranks test								
	(Self-org)–(Greedy)				(Self-org)–(Opt-stat omni)			
Hosting	W^+	W^-	N	$\leqslant p$-value	W^+	W^-	N	$\leqslant p$-value
Two-services	8951	2677	152	7.978×10^{-9}	7572	3904	151	6.591×10^{-4}
Three-services	12 502	378	160	5.433×10^{-25}	6834	4491	150	2.801×10^{-2}
Four-services	8863	4017	160	3.676×10^{-5}	4528	6948	151	2.465×10^{-2}
Overall	91 823.50	19 802.50	472	6.237×10^{-34}	54 127	48 251	452	2.904×10^{-1}

Table 7. Wilcoxon matched-pairs signed-ranks test for the self-organizing algorithm against the greedy and optimal-static omniscient algorithms aggregated across traffic streams (step, heavy, inhomogeneous Poisson and hybrid) where for each co-hosting instance (two, three and four services), $n = 40$ (10×4) and for overall, $n = 120$ ($10 \times 4 \times 3$).

Wilcoxon matched-pairs signed-ranks test									
		(Self-org)–(Greedy)				(Self-org)–(Opt-stat omni)			
ρ	Hosting	W^+	W^-	N	$\leqslant p$-value	W^+	W^-	N	$\leqslant p$-value
Low	Two-services	278	250	32	8.007×10^{-1}	353	143	31	4.058×10^{-2}
Low	Three-services	717	103	40	3.795×10^{-5}	258	207	30	6.071×10^{-1}
Low	Four-services	535	285	40	9.424×10^{-2}	194	302	31	2.944×10^{-1}
Low	Overall	4429	1849	112	2.414×10^{-4}	2161	2117	92	9.333×10^{-1}
Med	Two-services	672	148	40	4.400×10^{-4}	407	413	40	9.732×10^{-1}
Med	Three-services	809	11	40	8.510×10^{-8}	373	447	40	6.237×10^{-1}
Med	Four-services	380	440	40	6.917×10^{-1}	187	633	40	2.784×10^{-3}
Med	Overall	5595	1665	120	2.683×10^{-7}	2714	4546	120	1.650×10^{-2}
High	Two-services	670	150	40	4.867×10^{-4}	647	173	40	1.479×10^{-3}
High	Three-services	818	2	40	4.328×10^{-8}	588	232	40	1.704×10^{-1}
High	Four-services	686	134	40	2.131×10^{-4}	317	503	40	2.138×10^{-1}
High	Overall	6605	655	120	6.773×10^{-15}	4323	2937	120	6.974×10^{-2}
Super high	Two-services	724	96	40	2.512×10^{-5}	538	282	40	8.657×10^{-2}
Super high	Three-services	814	6	40	5.856×10^{-6}	614	206	40	6.232×10^{-2}
Super high	Four-services	616	124	40	1.243×10^{-4}	543	277	40	7.492×10^{-1}
Super high	Overall	6629	631	120	4.112×10^{-15}	4965	2295	120	4.774×10^{-4}

such that the self-organizing algorithm is penalized for being responsive, that is, adapting to a condition which may only have a short duration. It is better to be unresponsive when conditions are highly dynamic as characterized by the optimal-static omniscient algorithm which makes a single optimal static orchestration decision based on complete knowledge of the future traffic stream arrival behavior and does not attempt to adapt for the rest of the time horizon. Therefore, under highly variable arrival behavior, the optimal-static omniscient algorithm is likely to outperform the self-organizing algorithm. Conversely, if the traffic stream exhibits low variability (a change in mean interarrival time is expected to last) then it is advantageous for the self-organizing algorithm to be responsive and adapt to changes in conditions whilst the optimal-static omniscient will be penalized for being unresponsive.

9. Application of *swarm intelligence:* a small survey

The swarm intelligence methodology has been applied in a variety of domains, with varying degrees of success. Here we briefly survey some of these applications, looking for common patterns that may lead to success or failure.

Schoonderwoerd *et al* [41] developed an ant-colony-inspired pheromone-laying algorithm to select the next path for a message in a telecommunication network. The underlying issue in this system is the selection of a specific path between each source–destination pair so as to balance the load on the different links in the network. Other researchers have explored ant-colony-inspired methods for 'connectionless' network routing problems, such as Internet routing, in which there is no specific path (see [42] for a fuller explanation). Network links can experience unpredictable congestion or failure. The routing problem is even more dynamic in these connectionless domains. Di Caro and Dorigo [43] imitate ant pheromone trail laying and following behavior to dynamically route packets on the Internet. The resulting algorithms are simple and intuitive. Di Caro and Dorigo report that their performance is superior to more complex alternatives in terms of throughput and packet delay. Overall, dynamic network routing appears to be one of the leading successes in swarm intelligence applications. Other promising applications include image clustering and segmentation [44], fault-tolerant vehicle scheduling [45] and factory control/scheduling [46–48].

