Dynamic load balancing with learning model for Sudoku solving system

doi: 10.1016/j.dcan.2019.03.002

Digital Communications and Networks(DCN)

HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/dcan

Dynamic load balancing with learning model for Sudoku solving system

Nattapong Kitsuwan, Praphan Pavarangkoon, Hendro Mulyo Widiyanto, and Eiji Oki

Department of Computer and Network Engineering, The University of Electro-Communications, 1-5-1, Chofugaoka, Chofi-shi, Tokyo 182-8585, Japan

Abstract

This paper proposes a dynamic load balancing with learning model for a Sudoku problem solving system that has multiple workers and multiple solvers. The objective is to minimise the total processing time of problem solving. Our load balancing with learning model distributes each Sudoku problem to an appropriate pair of worker and solver when it is received by the system. The information of estimated solution time for a specific number of given input values, estimated finishing time of each worker, and idle status of each worker are used to determine the worker-solver pairs. In addition, the proposed system can estimate the waiting period for each problem. Test results show that the system has shorter processing time than conventional alternatives.

KEYWORDS: Dynamic load balancing, learning model, Sudoku

1. Introduction

Sudoku is a logic-based combinatorial number-placement puzzle. It contains $d^2 \times d^2$ cells in table form. The table consists of d^2 minigrids, where each minigrid contains $d \times d$ cells none of which overlap, as shown in Fig. 1. A popular table format is $d = 3$. Solving a Sudoku puzzle demands that every cell be filled with a number value so that three conditions are satisfied: first, the number values in each cell on the same row must be different: second, the number values in each cell in the same column must be different; finally, the number values in the same minigrid must be different.

There are basically two approaches to solving Sudoku problems. First, we can formulate Sudoku as an Integer Linear Programming (ILP) problem, which can allows ILP solvers to be applied. The second is to adopt one of the non-ILP approaches as in Refs. [1] - [8]. Human strategies [1], such as naked pair and x-wing, can be applied as non-ILP approaches. The human strategies may be faster than the ILP approach if the pattern of the problem matches the human strategy adopted. This approach is not very effective as several strategies may need to be tried to solve one Sudoku problem, and the problem cannot be solved if the pattern does not match any strategy. Human strategies need to be combined with ILP or non-ILP approaches to ensure problem solution.

Existing Sudoku solvers consider only calculation time in managing the queue of the load balancer, queuing time on the load balancer, and time to solve the Sudoku problem. If the load balancer and the Sudoku server are spatially separated, the link propagation delay of the network influences queuing performance. Therefore, considering the link propagation delay can make the load balancing system more efficient.

The basic Sudoku solver consists of an input interface and a worker with a solver. Sudoku solvers were developed in Refs. [9] - [13]. The website [9] provides 785 free Sudoku puzzles and allows users to create Su-

*Nattapong Kitsuwan (Corresponding author) (email:kitsuwan@uec.ac.jp).

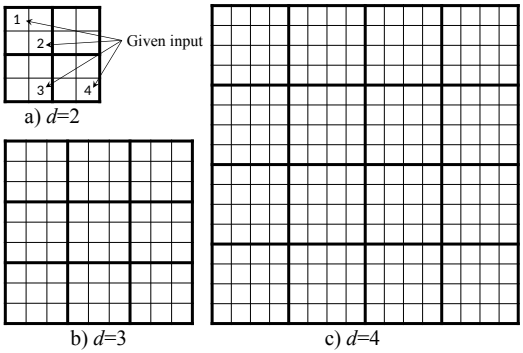


Fig. 1: Sudoku tables

doku problems by themselves. Limits are placed on the number of given input values, as the website does not yield puzzles with a small number of given input values. Another website in Ref. [10] provides a Command Line Interface (CLI) to a Sudoku solver. The user can use the system to generate a problem and determine whether an answer is a unique solution. A system to solve Sudoku puzzles with $d = 4$ was presented in Ref. [11]. The output shows multiple solutions (if available) and steps in the solutions. Spreadsheet software [12] has been presented to solve $d = 5$ Sudoku. That work includes a system that allows the user to define Sudoku size [13].

All the above Sudoku solvers systems employ pipelining to solve Sudoku problems one by one. If there are many requests, however, parallel solving is required to achieve practical processing time. Therefore, Sudoku solving systems should employ multiple workers. The performance of a solving approach is based on the characteristics of the problem. Some approaches may find solutions rapidly when the number of given input values is small. For reducing the solving time, an attractive alternative is to equip each worker with several approaches and to choose the most suitable approach for each request. Such systems need load balancing to distribute requests to appropriate worker-solver pairs. We consider user requirements as follows: first, the user wants to obtain a solution as quickly as possible; second, the user wants an estimation of time taken to obtain a solution before obtaining the solution; third, the user wants to define the given input values without any restriction; fourth, the user wants to define Sudoku size. None of the previous proposals can satisfy all requirements. The works in Refs. [9] - [13] satisfy only the first, third and/or fourth requirements. However, when the size of the Sudoku problem increases, the systems take too long to solve each problem. The user does not know when the solution will be obtained, and so does not know how long he/she has to wait.

