System Intelligence: Model, Bounds and Algorithms

Longbo Huang
IIIS@Tsinghua University
longbohuang@tsinghua.edu.cn

ABSTRACT

We present a general framework for understanding system intelligence, i.e., the level of system smartness perceived by users, and propose a novel metric for measuring intelligence levels of dynamical human-in-the-loop systems, defined to be the maximum average reward obtained by proactively serving user demands, subject to a resource constraint. Our metric captures two important elements of smartness, i.e., being able to know what users want and pre-serve them, and achieving good resource management while doing so. We provide an explicit characterization of the system intelligence, and show that it is jointly determined by user demand volume (opportunity to impress), demand correlation (user predictability), and system resource and action costs (flexibility to pre-serve).

We then propose an online learning-aided control algorithm called Learning-aided Budget-limited Intelligent System Control (`LBISC`). We show that `LBISC` achieves an intelligence level that is within $O(N(T)^{-\frac{1}{2}}+\epsilon)$ of the highest level, where $N(T)$ represents the number of data samples collected within a learning period T and is proportional to the user population size in the system, while guaranteeing an $O(\max(N(T)^{-\frac{1}{2}}/\epsilon, \log(1/\epsilon)^2))$ average resource deficit. Moreover, we show that `LBISC` possesses an $O(\max(N(T)^{-\frac{1}{2}}/\epsilon, \log(1/\epsilon)^2)+T)$ convergence time, which is much smaller compared to the $\Theta(1/\epsilon)$ time required for non-learning based algorithms. The analysis of `LBISC` rigorously quantifies the impact of data and user population (captured by $N(T)$), learning (captured by our learning method), and control (captured by `LBISC`) on the achievable system intelligence, and provides novel insight and guideline into designing future smart systems.

1. INTRODUCTION

Due to rapid developments in sensing and monitoring, machine learning, and hardware manufacturing, building intelligence into systems has recently received strong attention, and clever technologies and products have been developed to enhance user experience. For instance, recommendation systems [1], smart home [2], artificial intelligence engines [3], and user behavior prediction [4]. Despite the prevailing success in practice, there has not been much theoretical understanding about system smartness. In particular, how do we measure the intelligence level of a system? How do we compare two systems and decide which one is smarter? What elements in a system contribute most to the level of smartness? Can the intelligence level of a system be pushed arbitrarily high?

Motivated by these fundamental questions, in this paper, we propose a general framework for modeling system smartness and propose a novel metric for measuring system intelligence. Specifically, we consider a discrete time system where a server serves a set of applications of a user. The status of each application changes from time to time, resulting in different demand that needs to be fulfilled. The server observes the demand condition of each application over time and decides at each time whether to *pre-serve* (i.e., serve before the user even place a request) the demand that can come in the next timeslot (which may not be present then), or to wait until the next timeslot and serve it then if it indeed arrives. Depending on whether demand is served passively or proactively, the server receives different rewards representing user's satisfaction levels, or equivalently, different user perception of system smartness (a system that can serve us before being asked is often considered smart). On the other hand, due to time-varying service conditions, the service actions incur different costs. The objective of the server is to design a control policy that achieves the *system intelligence*, defined to be the maximum achievable reward rate subject to a constraint on average cost expenditure.

This model models many examples in practice. For example, newsfeed pushing and video prefetching [5], [6], instant searching [7], and branch prediction in computer architecture [8], [9]. It captures two key elements of a smart system, i.e., being able to know what users want and pre-execute actions, and performing good resource management while doing so. Note that resource management here is critical. Indeed, one can always pre-serve all possible demands to impress users at the expense of very inefficient resource usage, but an intelligent system should do more than that.

Solving this problem is non-trivial. First of all, rewards generated by actions now depend on their execution timing, i.e., before or after requests. Thus, this problem is different from typical network control problems, where outcomes of traffic serving actions are independent of timing. Second, application demands are often correlated over time.

MobiHoc'16, July 04-08, 2016, Paderborn, Germany

DOI: http://dx.doi.org/10.1145/2942358.2942387

Hence, algorithms must be able to handle this issue, and to efficiently explore such correlations. Third, statistical information of the system dynamics are often unknown. Hence, control algorithms must be able to quickly learn and utilize the information, and must be robust against errors introduced in learning.

There have been many previous works studying optimal stochastic system control with resource management. [10] designs algorithms for minimizing energy consumption of a stochastic network. [11] studies the tradeoff between energy and robustness for downlink systems. [12] and [13] develop algorithms for achieving the optimal utility-delay tradeoff in multihop networks. [14] studies the problem of scheduling delay-constrained flows over wireless systems. However, all these works focus only on causal systems, i.e., service begins only after demand enters the system. Recent works [15], [16], [17], [18] and [19] consider queueing system control with future traffic demand information. Works [20] and [21] also consider similar problems where systems can proactively serve user demand. They obtain interesting results that characterize cost reduction under proactive service, and the impact of number of users and prediction in terms of proactive service window size. However, system utilities in the aforementioned works are measured by average metrics, e.g., throughput or outage probabilities, and actions taken at different times for serving traffic are equivalent. Moreover, they do not investigate the impact of user predictability and benefits of learning.

We tackle the problem by first establishing an explicit characterization of the maximum achievable intelligence level. The characterization provides a fundamental limit of system intelligence and reveals that it is jointly determined by user demand volume (opportunity to impress), demand correlation (user predictability), and system resource and control costs (flexibility to pre-serve). Then, by carefully defining effective rewards and costs that represent action outcomes in pairs of slots, we propose an ideal control algorithm that assumes perfect system statistics, called Budget-limited Intelligent System Control (`BISC`).

We further develop Learning-aided `BISC` (`LBISC`), by incorporating a maximum-likelihood-estimator (`MLE`) for estimating statistics, and a dual learning component (`DL`) [22] for learning an empirical Lagrange multiplier that can be integrated into `BISC`, to facilitate algorithm convergence and reduce resource deficit. We show that `LBISC` achieves a system intelligence that can be pushed arbitrarily close to the highest value while ensuring a deterministic budget deficit bound. Furthermore, we investigate the *user-population effect* on algorithm design in system intelligence, and rigorously quantify the degree to which the user population size can impact algorithm performance, i.e., algorithm convergence speed can be boosted by a factor that is proportional to the *square-root* of the number of users. The analysis of `LBISC` quantifies how system intelligence depends on the amount of system resource, action costs, number of data samples, and the control algorithm. To the best of our knowledge, we are the first to propose a rigorous metric for quantifying system intelligence, and to jointly analyze the effects of different factors.

The contributions of this paper are summarized as follows.

- We propose a mathematical model for investigating system intelligence. Our model captures three important components in smart systems including observation (data), learning and prediction (model training), and algorithm design (control).
- We propose a novel metric for measuring system intelligence, and explicitly characterize the optimal intelligence level. The characterization shows that intelligence is jointly determined by system resource and action costs (flexibility to pre-serve), steady-state user demand volume (opportunity to impress), and the degree of demand correlation (predictability, captured by demand *transition rates*).
- We propose an online learning-aided algorithm, called Learning-aided Budget-limited Intelligent System Control (`LBISC`), for achieving maximum system intelligence. Our algorithm consists of a maximum-likelihood-estimator (`MLE`) for learning system statistics, a dual-learning component (`DL`) for learning a control-critical empirical Lagrange multiplier, and an online queue-based controller based on carefully defined effective action rewards and costs.
- We show that `LBISC` achieves an intelligence level that is within $O(N(T)^{-1/2}+\epsilon)$ of the maximum, where T is the algorithm learning time, $N(T)$ is the number of data samples collected in learning and is proportional to the user population of the system, and $\epsilon>0$ is an tunable parameter, while guaranteeing that the average resource deficit is $O(\max(N(T)^{-\frac{1}{2}}/\epsilon, \log(1/\epsilon)^2))$.
- We prove that `LBISC` achieves a convergence time of $O(T+\max(N(T)^{-1/2}/\epsilon, \log(1/\epsilon)^2))$, which can be much smaller than the $\Theta(1/\epsilon)$ time for its non-learning counterpart. The performance results of `LBISC` show that a company with more users has significant advantage over those with fewer, in that its algorithm convergence can be boosted by a factor that is proportional to the *square-root* of the user population.

The rest of the paper is organized as follows. In Section 2, we present a few examples of system smartness. We then present our general model and problem formulation in Section 3, and characterize the optimal system intelligence in Section 4. Our algorithms are presented in Section 5. Analysis is given in Section 6. Simulation results are presented in Section 7, followed by the conclusion in Section 8.