On the other hand, swarm intelligence has encountered some notable failures. For example, Bonabeau and Meyer [49] report that Enron established a swarm intelligence management model in its energy trading business, which produced a financial disaster of the first magnitude. The ACO (ant colony optimization) traveling salesman algorithms reported in [50–52] are heavily reliant on a different optimization method, neighbourhood search, for their success, but even so they are not remotely competitive with operations research/computer science methods [53–55]. Shortest path algorithms in computer science [56], likewise, outperform any reinforcement or brute force parallel swarm intelligence method by at least an order of magnitude.

What can we learn from these cases? It seems circular to us to assert that swarm intelligence, or biomimicry in general, is more apt to succeed if there is a close homology between the biology and the application, for it begs the question of what similarities make a good match between the two. A web-based banking service seems no more like a flower patch than a raven is like a writing desk. Yet, as demonstrated in this paper, and in the references above, this and the ant-email analogies can be successful.

We suggest that a biologically inspired solution to a problem is appropriate when the essential difficulty of the problem is similar to the challenge in the biological setting. Consider the failure of ACO to compete with conventional methods for the traveling salesman problem. The biological inspiration is the ant colony's problem of finding the shortest path to the prey object. However, the essential computational complexity of the shortest path problem is so vastly different from that of the traveling salesman problem (see [57] or any standard reference on computational complexity for detail) that the analogy is not useful. Why then does ACO also fail, though not so drastically, to compete with conventional shortest path algorithms? The shortest path problem lacks a principal difficulty faced by the ant colony. The ant colony searches for prey objects that are in unknown locations. The unknown state of the environment is a difficulty that has no analog in the classical shortest path problem. In contrast, searching for a short path in a network whose state is uncertain does share the same essential difficulty. Thus, it may be less surprising that ACO would perform competitively on dynamic network routing problems. Note that some ant species do not even behave in analogy to the shortest path problem, because their foragers head directly toward the hive after discovering a prey object, rather than retracing a path.

Consider the history of genetic algorithms (GAs) as another example. Most GAs were computational disasters until Storer *et al* [58] and others discovered how to mimic evolution more thoroughly. Job shop scheduling was one of the first successful GA applications. The problem is particularly difficult because the time window, machine availability and configuration, and precedence constraints require many different portions of a schedule to coordinate. A slight modification to a feasible schedule usually renders it infeasible. In the language of GAs, almost all mutations and crossovers are infeasible. This property stymies most search procedures. Consider now the analogy with biological evolution. One of the great difficulties of evolution is how to maintain viability when one small change in an organism is interrelated with many functions in many parts of the organism. Biological evolution handles this difficulty, at least in part, by not conducting its search in the space of the morphology of the organism itself. Rather, DNA is like a code that produces an organism, and evolutionary search takes place in the space of DNA. For job shop scheduling, the schedule space should not be the search space. Rather, one employs a code, usually a simple heuristic, that produces feasible solutions and conducts the GA search in the space of the code's parameters.

Focusing now on swarm intelligence, we seek patterns in the vicissitudes of its applications. One clue comes from the successes and failures discussed in the non-technical book '*The Wisdom of Crowds*' [59]. From a mathematical view, the point of the book is that the law of large numbers is true. That is, the sample mean of a large number of independent observations converges to the true mean. If a large number of people are asked independently to estimate the weight of an ox or the number of cherries in a bowl, the sample mean is apt to be quite accurate. On the other hand, mobs are infamous for their collective stupidity. A mob's opinion results from self-reinforcing (positive) feedback loops rather than from independent observations. In the Enron case, energy traders recruited hundreds of others to engage in the same activity, but the individual traders did not receive independent objective feedback as to the profitability of their actions. In contrast, each individual packet in a network 'knows' how successful it is in arriving quickly to its correct destination. Despite the stochasticity of the transit times, the aggregate measure, namely the pheromone trail, which mathematically speaking is simply a time-weighted sample mean (time-weighted because the chemical dissipates), is apt to be quite accurate. The same kind of accuracy is obtained through the aggregation of sampling information in our honeybee algorithm, as it is in natural honeybee colonies. See [60, page 789] for a similar insight into aggregation of noisy or stochastic information.