This paper proposes a dynamic load balancing with learning model for a Sudoku solving system that satisfies all four requirements detailed above. It is an extended version of our previous work in Ref. [14] in which the system combined multiple solvers with just a single worker. The extended version has an architecture that combines multiple workers with multiple solvers to simultaneously solve requests of multiple users. In addition, its load balancing manager distributes the requests to appropriate worker-solver pairs. The processing time includes calculation time and queuing time in the load balancing manager, round trip time between the load balancing manager and the selected worker, and time spent by the selected worker to solve the puzzle. The specifications of the load balancing manager, such as CPU and memory, influence the calculation time and the queuing time, while those of each worker affect the time spent by each worker. The proposed dynamic load balancing with learning model can reduce the processing time if the system receives multiple requests. The model takes into account the performance of each solver as a function of the number of input values, estimated completion time, and idle status of each worker. The estimated completion time is the sum of the current time and the estimated waiting period. Since the completion time is estimated by the learning model, it can be shown to the user before solving the problem. The proposed system also supports every Sudoku solving approach. Note that modification of conventional solving approaches is not required. Finally, we implement and demonstrate a SudokuWeb system.

The rest of the paper is organised as follows: section 2 reviews existing Sudoku solving approaches: section 3 illustrates the architecture of the Sudoku system used in this work: section 4 describes conventional load balancing models: section 5 presents the proposed load balancing with learning model: section 6 evaluates the performance of the proposed model. Finally, Section 7 concludes this paper.

2. Sudoku solving approaches

A Sudoku problem contains blank cells and filled cells. The values in the filled cells are called the given values. The goal is to fill in all of the blank cells without contravening any of the rules. The general rules of the Sudoku are described as follows:

- Only one value can be entered into each blank cell.
- Every cell must be filled.
- Each row must contain each value exactly once.
- Each column must contain each value exactly once.
- Each minigrid must contain each value exactly once.

There are two main approaches to solving a Sudoku puzzle. The first approach is based on Integer Linear Programming (ILP). The second approach is based on algorithms.

2.1. ILP approach

This approach formulates the Sudoku problem as a mathematical model [15]. The $n \times n$ Sudoku puzzle contains $d \times d$ minigrids, where $n = d^2$. The input number from 1 to n is validated in all cells.

The notations are as follows: let i and j be row and column indices, respectively; let k be a value for cell (i, j); let n be the number of rows or columns of the Sudoku puzzle; and let m be the number of rows or columns of a minigrid. The decision variable, x_{ijk}, is defined by:

$$x_{ijk} = \begin{cases} 1 \text{ if cell } (i, j) \text{ contains integer } k \\ 0 \text{ otherwise,} \end{cases} \quad (1)$$

where $i, j, k \in D$. D is a set of values 1 through n, and $D = \{1, ..., n\}$. Let G be a set of (i, j, k), where k is a given value for cell (i, j). Let P be a set of a, where $a = 1 + (d \times b)$ and $0 \leq b \leq d - 1$. For example, P of 9×9 and 16×16 Sudoku puzzles are $\{1, 4, 7\}$ and $\{1, 5, 9, 13\}$, respectively.

Constraints are as follows:

$$x_{ijk} = 1, \forall (i, j, k) \in G \quad (2)$$

$$\sum_{k=1}^{n} x_{ijk} = 1, \forall i, j \in D \quad (3)$$

$$\sum_{i=1}^{n} x_{ijk} = 1, \forall j, k \in D \quad (4)$$

$$\sum_{j=1}^{n} x_{ijk} = 1, \forall i, k \in D \quad (5)$$

$$\sum_{i=1}^{I+(d-1)} \sum_{i=1}^{J+(d-1)} x_{ijk} = 1, \forall k \in D, I \in P, J \in P \quad (6)$$

$$x_{ijk} \in \{0, 1\}, \forall i, j, k \in D. \quad (7)$$

Eq. (2) gives the condition that specifies the Sudoku problem. $x_{ijk} = 1$ for $(i, j, k) \in G$ is set. Eq. (3) specifies that element (i, j) has only one number from 1 to n. Eqs. (4) and (5) specify that number $k \in D$ appears once in each column and row, respectively. Eq. (6) specifies that number $k \in D$ appears once in each $d \times d$ minigrid. Eq. (7) limits the value of $x_{i,j,k}$ to be either 0 or 1. It should be noted that the Sudoku problem finds a feasible solution that satisfies all the constraints, and we do not intend to maximise or minimise any value. Therefore, no objective function must be defined [15].

The generalised Sudoku problem is NP-Complete [16]. The goal is to find at least one feasible solution that satisfies all of the constraints. If the size of Sudoku is large, the complexity of the ILP computations increases, and it becomes virtually impossible to solve it in a practical time. Some algorithmic approaches were introduced to overcome this weakness.

2.2. Non-ILP approach

The non-ILP approach is to use a suite of algorithms to determine a suitable solution in reasonable time. Several algorithms are usually employed as at least one of them is expected to output a solution, if the problem is solvable. In some conditions, the non-ILP approach can solve Sudoku problems faster than the ILP approach. Examples of the non-ILP approach are the brute-force algorithm and the backtracking algorithm [2].