2. EXAMPLES

In this section, we provide a few examples that will serve both as explanations and motivation for our general model.

Instant searching [7]: Imagine you are searching on a search engine. When you start typing, the search engine tries to guess whether you will type in a certain keyword and pre-computes search results that it believes are relevant (predict and pre-service). If the server guesses correctly, results can be displayed immediately after typing is done, and search latency will be significantly reduced, resulting in a great user experience (high reward). If the prediction is inaccurate, the search engine can still process the query after getting the keyword, with the user being less impressed by the performance (low reward) and resources being wasted computing the wrong results (cost).

Video streaming [5]: When a user is watching videos on Youtube or a smart mobile device, the server can predict whether the user wants a particular video clip, and preload the video to the user device (predict and pre-service). This way, if the prediction is correct, user experience will be

greatly improved and he will enjoy a large satisfaction (high reward). If the prediction is incorrect, the bandwidth and energy spent in pre-loading are wasted (cost), but the server can still stream the content video to on the fly, potentially with a degraded quality-of-service (low reward).

Smart home [23]: Consider a smart home environment where a thermostat manages room temperatures in the house. Depending on its prediction about the behavior of hosts, the thermostat can pre-heat/pre-cool some of the rooms (predict and pre-service). If the host enters a room where temperature is already adjusted, he receives a high satisfaction (high reward). If the prediction is incorrect, the room temperature can still be adjusted, but may affect user experience (low reward). Moreover, the energy spent is wasted (cost).

In all these examples, we see that the smart level of a system perceived by users is closely related to whether his demand is served proactively, and whether such predictive service is carried out without too much unnecessary resource expenditure. These factors will be made precise in our general given in the next section.

3. SYSTEM MODEL

We consider a system where a single server is serving a customer with M applications (Fig. 1). Here each application can represent, e.g., a smartphone application, watching a particular video clip, or a certain type of computing task the customer executes regularly. We assume that the system operates in slotted time, i.e., $t \in \{0, 1, ...\}$.

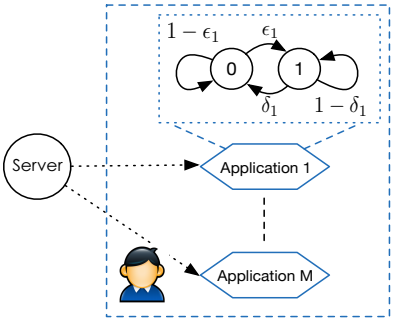

Figure 1: A multi-application system where a server serves a set of applications of a customer.

3.1 The Demand Model

We use $\boldsymbol{A}(t) = (A_m(t), m = 1, ..., M)$ to denote the demand state of the customer at time t. We assume that $A_m(t) \in \{0, 1\}$, where $A_m(t) = 1$ means there is a unit demand at time t and $A_m(t) = 0$ otherwise. For instance, if application m represents a video clip watching task, $A_m(t)$ can denote whether the users wants to watch the video clip in the current slot. If so, the server needs to stream the video clip to the customer's device.

We assume that for each application m, $A_m(t)$ evolves according to an independent two-state Markov chain depicted in Fig. 1, where the transition probabilities ϵ_m and δ_m are as shown in the figure. Note that such an ON/OFF model has been commonly adopted for modeling network traffic states, e.g., [24] and [25].[1] We assume that ϵ_m and δ_m are unknown to the server, but the actual states can be observed every time slot.[2]

3.2 The Service and Cost Model

In every time slot t, the server serves application m's demand as follows. If $A_m(t)$ has yet been served in slot $t-1$, it will be served in the current slot. Otherwise the current demand is considered completed. Then, in addition to serving the current demand, the server can also try to *pre-serve* the demand in time $t+1$. We denote $\mu_{mc}(t) \in \{0, 1\}$ the action taken to serve the current demand and $\mu_{mp}(t) \in \{0, 1\}$ to denote the action taken to serve the demand in time $t+1$.[3] It can be seen that:

$$\mu_{mc}(t) = \max[A_m(t) - \mu_{mp}(t-1), 0]. \quad (1)$$

That is, demand will be fulfilled in the same time slot. Since $\mu_{mc}(t)$ is completely determined by $\mu_{mp}(t-1)$ and $A_m(t)$, we define $\boldsymbol{\mu}(t) \triangleq (\mu_{mp}(t), \forall m)$ for notation simplicity and view $\boldsymbol{\mu}(t)$ as the only control action at time t.

We assume that each service to application m, either proactive or passive, consumes resource of the server. This can be due to energy expenditure or bandwidth consumption. To capture the fact that the condition under which actions are taken can be time-varying, we denote $S_m(t)$ the resource state for application m at time t, which affects how much resource is needed for service, e.g., channel condition of a wireless link, or cost spent for getting a particular video clip from an external server. We denote $\boldsymbol{S}(t) = (S_1(t), ..., S_M(t))$ the overall system resource state, and assume that $\boldsymbol{S}(t) \in \mathcal{S} = \{s_1, ..., s_K\}$ with $\pi_k = \Pr\{\boldsymbol{S}(t) = s_k\}$ and is i.i.d. every slot (also independent of $\boldsymbol{A}(t)$). Here we assume that the server can observe the instantaneous state $\boldsymbol{S}(t)$ and the π_k values are known.[4] To model the fact that a given resource state s_k typically constrains the set of feasible actions, we denote $\mathcal{U}_{\boldsymbol{S}(t)}$ the set of feasible actions under $\boldsymbol{S}(t)$. Examples of $\mathcal{U}_{\boldsymbol{S}(t)}$ include $\mathcal{U}_{s_k} = \{0/1\}^M$ for the unconstrained case, or $\mathcal{U}_{s_k} = \{\boldsymbol{\mu} \in \{0/1\}^M : \sum_m \mu_{mp} \leq N_k\}$ when we are allowed to pre-serve only N_k applications. We assume that $\mathcal{U}_{s_k}$ is compact and if $\boldsymbol{\mu} \in \mathcal{U}_{s_k}$, then a vector obtained by setting any entry to zero in $\boldsymbol{\mu}$ remains in $\mathcal{U}_{s_k}$.

Under the resource state, the instantaneous cost incurred to the server is given by $C(t) = \sum_m C_m(t)$, where

$$C_m(t) \triangleq C_m(\mu_{mc}(t), \boldsymbol{S}(t)) + C_m(\mu_{mp}(t), \boldsymbol{S}(t)). \quad (2)$$

With (2), we assume that the cost in each slot is linear in $\boldsymbol{\mu}(t)$, a model that fits situations where costs for serving applications are additive, e.g., amount of bandwidth required for streaming videos. We assume that $C_m(\cdot, s_k) = 0$

[1]It is possible to adopt a more general multi-state Markov chain for each application. Our results can also likely be extended to model systems where user behavior exhibits periodic patterns. In these scenarios, Markov models can be built for user's behavior in different time periods and can be learned with data collected in those periods.

[2]This is due to the fact that the states essentially denote whether or not the user has requested service from the server. Thus, by observing the user's response we can see the states.

[3]Our results can be extended to having $\mu_{mp}(t) \in [0, 1]$, in which case partial pre-service is allowed.

[4]This assumption is made to allow us to focus on the user demand part. It is also not restrictive, as $\boldsymbol{S}(t)$ is a non-human parameter and can often be learned from observations serving different applications, whereas $\boldsymbol{A}(t)$ is more personalized and targeted learning is needed. Nonetheless, our method also applies to the case when $\{\pi_k, k = 1, ..., K\}$ is unknown.

is continuous, $C_m(0, s_k) = 0$ and $C_m(1, s_k) \leq C_{\max}$ for some $C_{\max} < \infty$ for all k and m. We define

$$C_{\text{av}} \triangleq \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{C(\tau)\} \tag{3}$$

as the average cost spent serving the demand. For notation simplicity, we also denote $\overline{C}_m \triangleq \sum_k \pi_k C_m(1, s_k)$ as the expected cost for serving one unit demand.

3.3 The Reward Model

In each time slot, serving each application demand generates a reward to the server. We use $r_m(t)$ to denote the reward collected in time t from application m. It takes the following values:

$$r_m(t) = \begin{cases} 0 & A_m(t) = 0 \\ r_{mc} & A_m(t) = 1 \ \& \ \mu_{mp}(t-1) = 0 \\ r_{mp} & A_m(t) = 1 \ \& \ \mu_{mp}(t-1) = 1 \end{cases} \tag{4}$$

By varying the values of r_{mp} and r_{mc}, we can model different sensitivity levels of the user to pre-service. We assume that $r_{mp} \geq r_{mc}$ are both known to the server.[5] This is natural for capturing the fact that a user typically gets more satisfaction if his demand is pre-served. We denote $r_{md} = r_{mp} - r_{mc}$ and denote $r_d \triangleq \max_m(r_{mp} - r_{mc})$ the maximum reward difference between pre-service and normal serving.