Related to this feature is the presence or absence of varying local information. This can be thought of as varying information in space rather than in time (dynamicism). We suggest that variability in (possibly virtual) space is another feature which tends to make swarm intelligence appropriate as a solution approach. This is because each of the many individual agents can sense and react to its local information, creating a non-homogeneous, locally adaptive policy. For example, in the wireless network domain mobile *ad hoc* networks are spatially and temporally formed by an autonomous collection of mobile devices [61]. If both of these features are present, a swarm intelligence solution can be rapidly adaptive to changing local conditions. Such situations seem particularly appropriate to this approach. Some other prominent swarm intelligence methods also thrive on variability of information. Particle swarm optimization (PSO, see e.g. [62]) is noted for its ability to bypass local minima because different particles tend to be in different locales, but share information.

We suspect that another telling feature in these cases is whether or not the parallelism is real or simulated. In the ACO algorithms for combinatorial optimization problems [52], the insect colony members are simulated on a computer. The algorithms do not actually run on a myriad of small cheap independent processors. In contrast, natural insect swarms do contain thousands or millions of individual insects, communications networks do handle a myriad of individual packets on a large set of links and nodes, and our Internet hosting center does contain many autonomous servers. Stochastic diffusion search (SDS) was developed and is particularly effective for problems in which the objective is separable into many independent components. Different agents in the SDS procedure may evaluate different components. Parallelism is thus inherent in these separable problems. We suggest that swarm intelligence is more likely to be successful if the parallelism is inherent in the system to be optimized, rather than artificially added on.

Finally, we observe that swarm intelligence seems to be more successful when it implements at least one balancing (negative) feedback loop. If swarm intelligence offers only a positive feedback loop, i.e. of the form 'if this activity is good, do more', then it provides only a weak and imprecise thrust toward optimality. For example, positive feedback is an inefficient way to maximize a linear function [63]. In contrast, congestion creates negative feedback loops in networks, so that in an optimal configuration different packets traverse different routes. Similarly, in an optimal server allocation different servers host different web services.

10. Conclusions

In this paper, we described the self-organizing honeybee forager orchestration model and highlighted the resemblance between the forager allocation and the server allocation problems. We probed the homology between honeybee colony forager allocation and Internet host server allocation that inspired a new self-organizing biomimetic server orchestration algorithm for management of Internet hosting centers. We exposed many close parallels between the problems. We also found a qualitative difference which appears at the rather fine level of the second derivatives of the nectar and revenue influx functions. The biomimetic algorithm adapts very well under a variety of circumstances whilst being competitive against benchmark assessment algorithms. We opine that the adaptiveness of the algorithm is due to the tightly coupled homology between the problems in the natural and human domains.

Based on our experience and on a small survey of other cases, we conjecture that the following conditions promote the suitability and success of swarm intelligence applications: inherent parallelism in the problem domain, presence of at least one balancing (negative) feedback loop, noisy or variable data—often temporally or spatially local—that tends to be accurate in the aggregate and the need for rapid and effective responsiveness to greatly changing circumstances.

Acknowledgments

We would like to thank Professor Tom Seeley at Cornell University for providing invaluable insights into the workings of honeybee colonies. We would also like to thank the organizers of the 1st International Symposium for Biologically-Inspired Design and Engineering (2006) at the Georgia Institute of Technology and National Science Foundation for financially supporting the symposium.

References

[1] Nakrani S and Tovey C 2003 *Proc. 2nd Int. Workshop on The Mathematics and Algorithms of Social Insects (Atlanta)* pp 115–22

[2] Arlitt M and Jin T 1999 Workload characterization of the 1998 World Cup web site *Technical Report HPL-1999-35R1 HP Laboratories*

[3] Chase J, Anderson C, Thakar P and Vahdat A 2001 *Proc. 18th ACM Symp. on Operating Systems Principles (SOSP)*

[4] Cherkasova L 1999 *Technical Report HPL-1999-52(R.1) HP Laboratories*

[5] Jayram T, Kimbrel T, Krauthgamer R, Schieber B and Sviridenko M 2001 *Proc. STOC (Hersonissos, Crete, Greece)*

[6] Manasse M S, McGeoch L A and Sleator D D 1988 Competitive algorithm for online problems *20th Annual ACM Symp. on Theory of Computing* pp 323–33

[7] Banga G, Druschel P and Mogul J C 1999 Resource containers: a new facility for resource management in server systems *Proc. 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI)* (New Orleans, LA)