The brute-force algorithm visits the empty cells from left to right, and from top to bottom by assigning possible values that comply with the rules. At first, the brute-force finds the first empty cell to fill. It fills the empty cell with the lowest possible value. It repeats the same process in the next empty cell. If the value conflicts the rules, it resets all values and refills the first empty cell by increasing the value. The algorithm is iterated until all empty cells are filled.

The backtracking algorithm incrementally builds candidates to the solutions. Its difference from the brute-force algorithm is that the assigned values are not reset when a rule conflict occurs. The backtracking algorithm traces back to the previous step and increases the value by one. The process is repeated until all cells are filled. The backtracking algorithm can yield solutions more rapidly than the brute-force algorithm.

The efficiency of the backtracking algorithm has been investigated in Ref. [2]. The backtracking algorithm has been improved by using minigrid-based backtracking in Ref. [3]. It considers 3×3 minigrids (instead of blank cells in isolation), and uses pre-processing to calculate all valid permutations for each minigrid based on the clues in a given Sudoku puzzle. Instead of considering the individual (blank) cells, it uses minigrids to find only the valid solutions of a given Sudoku puzzle. It can reduce the time taken to find solutions. In general, both the brute-force and the backtracking algorithms are not guaranteed to terminate within polynomial time. The time taken depends on the number of trace-back steps. If the number of trace-back steps becomes large, the time taken increases. In other words, if the same number of given input values are distributed differently across the Sudoku puzzle, the time taken is not guaranteed to be the same. As a result, it is difficult to predict or understand the puzzle difficulties based on the solution time.

Both ILP and non-ILP approaches have advantages and disadvantages. The ILP solver, which is an optimization software package, achieves shorter solution time than the non-ILP approach when the number of given values is large. In contrast, the non-ILP approach is faster when the number of given values is small.

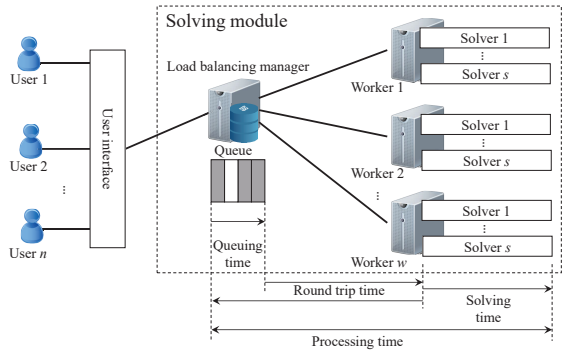


Fig. 2: Proposed SudokuWeb system architecture

3. Sudoku solving system architecture

The Sudoku solving system is a web-based system that satisfies the four user requirements of obtaining solutions quickly, getting estimates of the waiting period, and freedom in defining given input values and Sudoku size. It consists of an interface and a solving module. The interface may be implemented using a Graphic User Interface (GUI) or a CLI. Functions of the interface are collecting requests from users, submitting the requests to the solving module, returning estimates of waiting period, and displaying the results to the users. The solving module consists of a load balancing manager and w workers, as shown in Fig. 2. The load balancing manager uses distributed load balancing with learning model. Each worker employs s solvers, and each solver adopts a different approach. The process of the system is as follows.

- Step 1: A Sudoku problem request is sent by a user from the interface to the solving module.
- Step 2: The load balancing manager in the solving module receives the request and selects the worker-solver pair that has the earliest estimated finishing time by using information held in the database.
- Step 3: The load balancing manager informs the user of the estimated finishing time, and forwards the problems to the selected worker-solver pair.
- Step 4: The selected worker-solver pair processes the request, and sends the result back to the user interface.

3.1. Interface module

The user interface of the system shows the estimated waiting period to the user before returning the solution. It allows users to interact with the load balancing manager. Developed as a web page, it mainly consists of user inputs and result fields to show the solution or error messages. The time taken depends on the Sudoku size and the number of given input values. The result page presents the estimated waiting period to the user. This provides better user experience in terms of awareness, because the user knows when the solution is expected to be returned.

3.2. Solving module

The solving module consists of workers, solvers, and a load balancing manager. The optimal worker-solver pair is selected by the load balancing manager to solve the Sudoku problem. The load balancing manager works as a centralised controller. It receives requests from users, and distributes them to the appropriate worker-solver pairs based on a load balancing with learning model. Once it selects the worker-solver pair, it estimates the processing time and informs the user of this time. Upon receiving solutions from the worker-solver pairs, the load balancing manager forwards them to the corresponding users.

4. Conventional load balancing models

Load balancing models can be divided into two categories based on the decision making approach adopted: static or dynamic. In static load balancing models, the work load distribution is pre-determined and remains the same. Several static load balancing models have been introduced in Refs. [24] - [27]. Round robin is an example of static load balancing model. Each worker is indexed at the beginning. The requests are distributed among the workers in the ascending order of index number. If the workers are heterogeneous in terms of performance, a worker with higher performance may be idle for some time until it is selected. The worker has to wait until the other worker in the previous turn finishes processing the request. As a result, the waiting period may be long if the idle time is large.