To evaluate the performance of the server's control policy, we define the following average reward rate, i.e.,

$$r_{\text{av}} = \liminf_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_m \mathbb{E}\{r_m(\tau)\}. \tag{5}$$

r_{av} is a natural index of system smartness. A higher value of r_{av} implies that the server can better predict what the user needs and pre-serves him. As a result, the user perceives a smarter system. In the special case when $r_{mp} = 1$ and $r_{mc} = 0$, the average reward captures the rate of correct prediction.

3.4 System Objective

In every time slot, the server accumulates observations about applications, and tries to learn user preferences and to choose proper actions. We define Γ the set of feasible control algorithms, i.e., algorithms that only choose feasible control actions $\boldsymbol{\mu}(t) \in \mathcal{U}_{\boldsymbol{S}(t)}$ in every time slot, possibly with help from external information sources regarding application demand statistics. For each policy $\Pi \in \Gamma$, we denote $I(\Pi, \rho) = r_{\text{av}}(\Pi)$ and $C_{\text{av}}(\Pi)$ the resulting algorithm intelligence and average cost rate, respectively.

The objective of the system is to achieve the *system intelligence* $I(\rho)$, defined to be the maximum reward rate r_{av} achievable over all feasible policies, subject to a constraint $\rho \in (0, \rho_{\max}]$ on the rate of cost expenditure, i.e.,

$$I(\rho) \triangleq \max: \quad I(\Pi, \rho) \tag{6}$$

$$\text{s.t.} \quad C_{\text{av}}(\Pi) \leq \rho \tag{7}$$

$$\Pi \in \Gamma.$$

[5]This can be done by monitoring user feedbacks, e.g., display a short message and ask the user to provide instantaneous feedback. In the case when they are not known a-priori, they can be learned via a similar procedure as in the LBISC algorithm presented later.

Here $\rho_{\max} \triangleq \sum_k \pi_k \sum_m \sum C_m(1, s_k)$ is the maximum budget needed to achieve the highest level of intelligence.[6]

3.5 Discussions of the Model

In our model, we have assumed that user demand must be served within the same slot it is placed. This is a suitable model for many task management systems where jobs are time-sensitive, e.g., newsfeed pushing, realtime computation, elevator scheduling, video streaming and searching. In these problems, a user's perception about system smartness is often based on whether jobs are pre-served correctly.

Our model captures key ingredients of a general intelligent system including observations, learning and prediction, and control. Indeed, general monitoring and sensing methods can be integrated into the observation part, various learning methods can be incorporated into our learning-prediction step, and control algorithms can be combined with or replace our control scheme presented later.

4. CHARACTERIZING INTELLIGENCE

In this section, we first obtain a characterization of $I(\rho)$. This result provides interesting insight into system intelligence, and provides a useful criteria for evaluating smartness of control algorithms.

To this end, denote $z(t) = (\boldsymbol{A}(t), \boldsymbol{S}(t))$ and $\mathcal{Z} = \{z_1, ..., z_H\}$ the state space of $z(t)$, and denote π_h the steady-state distribution of z_h. Furthermore, define for notation simplicity the transition probability $a_m^{(i_h)}$ as:

$$a_m^{(i_h)} = \begin{cases} 1 - \delta_m & \text{when} \quad i_h = A_m^{(h)} = 1 \\ \epsilon_m & \text{when} \quad i_h = A_m^{(h)} = 0 \end{cases} \tag{8}$$

That is, $a_m^{(i)}$ denotes the probability of having $A_m(t+1) = 1$ in the next time slot given the current state. Then, our theorem is as follows.

Theorem 1. *$I(\rho)$ is equal to the optimal value of the following optimization problem:*

$$\max: \quad \sum_h \pi_h \sum_{j=1}^{3} \theta_j^{(h)} \sum_m a_m^{(i_h)} [\mu_{mpj}^{(h)} r_{mp} + (1 - \mu_{mpj}^{(h)}) r_{mc}] \tag{9}$$

$$\text{s.t.} \quad \sum_h \pi_h \sum_{j=1}^{3} \theta_j^{(h)} \sum_m [C_m(\mu_{mpj}^{(h)}, z_h) + (1 - \mu_{mpj}^{(h)}) a_m^{(i_h)} \overline{C}_m] \leq \rho \tag{10}$$

$$\theta_j^{(h)} \geq 0, \ \sum_j \theta_j^{(h)} = 1, \ \forall\, z_h, j$$

$$\boldsymbol{\mu}_j^{(h)} \in \mathcal{U}_{z_h}, \ \forall\, z_h, j.$$

Here $\theta_j^{(h)}$ represents the probability of adopting the pre-service vector $\boldsymbol{\mu}_j^{(h)}$ under state z_h.

Proof. See Appendix A. □

Theorem 1 states that no matter what learning and prediction methods are adopted and how the system is controlled, the intelligence level perceived by users will not exceed the value in (9). This is a powerful result and provides

[6]This corresponds to the case of always pre-serving user demand. Despite a poor resource utilization, all demands will be pre-served and $I(\rho_{\max}) = \sum_m \frac{\epsilon_m r_{mp}}{\epsilon_m + \delta_m}$ where $\frac{\epsilon_m}{\epsilon_m + \delta_m}$ is the steady-state distribution of having $A_m(t) = 1$.

a *fundamental limit* about the system intelligence. Theorem 1 also reveals some interesting facts and rigorously justifies various common beliefs about system intelligence. (i) When user demands are more predictable (captured by transition rates ϵ_m and δ_m, represented by $a_m^{(i_h)}$ in (9)), the system can achieve a higher intelligence level. (ii) A system with more resources (larger ρ) or better cost management (smaller C_m functions) can likely achieve a higher level of perceived smartness. (iii) When there is more demand from users (captured by distribution π_h), there are more opportunities for the system to impress the user, and to increase the perceived smartness level. The inclusion of transition rates in the theorem shows that our problem can be very different from existing network optimization problems, e.g., [26] and [27], where typically steady-state distributions matter most.

As a concrete example, Fig. 2 shows the $I(\rho)$ values for a single-application system under (i) $\epsilon = \delta$ and (ii) $\delta = 0.6$. We see that in the symmetric case, where the steady-state distribution is always $(0.5, 0.5)$, $I(\rho)$ is inverse-proportional to the entropy rate of the demand Markov chain, which is consistent with our finding that a higher intelligence level is achievable for more predictable systems (lower entropy). For the $\delta = 0.6$ case, $I(\rho)$ first increases, then decreases, and then increases again. The reason is as follows. At the beginning, as ϵ increases, demand increases. Then, when $\epsilon \in [0.4, 0.7]$, $I(\rho)$ is reduced by either the increasing randomness (less predictable) or decreased demand (less opportunity). After that, predictability increases and $I(\rho)$ increases again. This shows that $I(\rho)$ is jointly determined by the steady-state distribution and the transition rates.

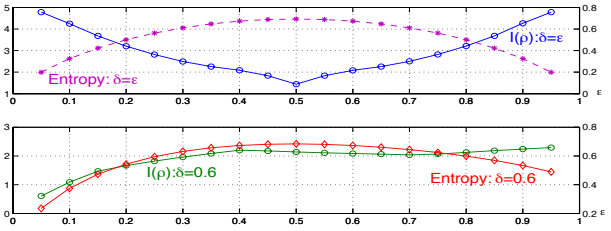

Figure 2: System intelligence ($M = 1$): The left y-axis is for $I(\rho)$ and the right y-axis is for the entropy rate. The x-axis shows the value of ϵ. We see that $I(\rho)$ is consistent with our finding that one can achieve higher intelligence levels for more predictable systems.