[8] Dusseau A C, Arpaci R H and Culler D E 1996 Effective distributed scheduling of parallel workloads *ACM SIGMETRICS'96 Conf. on the Measurement and Modeling of Computer Systems*

[9] Ford B and Susarla S 1996 CPU inheritance scheduling *Usenix Association Second Symposium on Operating Systems Design and Implementation (OSDI)* pp 91–105

[10] Jones M B, Rosu D and Rosu M 1995 Modular real-time resource management in the Rialto operating system *Technical Reposrt MSR-TR-95-16* Microsoft Research

[11] Jones M B, Rosu D and Rosu M 1997 CPU reservations and time constraints: efficient, predictable scheduling of independent activities *Symp. on Operating Systems Principles* pp 198–211

[12] Waldspurger C A and Weihl W E 1995 Stride scheduling: deterministic proportional-share resource management *Technical Memorandum MIT/LCS/TM-528* MIT Laboratory for Computer Science

[13] Waldspurger C A and Weihl W E 1994 Lottery scheduling: flexible proportional-share resource management *Operating Systems Design and Implementation* pp 1–11

[14] Dusseau A C and Culler D E 1997 Extending proportional-share scheduling to a network of workstations *Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*

[15] Buyya R, Abramson D and Giddy J 2000 Economy driven resource management architecture for computational power grids *Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA2000) (Las Vegas, USA)*

[16] Baker M, Buyya R and Laforenza D 2000 The grid: international efforts in global computing *Proc. Int. Conf. on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (31 July–6 August, L'Aquila, Italy)*

[17] Baratloo A, Dasgupta P and Kedem Z M 1995 CALYPSO: a novel software system for fault-tolerant parallel processing on distributed platforms *Proc. 4th IEEE Int Symp. on High Performance Distributed Computing HPDC-4*

[18] Baratloo A, Itzkovitz A, Kedem Z M and Zhao Y 1998 Just-in-time transparent resource management in distributed systems *Technical Reposrt TR1998-762* Computer Science Department, New York University

[19] Czajkowski K, Foster I T, Karonis N T, Kesselman C, Martin S, Smith W and Tuecke S 1998 A resource management architecture for metacomputing systems *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing* pp 62–82

[20] Dasgupta P, Kedem Z M and Rabin M O 1995 Parallel processing on networks of workstations: a fault-tolerant, high performance approach *Int. Conf. on Distributed Computing Systems*

[21] Hill J M D, Donaldson S R and Lanfear T 1998 Process migration and fault tolerance of BSPlib programs running on networks of workstations *European Conf. on Parallel Processing*

[22] Litzkow M, Tannenbaum T, Basney J and Livny M 1997 Checkpoint and migration of UNIX processes in the condor distributed processing system *Tech. Rep. 1346* Computer Science Department, University of Wisconsin at Maddison

[23] Wolf J L and Yu P S 2001 On balancing the load in a clustered web farm *ACM Trans. Internet Technol.* **1** 231–61