Dynamic load balancing models use current information of the workers, e.g., CPU performance and latency, in deciding request distributions. The load balancing manager may dynamically change the order when distributing requests. Hwang and Jung [20] have discussed how to define the load of each worker, and how to determine load limits. In their model, the load balancer has a weight table that is updated according to changes in the loads, and the distribution of service requests is controlled according to the table. Two hierarchical dynamic models were presented by Barazandeh and Mortazavi [22]. The first method, namely the biasing process, is used to allocate weights. Biases are determined based on the current load state of the groups. The second method improves the round-robin algorithm so that the group with the minimum load state takes priority over others in being assigned tasks by the load balancing manager in the specific time. The third method, throttled load balancing, was introduced by James and Verma [21]. In this method, availability and processing speed are considered as performance attributes of the worker. At each timeslot, the load balancing system distributes a request to the

worker that has the best performance. The idle time of workers is taken in account, so the queuing time can be reduced.

Load balancing for a web-server system was introduced to distribute incoming user requests among several workers in Ref. [23]. This model consists of load balancing and a set of workers and supports Local Area Network (LAN) and Wide Area Network (WAN) operations. The processing time includes calculation time and queuing time at the load balancing manager, round trip time between the load balancing manager and the worker, and time taken by the worker. Distributing workloads by considering the processing time is a popular approach. The processing time may be reduced by using these load balancing models. In fact, only the queuing time is reduced. The time taken can be reduced only by upgrading the workers.

5. Proposed load balancing with learning model

Our load balancing with learning model selects the worker-solver pair that is estimated to be the fastest in solving the input Sudoku puzzle. The processing time of each worker-solver pair differs with the number of given input values. The model learns from previous Sudoku puzzles the processing time of each worker-solver for each number of given input values. Then it calculates the estimated finishing time, which is the summation of the current time and estimated processing time, for a given Sudoku puzzle for each worker-solver pair. The combination that has the earliest finishing time is selected.

The average processing time is estimated by averaging the processing time of worker-solver pairs for a specific number of given input values in the database. Let (i, j) be the pairing of worker i and solver j. Let $A^d_{i,j}$ be the average processing time of (i, j) for $d^2 \times d^2$ table.

The finishing time of each (i, j) is also estimated. Let $S_{i,j}$ be the start timestamp of (i, j). The estimated finishing time is $S_{i,j} + A^d_{i,j}$ if the status of (i, j) is busy. It is the summation of the current timestamp and $A^d_{i,j}$ if the status of (i, j) is idle. (i, j) with the earliest estimated finishing time is selected to solve the Sudoku puzzle. The estimated finishing time is passed to the user. It should be noted that the estimated finishing time may be expressed in terms of duration.

Figure 3 shows an example of selecting a worker-solver pair based on the load balancing model. There are three workers. Each worker employs two solvers. The model estimates the processing time for $A^d_{i,j}$. $A^d_{1,1}$ is 12 sec, $A^d_{1,2}$ is 5 sec, $A^d_{2,1}$ is 21 sec, $A^d_{2,2}$ is 14 sec, $A^d_{3,1}$ is 13 sec, and $A^d_{3,2}$ is 8 sec. Although $A^d_{1,2}$ has the lowest processing time, $(1, 2)$ is not selected because the estimated finishing time is not the earliest. Worker 2 is idle but the finishing time is also not the earliest, so $(2, 1)$ and $(2, 2)$ are not selected. The model selects

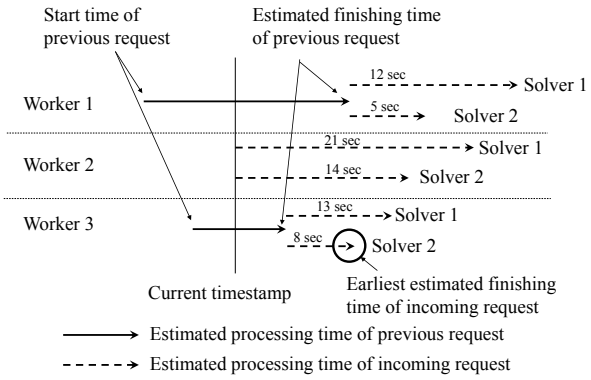


Fig. 3: Example of worker and solver pairing

$(3, 2)$ to solve the incoming request, for it has the earliest finishing time.

	Size	No. given	Worker ID	Approach ID	Start time	Processing time	
①	9	4	1	2	1468563334.4062	0.212893	Average 0.25157
②	9	4	1	2	1468563334.8221	0.290248	
③	9	4	1	2	1468563335.2268	0.244278	0.2491396
④	9	4	1	2	1468564248.0516		

Fig. 4: Database structure and example of data

The load balancing manager is provided with a database. The database keeps the information of Sudoku size, the number of given input values, worker ID, approach ID, the starting timestamp, and processing time, as shown in Fig. 4. It should be noted that the start timestamp is the current Unix timestamp with units of microseconds. At first, the information in the database is prepared as follows: first, Sudoku problems with several given input values $(1, 2, 3, \cdots, n^2 - 1)$ are generated. The pattern of the given input values is random. Second, the problems are solved using every worker-solver pair. The starting timestamp, the processing time of each combination with given size and the number of given input values are entered in the database.