Note that solving problem (9) is non-trivial due to the need of system statistics and the potentially complicated structure of $\mathcal{U}_{z_h}$. Thus, in the next section, we propose a learning-based algorithm for solving the optimal control problem. For our algorithm design and analysis, we define the following modified dual function for (9), where $V \geq 1$ is a control parameter introduced for later use:[7]

$$g_{\boldsymbol{\pi}}(\gamma) \triangleq \sum_h \pi_h \sup_{\boldsymbol{\mu}_p^{(h)}} \sum_m \Big\{ V a_m^{(i_h)} [\mu_{mp}^{(h)} r_{mp} + (1-\mu_{mp}^{(h)}) r_{mc}] - \gamma [C_m(\mu_{mp}^{(h)}, z_h) + (1-\mu_{mp}^{(h)}) a_m^{(i_h)} \overline{C}_m - \rho] \Big\}. \quad (11)$$

Here we use the subscript $\boldsymbol{\pi}$ to denote that the dual function is defined with distribution $\boldsymbol{\pi}$. We also use γ^* to denote the minimizer of the dual function. It is shown in [28] that:

$$g_{\boldsymbol{\pi}}(\gamma^*) \geq V \times I(\rho). \quad (12)$$

We also define the dual function for each state z_h as follows:

$$g_h(\gamma) \triangleq \sup_{\boldsymbol{\mu}_p^{(h)}} \sum_m \Big\{ V a_m^{(i_h)} [\mu_{mp}^{(h)} r_{mp} + (1-\mu_{mp}^{(h)}) r_{mc}] - \gamma [C_m(\mu_{mp}^{(h)}, z_h) + (1-\mu_{mp}^{(h)}) a_m^{(i_h)} \overline{C}_m - \rho] \Big\}. \quad (13)$$

It can be seen that $g_{\boldsymbol{\pi}}(\gamma) = \sum_h \pi_h g_h(\gamma)$.

5. ALGORITHM DESIGN

In this section, we present an online learning-aided control algorithm for achieving maximum system intelligence. To facilitate understanding, we first present an ideal algorithm that assumes full information of ϵ_m, δ_m, and π_h. It serves as a building block for our actual algorithm.

5.1 An Ideal Algorithm

To do so, we first define the *effective* reward and cost for each application m as functions of $\boldsymbol{A}(t)$ and $\boldsymbol{\mu}(t)$. Specifically, when $A_m(t) = i$, we have: :

$$\tilde{r}_m^{(i)}(\mu_{mp}(t)) = \begin{cases} a_m^{(i)} r_{mp} & \text{if} \quad \mu_{mp}(t) = 1 \\ a_m^{(i)} r_{mc} & \text{if} \quad \mu_{mp}(t) = 0 \end{cases} \quad (14)$$

Here $a_m^{(i)}$ is defined in (8) as the probability to get into state $A_m(t+1) = 1$ conditioning on the current state being i. To understand the definition, we see that when $A_m(t) = i$, by taking $\mu_{mp}(t) = 0$, the server decides not to pre-serve the potential future demand at $A_m(t+1)$. Hence, if there is demand in slot $t+1$ (happens with probability $a_m^{(i)}$), it will be served by $\mu_{mc}(t+1)$, resulting in a reward of r_{mc}. On the other hand, if $\mu_{mp}(t) = 1$, with probability $a_m^{(i)}$, the future demand will be pre-served and a reward r_{mp} can be collected. It is important to note that the effective reward is defined to be the reward collected in slot $t+1$ as result of actions at time t. We denote $\tilde{r}(\boldsymbol{\mu}(t)) \triangleq \sum_m \tilde{r}_m^{(A_m(t))}(\mu_{mp}(t))$.

Similarly, we define the *effective* cost as a function of $\boldsymbol{\mu}(t)$ for $A_m(t) = i$:

$$\tilde{C}_m^{(i)}(\mu_{mp}(t)) = \begin{cases} C_m(1, \boldsymbol{S}(t)) & \text{if} \quad \mu_{mp}(t) = 1 \\ a_m^{(i)} \sum_k \pi_k C_m(1, s_k) & \text{if} \quad \mu_{mp}(t) = 0 \end{cases} \quad (15)$$

Note that $\tilde{C}_m(\mu_{mp}(t))$ is the expected cost spent in slots t and $t+1$. As in the effective reward case, we denote $\tilde{C}(\boldsymbol{\mu}(t)) \triangleq \sum_m \tilde{C}_m^{(A_m(t))}(\mu_{mp}(t))$.

With the above definitions, we introduce a *deficit* queue $d(t)$ that evolves as follows:

$$d(t+1) = \max[d(t) + \tilde{C}(t) - \rho, 0], \quad (16)$$

with $d(0) = 0$.Then, we define a Lyapunov function $L(t) \triangleq \frac{1}{2} d^2(t)$ and define a single-slot sample-path drift $\Delta(t) \triangleq L(t+1) - L(t)$. By squaring both sides of (16), using $(\max[x, 0])^2 \leq x^2$ for all $x \in \mathbb{R}$, and $\tilde{C}(\boldsymbol{\mu}(t)) \leq M C_{\max}$, we obtain the following inequality:

$$\Delta(t) \leq B - d(t)[\rho - \tilde{C}(t)]. \quad (17)$$

Here $B \triangleq \rho_{\max}^2 + M^2 C_{\max}^2$. Adding to both sides the term $V \sum_m \tilde{r}_m(\mu_{mp}(t))$, where $V \geq 1$ is a control parameter, we

[7] Notice that although (11) does not include the $\theta_j^{(h)}$ variables. It can be shown to be equivalent. Moreover, (11) is sufficient for our algorithm design and analysis.

obtain:

$$\Delta(t) - V\tilde{r}(\boldsymbol{\mu}(t)) \leq B - \Big(V\tilde{r}(\boldsymbol{\mu}(t)) + d(t)[\rho - \tilde{C}(\boldsymbol{\mu}(t))]\Big). \quad (18)$$

Having established (18), we construct the following ideal algorithm by choosing pre-service actions to minimize the right-hand-side of the drift.

Budget-limited Intelligent System Control (BISC): At every time t, observe $\boldsymbol{A}(t)$, $\boldsymbol{S}(t)$ and $d(t)$. Do:

- For each m, define the cost-differential as follows:

$$D_m(t) \triangleq C_m(1, \boldsymbol{S}(t)) - a_m^{(i)}\overline{C}_m. \quad (19)$$

  Here $i = A_m(t)$. Then, solve the following problem to find the optimal pre-serving action $\boldsymbol{\mu}(t)$:

$$\max: \sum_m \mu_{mp}(t)[Va_m^{(i)}(r_{mp} - r_{mc}) - d(t)D_m(t)] \quad (20)$$
$$\text{s.t. } \boldsymbol{\mu}(t) \in \mathcal{U}_{\boldsymbol{S}(t)}.$$

- Update $d(t)$ according to (16). $\diamond$

A few remarks are in place. (i) The value $a_m^{(i)}(r_{mp} - r_{mc})$ can be viewed as the expected reward loss if we choose $\mu_{mp}(t) = 0$ and the value $D_m(t)$ is the expected cost saving for doing so. The parameter V and $d(t)$ provide proper weights to the terms for striking a balance between them. If the weighted cost saving does not overweight the weighted reward loss, it is more desirable to pre-serve the demand in the current slot. (ii) For applications where $a_m^{(i)}(r_{mp} - r_{mc})$ is smaller, it is less desirable to pre-serve the demand, as the user perception of system intelligence may not be heavily affected. (iii) In the special case when $\rho \geq \rho_{\max}$, we see that $d(t)$ will always stay near zero, resulting in $\mu_{mp}(t) = 1$ most of the time. (iv) BISC is easy to implement. Since each $\mu_{mp}(t)$ is either 0 or 1, problem (20) is indeed finding the maximum-weighted vector from $\mathcal{U}_{\boldsymbol{S}(t)}$. In the case when $\mathcal{U}_{\boldsymbol{S}(t)}$ only limits the number of non-zero entries, we can easily sort the applications according to the value $Va_m^{(i)}(r_{mp} - r_{mc}) - d(t)D_m(t)$ and choose the top ones.

5.2 Learning-aided Algorithm with User Population Effect

In this section, we present an algorithm that learns δ_m and ϵ_m online and performs optimal control simultaneously. We also explicitly describe how the system user population size can affect algorithm performance.

To rigorously quantify this user effect, we first introduce the following *user-population effect* function $N(T)$.

Definition 1. *A system is said to have a user population effect $N(T)$ if within T slots, (i) it collects a sequence of demand samples $\{\boldsymbol{A}(0), ..., \boldsymbol{A}(N(T)-1)\}$ generated by the Markov process $\boldsymbol{A}(t)$, and (ii) $N(T) \geq T$ for all T.* $\diamond$

$N(t)$ captures the number of useful user data samples a system can collect in T time slots, and is a natural indicator about how user population contributes to learning user preferences. For instance, if there is only one user, $N(T) = T$. On the other hand, if a system has many users, it can often collect samples from similar users (often determined via machine learning techniques, e.g., clustering) to study a target user's preferences. In this case, one typical example for $N(T)$ can be:

$$N(T) = f(\#\text{ of user}) \cdot T, \quad (21)$$

where $f(\#\text{ of user})$ computes the number of similar users that generate useful samples.