[24] De Farias D P, King A J, Squillante M S and Roy B V 2002 Dynamic control of web server farms *INFORMS Conf.–Revenue Management Section (Columbia University, New York, 13–14 June)*
[25] Palmer J and Mitrani I 2003 Dynamic server allocation in heterogeneous clusters *Technical Report CS-TR: 799* School of Computing Science, University of Newcastle
[26] Appleby K, Fakhouri S, Fong L, Goldszmidt G, Kalantar M, Krishnakumar S, Pazel D P, Pershing J and Rochwerger B 2001 Oceano—SLA based management of a computing utility *7th IFIP/IEEE Int. Symp. on Integrated Network Management*
[27] Seeley T 1995 *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies* (Cambridge, MA: Harvard University Press)
[28] Charnov E L 1976 Optimal foraging: the marginal value theorem *Theor. Pop. Biol.* **9** 129–36
[29] Bonabeau E, Dorigo M and Theraulaz G 1999 *Swarm Intelligence: From Natural to Aritificial Systems* (Oxford: Oxford University Press)
[30] Seeley T, Camazine S and Sneyd J 1991 Collective decision making in honey bees: how colonies choose among nectar sources *Behav. Ecol. Sociol.* **28** 277–90
[31] Bartholdi J III, Seeley T, Tovey C and Vate J 1993 The pattern and effectiveness of nectar forager allocation *J. Theor. Biol.* **160** 23–40
[32] Pai V, Aron M, Banga G, Svendsen M, Druschel P, Zwaenepoel W and Nahum E 1998 Locality-aware request distribution in cluster-based network servers *Proc. 8th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII) (San Jose, CA, USA)*
[33] Nakrani S and Tovey C 2004 On honey bees and dynamic server allocation in internet hosting centers *Adapt. Behav.* **12** 223–40
[34] Little M C 1994 C++SIM. University of Newcastle Upon Tyne. http://cxxsim.ncl.ac.uk/
[35] Nielsen J 2000 *Designing Web Usability* (Indianapolis: New Riders Publishing)
[36] Brown L, Gans N, Mandelbaum A, Sakov A, Shen H, Zeltyn S and Zhao L 2002 Statistical Analysis of a Telephone Call Center: A Queueing-Science Perspective, http://stat.wharton.upenn.edu/∼haipeng/
[37] Crovella M E and Bestavros A 1997 Self-similarity in world wide web traffic: evidence and possible causes *IEEE/ACM Trans. Netw.* **5** 835–46
[38] Gelenbe E (ed) 2000 Internet traffic: periodicity, tail behavior and performance implication *System Performance Evaluation: Methodologies and Application* (Boca Raton, FL: CRC Press) chapter 2
[39] Conover W J 1980 *Practical Nonparametric Statistics* 2nd edn (New York: Wiley) pp 278–92
[40] Haahr M 2005 Random Number Service, http://www.random.org/
[41] Schoonderwoerd R, Holland O, Bruten J and Rothkrantz L 1996 *Technical Report (HPL-96-76)* HP Laboratories
[42] Merloti P 2004 http://www.merlotti.com/EngHome/Computing/AntsSim/ants.htm
[43] Di Caro G and Dorigo M 1998 AntNet: distributed stigmergetic control for communication network *J. Artif. Intell. Res.* **9** 317–65
[44] Ramos V, Muge F and Pina P 2002 Self-organized data and image retrieval as a consequence of inter-dynamic synergistic relationships in artificial ant colonies *Proc. 2nd Int. Conf. on Hybrid Intelligent Systems (Santiago, Chile)*
[45] Bullnheimer B, Hartl R and Strauss C 1997 An improved ant system algorithm for the vehicle routing problem Technical Report POM-10/97 Institute of Management Science, University of Vienna
[46] Cicirello V and Smith S 2001 Ant colony control for autonomous decentralized shop floor scheduling *Proc. 5th Int. Symp. on Autonomous Decentralized Systems (ISAD-2001)*
[47] Cicirello V and Smith S 2001 Improved routing wasps for distributed factory control *Proc. Workshop on AI and Manufacturing (IJCAI-01)*
[48] Cicirello V and Smith S 2001 Wasps nests for self-configurable factories *Proc. 5th Int. Conf. on Autonomous Agents (Agents 2001)*
[49] Bonabeau E and Meyer C 2001 *Harvard Business Review* May edition
[50] Dorigo M and Gambardella L 1997 Ant colonies for the travelling salesman problem *BioSystem* **43** 73–81
[51] Colorni A, Dorigo M and Maniezzo V 1991 Distributed optimization by ant colony *Proc. European Conf. on Artificial Life (ECAL91) (Paris, France)*
[52] Colorni A, Dorigo M, Maffioli F, Maniezzo V, Righini G and Trubian M 1996 Heuristics from nature for hard combinatorial optimization problems *Int. Trans. Oper. Res.* **2** 1–21
[53] Aarts E and Lenstra J (eds) 1997 *Local Search in Combinatorial Optimization* (New York: Wiley) pp 215–310
[54] Applegate D, Bixby R, Chv'atal V and Cook W 1998 The solution of traveling salesman problems *Documenta Mathematica* 645–56
[55] Cook W 2006 Optimal tour of Sweden, available at http://www.tsp.gatech.edu/sweden/index.html
[56] Cormen T, Leiserson C and Rivest R 1990 *Introduction to Algorithms* (Cambridge, MA: MIT Press) pp 329–55
[57] Tovey C A 2002 Tutorial on computational complexity *Interfaces* **32** 30–61
[58] Storer R H, Wu S D and Vaccari R 1992 New search spaces for sequencing problems with applications to job shop scheduling *Manage. Sci.* **37** 10
[59] Surowiecki J 2004 *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations* (New York: Random House)
[60] Passino K 2005 *Biomimicry for Optimization, Control, and Automation* (Berlin: Springer)
[61] Sesay S, Yang Z and He J 2004 A survey on mobile ad hoc wireless network *Inform. Technol.* **3** 168–75
[62] Clerc M 2006 *Particle Swarm Optimization* (London: ISTE)
[63] Goldberg D 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)