Figure 4 shows an example of the data when the Sudoku size is nine and the number of given values is four for worker ID 1 with approach ID 2. In this example, the data in the first two records are initially prepared by running two different Sudoku problems. When a user submits a Sudoku puzzle and the load balancing manager determines that the request should be solved by worker ID 1 with approach ID 2, the request is sent to worker ID 1. At that time, a new record, which is the third record, is added to the database with the information of Sudoku size, the number of given input values, worker ID, approach ID, and the starting timestamp. The load balancing manager estimates the time taken by averaging the processing time of the first and second records. This time is sent back to the user interface without being entered into the database. After the load balancing manager receives the result of the problem from the worker, the

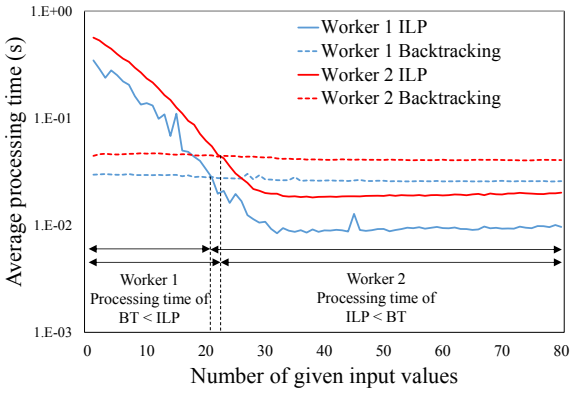


Fig. 5: Comparison of processing time of both workers with both approaches ($d = 3$).

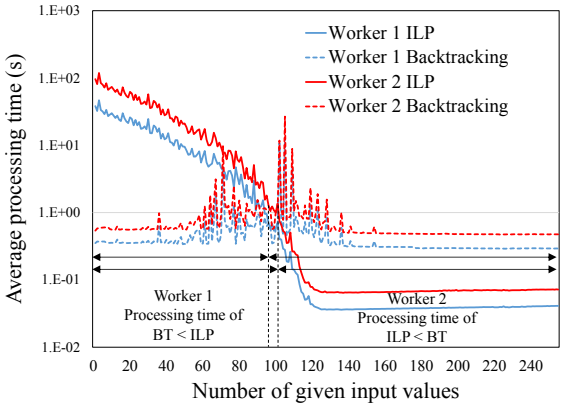


Fig. 6: Comparison of processing time of both workers with both approaches ($d = 4$).

processing time is written into the third record, and the result is forwarded to the user interface. If the load balancing manager determines that another new request with the same size and number of given values should be solved by worker ID 1 with approach ID 2, the fourth record is added to the database using the same process. For this request, the time taken is estimated by averaging the processing time from the first to the third record. Therefore, the time taken will dynamically change every time there is a request.

6. Performance Evaluation

We used the processing time of our testbed to evaluate the performance of the proposed system. The testbed currently has two workers, each with two solvers: the ILP approach and the backtracking algorithm. The results of the proposed system are compared with those of multiple workers with a single solver, using ILP or backtracking. Round robin or throttled [21] are taken as the conventional load balancing model. A computer with Intel Xeon CPU E5-2603 v3 @ 1.60GHz is used as the load balancing manager. A computer with Intel CoreTM i7-2600K CPU@3.40 GHz is used as worker 1. A computer with AMD PhenonTM II×4 955 Processor is used as worker 2. Both workers ran the ILP approach and the backtracking algorithm. Sudoku with $d = 3$ and $d = 4$ are used in the evaluation. It is assumed that one worker cannot solve multiple Sudoku puzzles simultaneously. No other job is running on the worker, so that the CPU is used only for the solvers. It should be noted that all computers in the experiment had different specifications, because we wanted to investigate the performance of the proposed model in a realistic scenario.

At the initial stage, 80 and 255 given input values were created for $d = 3$ and $d = 4$, respectively. For each given input value, 1,000 different patterns were randomly generated. Therefore, 80,000 and 255,000 Sudoku puzzles were created for $d = 3$ and $d = 4$, respectively. The puzzles were solved using all combinations of worker and solver. The processing time of each combination was captured in the database. Note that this information is used for subsequent estimation. Figures 5 and 6 compare the average processing time of each combination of worker and solver for each given input value for $d = 3$ and $d = 4$, respectively. Worker 1 solved the Sudoku puzzles faster than worker 2 for both approaches. When $d = 3$, the ILP approach solved the Sudoku puzzle faster than the backtracking algorithm when the number of given input values was more than 20 and 23 with worker 1 and worker 2, respectively. When $d = 4$, the ILP approach solved the Sudoku puzzle faster than the backtracking approach if the number of given input values exceeded 98 and 101 with worker 1 and worker 2, respectively.