Now we present our algorithm. We begin with the first component, which is a maximum likelihood estimator (MLE) [29] for estimating user demand statistics.[8]

Maximum Likelihood Estimator (MLE(T)): Fix a learning time T and obtain a sequence of samples $\{\boldsymbol{A}(0), ..., \boldsymbol{A}(N(T)-1)\}$ in $[0, T-1]$. Output:

$$\hat{\epsilon}_m(T) = \frac{\sum_{t=0}^{N(T)-1} 1_{\{A_m(t)=0, A_m(t+1)=1\}}}{\sum_{t=0}^{T-1} 1_{\{A_m(t)=0\}}} \quad (22)$$

$$\hat{\delta}_m(T) = \frac{\sum_{t=0}^{N(T)-1} 1_{\{A_m(t)=1, A_m(t+1)=0\}}}{\sum_{t=0}^{T-1} 1_{\{A_m(t)=1\}}}. \quad (23)$$

That is, use empirical frequencies to estimate the transition probabilities. $\diamond$

Note that after estimating $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{\delta}}$, we also obtain an estimation of $\hat{\boldsymbol{\pi}}$. We now have the second component, which is a *dual learning* module [22] that learns an empirical Lagrange multiplier based on $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{\delta}}$, and $\hat{\boldsymbol{\pi}}$.

Dual Learning (DL($\hat{\boldsymbol{\epsilon}}$, $\hat{\boldsymbol{\delta}}$, $\hat{\boldsymbol{\pi}}$)): Construct $\hat{g}_{\hat{\boldsymbol{\pi}}}(\gamma)$ with $\hat{\boldsymbol{\epsilon}}$, $\hat{\boldsymbol{\delta}}$, and $\hat{\boldsymbol{\pi}}$ according to (11). Solve the following problem and outputs the optimal solution γ_T^*.

$$\min: \hat{g}_{\hat{\boldsymbol{\pi}}}(\gamma), \text{ s.t. } \gamma \geq 0. \diamond \quad (24)$$

Here $\hat{g}_{\hat{\boldsymbol{\pi}}}(\gamma)$ is the dual function with true statistics being replaced by $\hat{\boldsymbol{\epsilon}}$, $\hat{\boldsymbol{\delta}}$, and $\hat{\boldsymbol{\pi}}$. With MLE(T) and DL($\hat{\boldsymbol{\epsilon}}$, $\hat{\boldsymbol{\delta}}$, $\hat{\boldsymbol{\pi}}$), we have our learning-aided BISC algorithm.[9]

Learning-aided BISC (LBISC(T, θ)): Fix a learning time T and perform the following:[10]

- (Estimation) For $t = 0, ..., T-1$, choose any $\boldsymbol{\mu}_p(t) \in \mathcal{U}_{S(t)}$ At time T, perform MLE(T) to obtain $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{\delta}}$, and $\hat{\boldsymbol{\pi}}$.
- (Learning) At time T, apply DL($\hat{\boldsymbol{\epsilon}}$, $\hat{\boldsymbol{\delta}}$, $\hat{\boldsymbol{\pi}}$) and compute γ_T^*. If $\gamma_T^* = \infty$, set $\gamma_T^* = V\log(V)$. Reset $d(T) = 0$.
- (Control) For $t \geq T$, run BISC with $\hat{\boldsymbol{\pi}}$, $\hat{\boldsymbol{\epsilon}}$ and $\hat{\boldsymbol{\delta}}$, and with effective queue size $\tilde{d}(t) = d(t) + (\gamma_T^* - \theta)^+$. $\diamond$

Here θ (to be specified) is a tuning parameter introduced to compensate for the error in γ_T^* (with respect to γ^*). It is interesting to note that LBISC includes three important functions in system control, namely, estimation (data), learning (training) and control (algorithm execution). It also highlights three major sources that contribute to making a system non-intelligent: lack of data samples, incorrect training and parameter tuning, and inefficient control algorithms. An intelligent system requires all three to provide good user experience and to be considered smart (Thus, if a search engine does not provide good performance for you at the beginning, it may not be because its algorithm is bad).

6. PERFORMANCE ANALYSIS

[8]Here we adopt MLE to demonstrate how learning can be rigorously and efficiently combined with control algorithms to achieve good performance. Any alternative estimator that possesses similar features as MLE can also be used here.

[9]The methodology can be applied to the case when r_{mp} and r_{mc} are also unknown.

[10]The main reason to adopt a finite T is for tractability. In actual implementation, one can continuously refine the estimates for $\hat{\boldsymbol{\epsilon}}$, $\hat{\boldsymbol{\delta}}$, and $\hat{\boldsymbol{\pi}}$ over time.

In this section, we analyze the performance of LBISC. We focus on three important performance metrics, i.e., achieved system intelligence, budget guarantee, and algorithm convergence time. The optimality and convergence analysis is challenging. In particular, the accuracy of the MLE estimator affects the quality of dual-learning, which in turn affects algorithm convergence and performance. Thus, the analysis must simultaneously take into account all three components.

Throughout our analysis, we make the following assumptions, in which we use $\hat{g}_{\hat{\boldsymbol{\pi}}}(\gamma)$ to denote the dual function in (11) with different $\boldsymbol{\epsilon}$, $\boldsymbol{\delta}$ and $\boldsymbol{\pi}$ values.

Assumption 1. *There exists a constant $\nu = \Theta(1) > 0$ such that for any valid state distribution $\boldsymbol{\pi}' = (\pi'_1, ..., \pi'_H)$ with $\|\boldsymbol{\pi}' - \boldsymbol{\pi}\| \leq \nu$, there exist a set of actions $\{\boldsymbol{\mu}_j^{(h)}\}_{j=1,2,3}$ with $\boldsymbol{\mu}_j^{(h)} \in \mathcal{U}_{z_h}$ and some variables $\{\theta_j^{(h)} \geq 0\}_{j=1,2,3}$ with $\sum_j \theta_j^{(h)} = 1$ for all z_h (possibly depending on $\boldsymbol{\pi}'$), such that:*

$$\sum_h \pi'_h \sum_{j=1}^{3} \theta_j^{(h)} \sum_m [C_m(\mu_{mpj}^{(h)}, z_h) + (1 - \mu_{mpj}^{(h)}) a_m^{(i_h)} \overline{C}_m] \leq \rho_0,$$

where $\rho_0 \triangleq \rho - \eta > 0$ with $\eta = \Theta(1) > 0$ is independent of $\boldsymbol{\pi}'$. Moreover, for all transition probabilities $\boldsymbol{\epsilon}'$ and $\boldsymbol{\delta}'$ with $\|\boldsymbol{\epsilon}' - \boldsymbol{\epsilon}\| \leq \nu$ and $\|\boldsymbol{\delta}' - \boldsymbol{\delta}\| \leq \nu$, ρ satisfies:

$$\rho \geq \sum_m \max[\epsilon'_m \overline{C}_m, (1 - \delta'_m)\overline{C}_m]. \ \diamond \tag{25}$$

Assumption 2. *There exists a constant $\nu = \Theta(1) > 0$ such that, for any valid state distribution $\boldsymbol{\pi}' = (\pi'_1, ..., \pi'_H)$ with $\|\boldsymbol{\pi}' - \boldsymbol{\pi}\| \leq \nu$, and transition probabilities $\boldsymbol{\epsilon}'$ and $\boldsymbol{\delta}'$ with $\|\boldsymbol{\epsilon}' - \boldsymbol{\epsilon}\| \leq \nu$ and $\|\boldsymbol{\delta}' - \boldsymbol{\delta}\| \leq \nu$, $\hat{g}_{\hat{\boldsymbol{\pi}}}(\gamma)$ has a unique optimal solution $\gamma^* > \mathbf{0}$ in $\mathbb{R}$.* $\diamond$

These two assumptions are standard in the network optimization literature, e.g., [30] and [31]. They are necessary conditions to guarantee the budget constraint and are often assumed with $\nu = 0$. In our case, having $\nu > 0$ means that systems that are alike have similar properties. (25) is also not restrictive. In fact, $\sum_m [\pi_{m0} \epsilon'_m \overline{C}_m + \pi_{m1}(1 - \delta'_m)\overline{C}_m]$ (π_{mi} is the steady-state probability of being in state i for m) is the overall cost without any pre-service. Hence, (25) is close to being a necessary condition for feasibility.

We now have the third assumption, which is related to the structure of the problem. To state it, we have the following system structural property introduced in [13].