Figure 7 shows the processing time of worker 1 for both used and unused human strategies, in combination with ILP and the backtracking approaches. Naked single [29] and naked pair [30] algorithms were applied as the human strategy. As the first step, in the used human-strategy case, the human strategy is used to partly solve the given Sudoku problem as much as possible. The result by the human strategy increases the number of given input values. Second, the result of the first step becomes a given input for ILP and backtracking approaches. The results with and without the human strategy for the backtracking approach, which are indicated by dashed lines, are close for all given input values. This is because the processing time of the backtracking approach with and without the human strategy is not affected by the number of given input values. The results with and without the human strategy for the ILP approach, which are indicated by solid lines, are close when the number of given input values is less than 25. This is because the given input does not match any pattern in the human strategy when the number of given input values is low. The processing time of the human strategy with ILP is higher than that of only the ILP approach when the number of given input values is more than 25, where

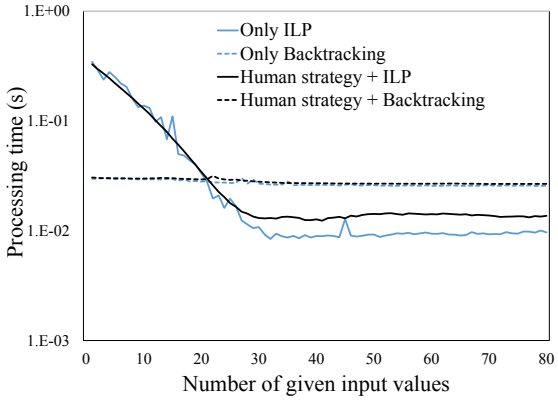


Fig. 7: Comparison of processing time of worker 1 for ILP and Backtracking approaches, and human strategy together with ILP and Backtracking approaches (d = 3).

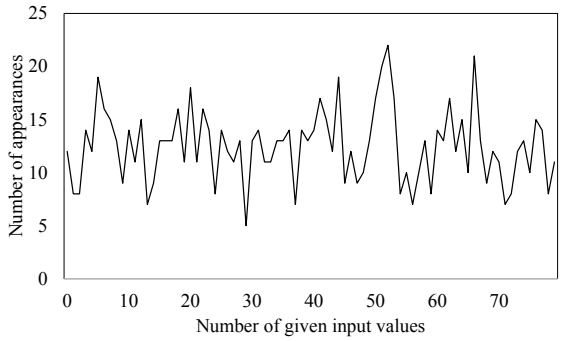


Fig. 8: Number of appearances of each given input value (d = 3).

the human strategy can be usefully applied. The number of given input values does not affect the processing time when it is higher than 25. If the number of input values exceeds 25, the processing time of the ILP approach with the human strategy does not change even if the number of given input values increases. In this region, the ILP approach with and without the human strategy is not affected by the number of given input values. As a result, the processing time of the combination of the human strategy and the ILP approach becomes the summation of the processing time of just the ILP approach and the human strategy.

We randomly generated 1,000 Sudoku puzzles for d = 3 and d = 4. Figures 8 and 9 show the number of appearances of each given input value in the cases of d = 3 and d = 4, respectively. Figures 10 and 11 show the total processing time of the proposed system, compared with conventional equivalents. The total processing time starts when the system receives the first Sudoku puzzle, and stops when the 1,000th Sudoku puzzle is solved. The proposed system achieves the lowest total processing time. The system with round robin distribution and the ILP approach is the slowest of all systems. The proposed system reduces the total processing time by 61% (d = 3) and 98% (d = 4) compared with the system with round robin using the

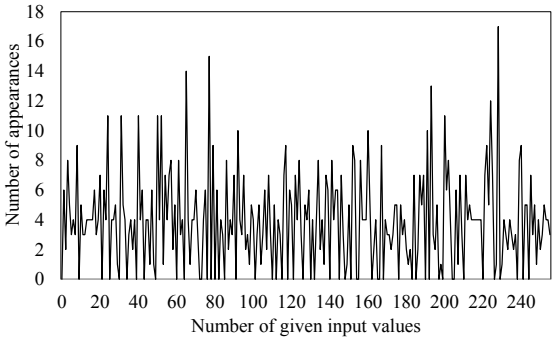


Fig. 9: Number of appearances of each given input value (d = 4).

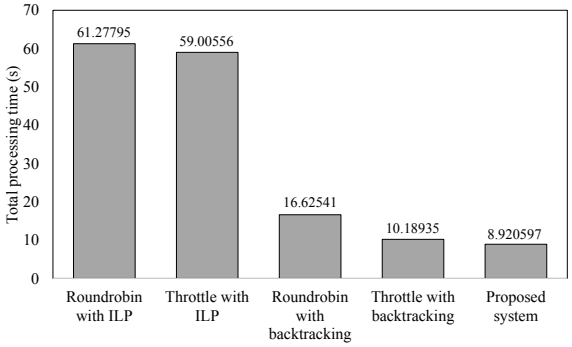


Fig. 10: Comparison of load balancing models when d = 3.

ILP approach, and by 10% (d = 3) and 59% (d = 4) compared with the system with throttle using the backtracking algorithm.

Figure 12 shows the probability density function of the waiting period when d = 3 and the number of given input values is four. The estimated waiting period is 50%, which is 0.02973 seconds in this example. The time will be shown to the user as a waiting period.

Figure 13 demonstrates an example of the SudokuWeb. The user inputs numbers in the Sudoku table as his/her requirement. The SudokuWeb shows the estimated period of waiting period, which is the estimated processing time from the load balancing with learning model to the user. When the Sudoku puzzle is solved, the solution is displayed.

Existing solvers are designed to solve specific Sudoku sizes. In case that the solver is able to solve the Sudoku with an expected size but has no history information of this size in the database, the load balancing with learning model is able to apply to any size of Sudoku. It should be noted that the waiting period cannot be determined if the system's information does not cover the problem submitted. Once the selected worker-solver pair finishes the problem, the information of this problem is added to the database so that the learning model is able to estimate the waiting period for the same size of Sudoku with the same number of given input values.

The processing time increases with the value of d. When d = 3 and d = 4, the number of patterns of the given input values is sufficient to estimate the average

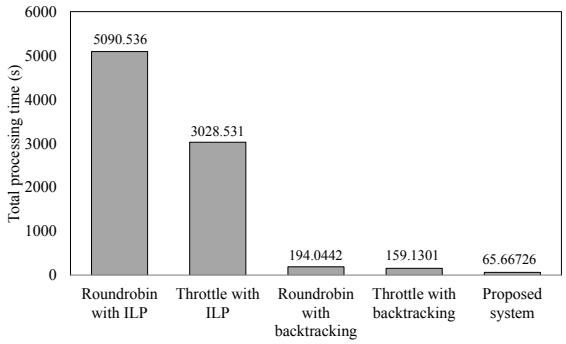


Fig. 11: Comparison of load balancing models when $d = 4$.

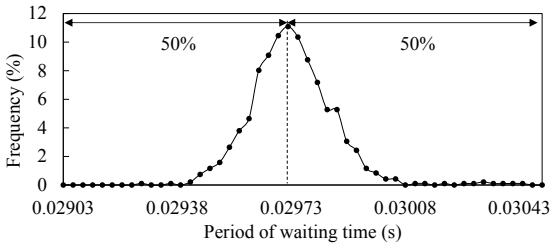
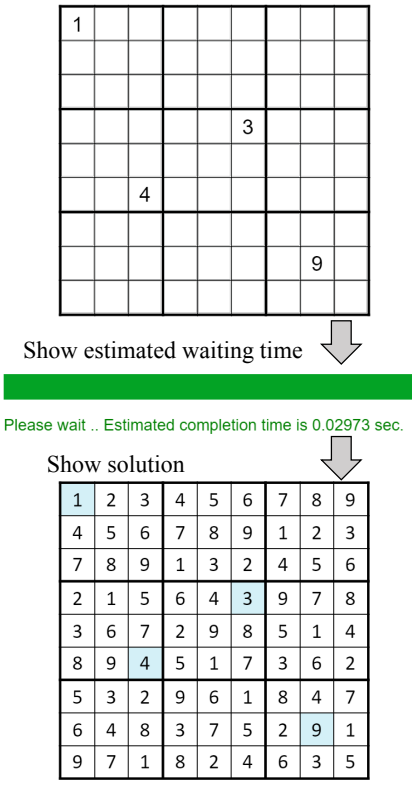


Fig. 12: Probability density function when $d = 3$, number of given input values = 4.



1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	3	2	4	5	6
2	1	5	6	4	3	9	7	8
3	6	7	2	9	8	5	1	4
8	9	4	5	1	7	3	6	2
5	3	2	9	6	1	8	4	7
6	4	8	3	7	5	2	9	1
9	7	1	8	2	4	6	3	5

Fig. 13: SudokuWeb appearance.

processing time. The database holds no information of processing time for each solver for larger d values. Initially, the average processing time of all solvers with the number of given input values is set to zero. The model selects a solver that has the lowest processing time when there is a request. If several solvers have the same lowest processing time, the model randomly selects one of them. Once the problem is solved, the result is added to the database. If there is a problem with the same number of given input values, the model selects the solver that has the lowest processing time.

The accuracy of the proposed model does not depend on Sudoku size but on the amount of information in the database. The model estimates the processing time by learning from the information in the database. If there is little information in the database, the accuracy rate is low.

Depending on hardware limitations, problems with larger Sudoku sizes take longer to complete. The amount of information in the database grows with the increase of Sudoku size. Information for all given input values cannot be prepared when the Sudoku size is large. In this case, the load balancing manager is not able to estimate the finishing time. Therefore, we provide the results with $d < 5$ in this evaluation.

7. Conclusion

We have proposed a load balancing with learning model for a web-based Sudoku solving system with multiple solvers and multiple workers. Our load balancing with learning model is used to distribute requests appropriately to each worker based on estimated finishing time. The model reduces the processing time by selecting the earliest estimated finishing time among all combinations of workers and solvers. It reduces the processing time by up to 61% and 98% for $d = 3$ and $d = 4$, respectively, compared with the conventional load balancing system with the longest processing time. In addition, the Sudoku solving system has the advantage of showing the estimated waiting period before the actual solution is commenced.