Definition 2. *A system is polyhedral with parameter $\beta > 0$ under distribution $\boldsymbol{\pi}$ if the dual function $g_{\boldsymbol{\pi}}(\gamma)$ satisfies:*

$$g_{\boldsymbol{\pi}}(\gamma) \geq g_{\boldsymbol{\pi}}(\gamma^*) + \beta \|\gamma^* - \gamma\|. \ \diamond \tag{26}$$

Assumption 3. *There exists a constant $\nu = \Theta(1) > 0$ such that, for any valid state distribution $\boldsymbol{\pi}' = (\pi'_1, ..., \pi'_H)$ with $\|\boldsymbol{\pi}' - \boldsymbol{\pi}\| \leq \nu$, and transition probabilities $\boldsymbol{\epsilon}'$ and $\boldsymbol{\delta}'$ with $\|\boldsymbol{\epsilon}' - \boldsymbol{\epsilon}\| \leq \nu$ and $\|\boldsymbol{\delta}' - \boldsymbol{\delta}\| \leq \nu$, $\hat{g}_{\hat{\boldsymbol{\pi}}}(\gamma)$ is polyhedral with the same β.* $\diamond$

The polyhedral property often holds for practical systems, especially when control action sets are finite (see [13] for more discussions).

6.1 System Intelligence and Budget

We first present the performance of LBISC in system intelligence and budget guarantee. The following theorem summarizes the results.

Theorem 2. *Suppose the system is polyhedral with $\beta = \Theta(1) > 0$. By choosing $\theta = \max(\frac{V \log(V)^2}{\sqrt{N(T)}}, \log(V)^2)$ and a sufficiently large V, with probability at least $1 - 2Me^{-\log(V)^2/4}$, LBISC achieves:*

- *Budget:*

$$\tilde{d}(t) \leq d_{\max} \triangleq V r_d / \hat{D}_{\min} + M C_{\max}, \ \forall \, t \tag{27}$$

$$\overline{d(t)} = O(\max(\frac{V \log(V)^2}{\sqrt{N(T)}}, \log(V)^2)). \tag{28}$$

  Here $\overline{d(t)} = \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{d(t)\}$ and $\hat{D}_{\min} = \Theta(1)$. (27) implies $C_{av}(\Pi) \leq \rho$.

- *System intelligence:*

$$I(\texttt{LBISC}, \rho) \geq I(\rho) - \frac{B_1 + 1}{V} - \max_{z_h} \frac{e_{\max}(\overline{T}_h^2 - \overline{T}_h)}{2V\overline{T}_h}. \tag{29}$$

  Here $B_1 \triangleq B + 2M(V r_d + d_{\max} C_{\max}) \log(V)/\sqrt{N(T)}$, $e_{\max} \triangleq (MC_{\max} + \rho)^2$, and $\overline{T}_h$ and $\overline{T}_h^2$ are the first and second moments of return times of state z_h. $\diamond$

Proof. Omitted due to space limitation. Please see our technical report [32]. $\square$

Theorem 2 shows that LBISC achieves an $[O(N(T)^{-\frac{1}{2}} + \epsilon), O(\max(N(T)^{-\frac{1}{2}}/\epsilon, \log(1/\epsilon)^2)]$ intelligence-budget tradeoff (taking $\epsilon = 1/V$), and the system intelligence level can be pushed arbitrarily close to $I(\rho)$ under LBISC. Thus, by varying the value of V, one can tradeoff the intelligence level loss and the budget deficit as needed. Although Theorem 2 appears similar to previous results with learning, e.g., [22], its analysis is different due to (i) the Markov nature of the demand state $\boldsymbol{A}(t)$, and (ii) the learning error in both transition rates in (20) and the distribution.

6.2 Convergence Time

We now look at the convergence speed of LBISC, which measures how fast the algorithm learns the desired system operating point. To present the results, we adopt the following definition of convergence time from [22] to our setting.

Definition 3. *Let $\zeta > 0$ be a given constant. The ζ-convergence time of a control algorithm, denoted by T_ζ, is the time it takes for $\tilde{d}(t)$ to get to within ζ distance of γ^*, i.e., $T_\zeta \triangleq \inf\{t : |\tilde{d}(t) - \gamma^*| \leq \zeta\}$.* $\diamond$

The intuition behind Definition 3 is as follows. Since the LBISC algorithm is a queue-based algorithm, the algorithm will starting making optimal choice of actions once $\tilde{d}(t)$ gets close to γ^*. Hence, T_ζ naturally captures the time it takes for LBISC to converge. We now present our theorem. For comparison, we also analyze the convergence time of BISC.[11]

Theorem 3. *Suppose the conditions in Theorem 2 hold. Under LBISC, with probability at least $1 - 2Me^{-\log(V)^2/4}$,*

$$\mathbb{E}\{T_{D_1}^{\texttt{LBISC}}\} = O(\max(\frac{V \log(V)^2}{\sqrt{N(T)}}, \log(V)^2)) + T, \tag{30}$$

[11] Here the slower convergence speed of BISC is due to the fact that it does not utilize the information to perform learning-aided control. We present this result to highlight the importance of learning in control.

$$\mathbb{E}\{T_{D_2}^{\texttt{BISC}}\} = \Theta(V). \quad (31)$$

Here $D_1 = O(V\log(V)/\sqrt{N(T)} + D)$ *with* $D = \Theta(1)$ *and* $D_2 = \Theta(1)$.

PROOF. See Appendix B. □

Here the reason why D_1 may be larger than D_2 is due to the fact that LBISC uses inaccurate estimates of $a_m^{(i)}$ for making decisions. In the case when $N(T) = T$, we can recover the $O(V^{2/3})$ convergence time results in [22] by choosing $T = V^{2/3}$. Theorem 3 also shows that it is possible to achieve faster convergence if a system has a larger population of users from which it can collect useful samples for learning the target user quickly. It also explicitly quantifies the speedup factor to be proportional to the *square-root* of the user population size, i.e., when $N(T) = N * T$ where N is the number of users, the speedup factor is $\sqrt{N(T)}/\sqrt{T} = \sqrt{N}$.

This result reveals the interesting fact that a big company that has many users naturally has advantage over companies with smaller user populations, since they can collect more useful data and adapt to a "smart" state faster.

7. SIMULATION

We now present simulation results for BISC and LBISC. We simulate a three-application system ($M = 3$) with the following setting. $(r_{1p}, r_{2p}, r_{3p}) = (3, 5, 8)$ and $r_{mc} = 1$ for all m. Then, we use $\boldsymbol{\epsilon} = (0.6, 0.5, 0.3)$ and $\boldsymbol{\delta} = (0.2, 0.6, 0.5)$. The channel state space is $\mathcal{S} = \{1, 2\}$ for all m, with $\Pr\{S_1(t) = 1\} = 0.5$, $\Pr\{S_2(t) = 1\} = 0.3$, and $\Pr\{S_3(t) = 1\} = 0.3$. The service cost is given by $C_m(1, \boldsymbol{S}(t)) = S_m(t)$. We simulate the system for $T_{sim} = 10^5$ slots, with $V = \{5, 10, 20, 50, 100\}$. For LBISC, we simulate a user population effect function as in (21), i.e., $N(T) = f(\#\text{ of user}) \cdot T$ and choose $f(\#\text{ of user}) = 2, 5, 8$. We also fix the value $\rho = 3.5$ and choose the learning time $T = V^{2/3}$.

We first present Fig. 3 that shows $I(\rho)$ as a function of ρ. For comparison, we include a second setting, where we change $(r_{1p}, r_{2p}, r_{3p}) = (4, 5, 3)$, $\boldsymbol{\epsilon} = (0.8, 0.4, 0.3)$, $\boldsymbol{\delta} = (0.2, 0.9, 0.5)$, and $\Pr\{S_2(t) = 1\} = 0.8$. It can be seen that $I(\rho)$ first increases as ρ increases. Eventually ρ becomes more than needed after all the possible predictability has been exploited. Then, $I(\rho)$ becomes flat. This *diminishing return* property is consistent with our understanding obtained in Section 4.

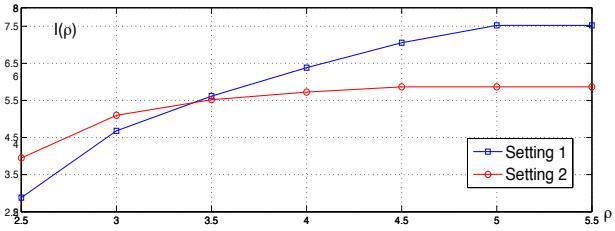

Figure 3: $I(\rho)$ versus ρ: in the two settings tested, $I(\rho)$ are both concave increasing in ρ.

We now look at algorithm performance. From Fig. 4 we see that both BISC and LBISC are able to achieve high intelligence levels. Moreover, LBISC does much better in controlling the deficit (2×-4× saving compared to BISC). We remark here that BISC assumes full knowledge beforehand, while LBISC learns them online.

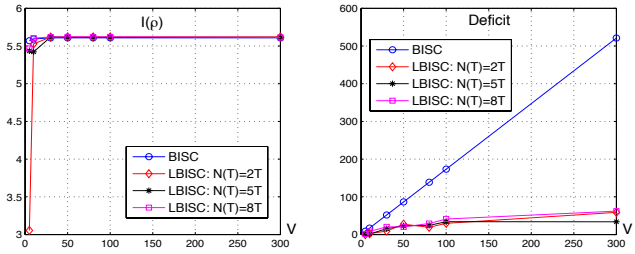

Figure 4: Intelligence and deficit performance of BISC and LBISC with different user-population effect.

Finally, we look at the convergence properties of the algorithms. Fig. 5 compares BISC and LBISC with $N(t) = 8T$ and $V = 300$. We see that LBISC converges at around 460 slots, whereas BISC converges at around 920 slots, resulting in a 2× improvement. Moreover, the actual deficit level under LBISC is much smaller compared to that under BISC (80 versus 510, a 6× improvement). From this result, we see that it is important to efficiently utilize the data samples collected over time, and dual learning provides one way to boost algorithm convergence.

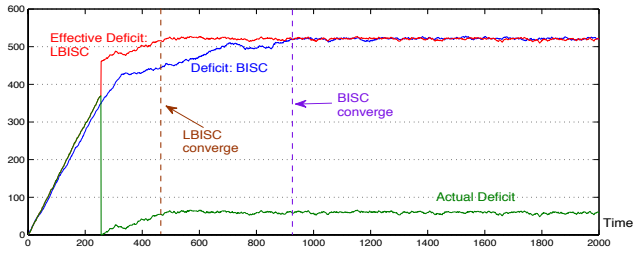

Figure 5: Convergence of BISC and LBISC with $N(T) = 8T$ for $V = 300$.

8. CONCLUSION

In this paper, we present a general framework for defining and understanding system intelligence, and propose a novel metric for measuring the smartness of dynamical systems, defined to be the maximum average reward rate obtained by proactively serving user demand subject to a resource constraint. We show that the highest system intelligence level is jointly determined by system resource, action costs, user demand volume, and correlation among demands. We then develop a learning-aided algorithm called Learning-aided Budget-limited Intelligent System Control (LBISC), which efficiently utilizes data samples about system dynamics and achieves near-optimal intelligence, and guarantees a deterministic deficit bound. Moreover, LBISC converges much faster compared to its non-learning based counterpart.

9. ACKNOWLEDGEMENT

This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61361136003, 61303195, Tsinghua Initiative Research Grant, Microsoft Research Asia Collaborative Research Award, and the China Youth 1000-talent Grant.

10. REFERENCES

[1] G. A. Gomez-Uribe and N. Hunt. The netflix recommender system: Algorithms, business value, and

innovation. *ACM Trans. on Management Information Systems, Vol. 6, No. 4*, Dec 2015.

[2] M. M. Gear. How to make the amazon echo the center of your smart home. *Wired, http://www.wired.com/2016/01/iot-cookbook-amazon-echo/*, Jan 2016.

[3] N. Benaich. Investing in artificial intelligence. *TechCrunch http://techcrunch.com/2015/12/25/investing-in-artificial-intelligence/*, Dec 2015.

[4] I. Weber and A. Jaimes. Who uses web search for what? and how? *Web Search and Data Mining (WSDM), pages 21-30*, 2011.

[5] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review, Volume 26, Issue 3, Pages 22-36*, July 1996.

[6] J. Lee, H. Kim, and R. Vuduc. When prefetching works, when it doesn't, and why. *ACM Transactions on Architecture and Code Optimization (TACO), Volume 9, Issue 1*, March 2012.

[7] M. Marrs. Predictive search: Is this the future or the end of search? *WordStream, http://techcrunch.com/2015/12/25/investing-in-artificial-intelligence/*, June 2013.

[8] T. Ball and J. R. Larus. Branch prediction for free. *Proceedings of the Conference on Programming Language Design and Implementation, ACM SIGPLAN Notices, volume 28, pages 300-13*, 1993.

[9] M. U. Farooq, Khubaib, and L. K. John. Store-load-branch (slb) predictor: A compiler assisted branch prediction for data dependent branches. *Proceedings of the 19th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, February 2013.

[10] M. J. Neely. Energy optimal control for time-varying wireless networks. *IEEE Transactions on Information Theory 52(7): 2915-2934*, July 2006.

[11] C. W. Tan, D. P. Palomar, and M. Chiang. Energy-robustness tradeoff in cellular network power control. *IEEE/ACM Transactions on Networking, Vol. 17, No. 3, pp. 912-925*, 2009.

[12] M. J. Neely. Super-fast delay tradeoffs for utility optimal fair scheduling in wireless networks. *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Nonlinear Optimization of Communication Systems*, 24(8):1489–1501, Aug. 2006.

[13] L. Huang and M. J. Neely. Delay reduction via Lagrange multipliers in stochastic network optimization. *IEEE Trans. on Automatic Control*, 56(4):842–857, April 2011.

[14] I. Hou and P.R. Kumar. Utility-optimal scheduling in time-varying wireless networks with delay constraints. *Proceedings of ACM MobiHoc*, 2010.

[15] J. Tadrous, A. Eryilmaz, and H. El Gamal. Proactive resource allocation: harnessing the diversity and multicast gains. *IEEE Tansactions on Information Theory*, 2013.

[16] J. Spencer, M. Sudan, and K Xu. Queueing with future information. *ArXiv Technical Report arxiv:1211.0618*, 2012.

[17] S. Zhang, L. Huang, M. Chen, and X. Liu. Proactive serving reduces user delay exponentially. *Proceedings of ACM Sigmetrics (Poster Paper)*, 2014.

[18] K. Xu. Necessity of future information in admission control. *Operations Research*, 2015.

[19] L. Huang, S. Zhang, M. Chen, and X. Liu. When Backpressure meets Predictive Scheduling. *Proceedings of ACM MobiHoc*, 2014.

[20] J. Tadrous and A. Eryilmaz. On optimal proactive caching for mobile networks with demand uncertainties. *IEEE/ACM Transactions on Networking*, To appear.

[21] J. Tadrous, A. Eryilmaz, and H. El Gamal. Proactive data download and user demand shaping for data networks. *IEEE/ACM Transactions on Networking, vol. 23, no. 6, pp. 1917-1930*, December 2015.

[22] L. Huang, X. Liu, and X. Hao. The power of online learning in stochastic network optimization. *Proceedings of ACM Sigmetrics*, 2014.

[23] M. Shann and S. Seuken. An active learning approach to home heating in the smart grid. *Proceedings of IJCAI*, 2013.

[24] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. on Networking, Vol. 2, pp. 1-15*, 1994.

[25] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. *ACM IMC*, 2010.

[26] L. Ying, S. Shakkottai, and A. Reddy. On combining shortest-path and back-pressure routing over multihop wireless networks. *Proceedings of IEEE INFOCOM*, April 2009.

[27] M. J. Neely, E. Modiano, and C. Li. Fairness and optimal stochastic control for heterogeneous networks. *IEEE/ACM Trans. on Networking, vol. 16, no. 2, pp. 396-409*, April 2008.

[28] L. Huang and M. J. Neely. Max-weight achieves the exact $[O(1/V), O(V)]$ utility-delay tradeoff under Markov dynamics. *arXiv:1008.0200v1*, 2010.

[29] J. Walrand. *Probability in Electrical Engineering and Computer Science*. Amazon, 2014.

[30] A. Eryilmaz and R. Srikant. Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control. *IEEE/ACM Trans. Netw.*, 15(6):1333–1344, 2007.

[31] X. Lin and N. B. Shroff. The impact of imperfect scheduling on cross-layer congestion control in wireless networks. *IEEE/ACM Trans. on Networking*, 2006.

[32] L. Huang. System intelligence: Model, bounds and algorithms. *arXiv technical report, arXiv:1605.02585*, 2016.

[33] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar. *Convex Analysis and Optimization*. Boston: Athena Scientific, 2003.

Appendix A – Proof of Theorem 1

We prove Theorem 1 here.