References

[1] Sudoku. http://www.sudokuwiki.org/, (accessed 16.11.01).
[2] M. Schottlender, The effect of guess choices on the efficiency of a backtracking algorithm in a Sudoku solver, in Proc. IEEE Long Island Systems, Applications and Technology Conference, New York, USA, May (2014) 1-6.
[3] A.K. Maji and R.K. Pal, Sudoku solver using minigrid based backtracking, in Proc. IEEE Int. Advance Computing Conf., Gurgaon, India, Feb. (2014) 36-44.
[4] M. Asif and R. Baig, Solving NP-complete problem using ACO algorithm, in Proc. Int. Conf. Emerging Technologies, Islamabad, Pakistan, Oct. (2009) 13-16.
[5] I. Sabuncu, Work-in-progress: Solving Sudoku puzzles using hybrid ant colony optimization algorithm, in Proc. Int. Conf. Industrial Networks and Intelligent Systems, Tokyo, Japan, Mar. (2015) 181-184.
[6] S. Kamal, S. S. Chawla, and N. Goel, Detection of Sudoku puzzle using image processing and solving by backtracking, simulated annealing and genetic algorithms: a comparative analysis, in Proc. Third International Conference on Image Information Processing, Waknaghat, India, Dec. (2015) 179-184.
[7] M. A. Al-Betar, M. A. Awadallah, A. L. Bolaji, and B. O. Alijla, β-Hill climbing algorithm for Sudoku game, in

Proc. Palestinian International Conference on Information and Communication Technology, Gaza City, Palestinian Authority, May (2017), 84-88.
[8] N. Musliu and F. Winter, A hybrid approach for the Sudoku problem: using constraint programming in iterated local search, IEEE Intelligent Systems, vol. 32, iss. 2, (2017), 52-62.
[9] Sudoku solving system. http://www.sudoku-solutions.com/, (accessed 17.12.19).
[10] Qqwing Sudoku solving system. https://qqwing.com/solve.html, (accessed 17.12.19).
[11] Hexadoku solving system. http://www.dcode.fr/hexadoku-sudoku-16-solver, (accessed 17.12.19).
[12] 25 × 25 Sudoku solving system. http://www.mario.pd.it/Excel_Sudoku_Variants_Solvers-Risolutori/Sudoku_25x25.htm, (accessed 17.12.19).
[13] Sudoku solving system. http://www.sudokuwiki.org/sudoku.htm, (accessed 17.12.19).
[14] H.M. Widiyanto, P. Pavarangkoon, and E. Oki, SudokuWeb: a web-based solver for mathematical puzzles, in Proc. IEEE Int. Conf. Network Infrastructure and Digital Content, Beijing, China, Sep. (2014) 515-519.
[15] E. Oki, Linear programminng and algorithms for communication networks, CRC Press, 2013.
[16] T. Yato, and T. Seta, Complexity and completeness of finding another solution and its application to puzzles, IEICE Trans. Funda. Elect. Commun. and Comp. Sci. E86-A (5) (2003) 1052-1060.
[17] N. Jussien, A-Z of Sudoku, ISTE Ltd., 2007.
[18] M. Timo, and K. Janne, Solving and rating Sudoku puzzles with genetic algorithms, in Proc. Finnish Artificial Intelligence Conf., Helsinki, Finland, Oct. (2006) 86-92.
[19] H. Simonis, Sudoku as a constraint problem, in Proc. Int. Work. on Modelling and Reformulating Constraint Satisfaction Problem, Barcelona, Spain, Oct. (2005) 17-25.
[20] S. Hwang, and N. Jung, Dynamic scheduling of web server cluster, in Proc. The Ninth International Conference on Parallel and Distributed Systems, Taiwan, Dec. (2002) 563-568.
[21] J. James, B. and Verma, Efficient VM load balancing algorithm for a cloud computing environtment, Int. Journal on Computer Science and Engineering (IJCSE) 4 (9) (2002) 1658-1663.
[22] I. Barazandeh, S. and Mortazavi, Two hierarchical dynamic load balancing algorithms in distributed systems, in Proc. Int. Conf. on Computer and Electrical Engineering, Bangi Selangor, Malaysia, Dec. (2009) 516-521.
[23] V. Cardellini, M. Colajanni, and P.S. Yu, Dynamic load balancing on web-server system, IEEE Internet Computing 3 (3) (1999) 28-39.
[24] S.S. Waraich, Classification of dynamic load balancing strategies in a network of workstations, in Proc. Fifth International Conference on Information Technology: New Generations, Las Vegas, USA, Apr. (2008) 1263-1265.
[25] M.C. Huang, S. Hossein Hosseini, K. and Vairavan, A receiver-initiated load balancing method in computer networks using fuzzy logic control, in Proc. IEEE GLOBECOM, San Francisco, USA, Dec. (2003) 4028-4033.
[26] H.C. Ahn, H.Y. Youn, K.Y. Jeon, and K.S. Lee, Dynamic load balancing for large-scale distributed system with intelligent fuzzy controller, in Proc. IEEE International Conference on Information Reuse and Integration, Las Vegas, USA, Aug. (2007) 576-581.
[27] S.H. Lee, T.W. Kang, M.S. Ko, G.S. Chung, J.M. Gil, and C.S. Hwang, A genetic algorithm method for sender-based dynamic load balancing algorithm in distributed systems, in Proc. Intemational Conference on Knowledge-Based Intelligent Electronic Systems, Adelaide, Australia, May (1997) 302-307.
[28] A. Yousofi, M. Banitaba, and S. Yazdanpanah, A novel method for achieving load balancing in web clusters based on congestion control and cost reduction, in Proc. IEEE Symposium on Computers & Informatics, Kuala Lumpur, Malaysia, Mar. (2011) 347-379.
[29] Naked single. http://hodoku.sourceforge.net/en/tech_singles.php, (accessed 17.12.19).
[30] Naked pair. http://hodoku.sourceforge.net/en/tech_naked.php, (accessed 17.12.19).