PROOF. (Theorem 1) Consider any control scheme Π and fix a time T. Define the joint state $z(t) = (\boldsymbol{A}(t), \boldsymbol{S}(t))$ and

denote its state space as $\mathcal{Z} = \{z_1, ..., z_H\}$. Then, consider a state z_h and let $\mathcal{T}_h(T)$ be the set of slots with $z(t) = z_h$ for $t = 1, ..., T$. Denote $\{\boldsymbol{\mu}_p(0), ..., \boldsymbol{\mu}_p(T)\}$ the pre-service decisions made by Π. Denote the following joint reward-cost pair:[12]

$$(Reward^{(h)}(T), Cost^{(h)}(T)) \triangleq$$
$$\frac{1}{T}\sum_{\tau=0}^{T-1}\sum_{m}\mathbb{E}\big\{I_{[A_m(\tau+1)=1]}[\mu_{mp}(\tau)r_{mp} + (1-\mu_{mp}(\tau))r_{mc}];$$
$$C_m(\mu_{mp}(\tau), z_h) + (1-\mu_{mp}(\tau))I_{[A_m(\tau+1)=1]}\overline{C}_m \mid z(\tau) = z_h\big\}.$$

Notice that this is a mapping from z_h to a subset in $\mathbb{R}^2$ and that both the reward and cost are continuous. Also note that $\mathbb{E}\big\{I_{[A_m(\tau+1)=1]}\big\} = a_m^{(i_h)}$, where $i_h = A_m^{(h)}$ is defined in (14) to denote the probability of having $A_m(\tau+1) = 1$ given $A_m(\tau) = A_m^{(h)}$. Using the independence of $\boldsymbol{A}(t)$ and $\boldsymbol{S}(t)$, and Caratheodory's theorem [33], it follows that there exists three vectors $\boldsymbol{\mu}_j^{(h)}(T) \in \mathcal{U}_{z_h}, j = 1, 2, 3$, with appropriate weights $\theta_j^{(h)}(T) \geq 0, j = 1, 2, 3$, and $\sum_i \theta_j^{(h)}(T) = 1$, so that:

$$(Reward^{(h)}(T), Cost^{(h)}(T)) \tag{32}$$
$$\triangleq \sum_{j=1}^{3}\theta_j^{(h)}(T)\sum_{m}\big(a_m^{(i_h)}[\mu_{mpj}^{(h)}(T)r_{mp} + (1-\mu_{mpj}^{(h)}(T))r_{mc}];$$
$$C_m(\mu_{mpj}^{(h)}(T), z_h) + (1-\mu_{mpj}^{(h)}(T))a_m^{(i_h)}\overline{C}_m\big).$$

Now consider averaging the above over all z_h states. We get:

$$(Reward_{av}(T), Cost_{av}(T)) \triangleq$$
$$\sum_h \pi_h \sum_{j=1}^{3}\theta_j^{(h)}(T)\sum_{m}\big(a_m^{(i_h)}[\mu_{mpj}^{(h)}(T)r_{mp} + (1-\mu_{mpj}^{(h)}(T))r_{mc}];$$
$$C_m(\mu_{mpj}^{(h)}(T), z_h) + (1-\mu_{mpj}^{(h)}(T))a_m^{(i_h)}\overline{C}_m\big).$$

Using a similar argument as in the proof of Theorem 1 in [10], one can show that there exist limit points $\theta_j^{(h)} \geq 0$ and $\boldsymbol{\mu}_j^{(h)}$ as $T \to \infty$, so that the reward-cost tuple can be expressed as:

$$(Reward_{av}, Cost_{av})$$
$$= \sum_h \pi_h \sum_{j=1}^{3}\theta_j^{(h)}\sum_{m}\big(a_m^{(i_h)}[\mu_{mpj}^{(h)}r_{mp} + (1-\mu_{mpj}^{(h)})r_{mc}];$$
$$C_m(\mu_{mpj}^{(h)}, z_h) + (1-\mu_{mpj}^{(h)})a_m^{(i_h)}\overline{C}_m\big).$$

This shows that for an arbitrarily control algorithm Π, its average reward and cost can be expressed as those in problem (9). Hence, its budget limited average reward cannot exceed Φ, which is the optimal value of (9). This shows that $\Phi \geq I(\rho)$.

The other direction $I(\rho) \geq \Phi$ will be shown in the analysis of the LBISC algorithm, where we show that LBISC achieves an intelligence level arbitrarily close to Φ. □

Appendix B – Proof of Theorem 3

We will use of the following technical lemmas for our proof.

[12]To save space, here we use the notation $\mathbb{E}\{X; Y|A\}$ to denote $(\mathbb{E}\{X|A\}, \mathbb{E}\{Y|A\})$.

Lemma 1. *With probability at least* $1 - 2Me^{-\log(V)^2/4}$, DL *outputs a* γ_T^* *that satisfies* $|\gamma_T^* - \gamma^*| \leq d_\gamma$ *where* $d_\gamma \triangleq \frac{cV\log(V)}{\sqrt{N(T)}}$ *and* $c = \Theta(1) > 0$. *Moreover,* $|\gamma^* - \hat{\gamma}^*| \leq d_\gamma$.

Proof. Omitted due to space limitation. Please see our technical report [32]. □

Lemma 2. *[33] Let* $\mathcal{F}_n$ *be filtration, i.e., a sequence of increasing* σ*-algebras with* $\mathcal{F}_n \subset \mathcal{F}_{n+1}$. *Suppose the sequence of random variables* $\{y_n\}_{n\geq 0}$ *satisfy:*

$$\mathbb{E}\big\{||y_{n+1} - y^*|| \mid \mathcal{F}_n\big\} \leq \mathbb{E}\big\{||y_n - y^*|| \mid \mathcal{F}_n\big\} - u_n, \tag{33}$$

where u_n *takes the following values:*

$$u_n = \begin{cases} u & \text{if } ||y_n - y^*|| \geq D, \\ 0 & \text{else.} \end{cases} \tag{34}$$

Here $u > 0$ *is a given constant. Then, by defining* $N_D \triangleq \inf\{k \mid \|y_n - y^*\| \leq D\}$, *we have:*

$$\mathbb{E}\big\{N_D\big\} \leq ||y_0 - y^*||/u. \diamond \tag{35}$$

We now present the proof.

Proof. (Theorem 3) To start, note that the first T slots are spent learning $\hat{\epsilon}$, $\boldsymbol{\delta}$, and $\hat{\boldsymbol{\pi}}$. Lemma 1 shows that after T slots, with high probability, we have $|\gamma_T^* - \gamma^*| \leq \frac{cV\log(V)}{\sqrt{N(T)}}$ for some $c = \Theta(1)$. Using the definition of $\tilde{d}(t)$, this implies that when V is large,

$$|\tilde{d}(T) - \gamma^*| \leq \theta/2 = \max(V\log(V)^2/\sqrt{N(T)}, \log(V)^2)/2. \tag{36}$$

Using Eq. (40) in the proof of Theorem 2 in [32], and applying Lemma 2, we see then the expected time for $\tilde{d}(t)$ to get to within some $D = \Theta(1)$ of $\hat{\gamma}^*$, denoted by $\tilde{T}_D$, satisfies:

$$\mathbb{E}\big\{\tilde{T}_D\big\} \leq N_1 \max(V\log(V)^2/\sqrt{N(T)}, \log(V)^2)/2\eta_1. \tag{37}$$

Here N_1 and η_1 are both $\Theta(1)$ constants.

Since $|\gamma^* - \hat{\gamma}^*| \leq d_\gamma = cV\log(V)/\sqrt{N(T)}$, by defining $D_1 = cV\log(V)/\sqrt{N(T)} + D$, we conclude that:

$$\mathbb{E}\big\{T_{D_1}^{\texttt{LBISC}}\big\} \leq N_1 \max(\frac{V\log(V)^2}{2\eta_1\sqrt{N(T)}}, \frac{\log(V)^2}{2\eta_1}) + T. \tag{38}$$

In the case of BISC, one can similarly show that there exists $\Theta(1)$ constants N_1, D (only related to $z(t)$ and β), and η_2, so that,

$$\mathbb{E}\big\{|d(t+N_1) - \gamma^*| \mid d(t)\big\} \leq |d(t) - \gamma^*| - \eta_2. \tag{39}$$

Since $\gamma^* = \Theta(V)$ [13] and $d(0) = 0$, we conclude that:

$$\mathbb{E}\big\{T_D^{\texttt{LBISC}}\big\} \leq N_1 V/\eta_2. \tag{40}$$

This completes the proof of the theorem. □