

Real-Time Bidirectional Search: Coordinated Problem Solving in Uncertain Situations

Toru Ishida

Abstract—This paper investigates *real-time bidirectional search (RTBS)* algorithms, where two problem solvers, starting from the initial and goal states, physically move toward each other. To evaluate the RTBS performance, two kinds of algorithms are proposed and are compared to real-time unidirectional search. One is called *centralized RTBS* where a supervisor always selects the best action from all possible moves of the two problem solvers. The other is called *decoupled RTBS* where no supervisor exists and the two problem solvers independently select their next moves.

Experiments on mazes and n -puzzles show that 1) in clear situations decoupled RTBS performs better, while in uncertain situations, centralized RTBS becomes more efficient, and that 2) RTBS is more efficient than real-time unidirectional search for 15- and 24-puzzles but not for randomly generated mazes. It will be shown that the selection of the problem solving organization is the selection of the problem space, which determines the baseline of the organizational efficiency; once a difficult problem space is selected, the local coordination among problem solvers hardly overcome the deficit.

Index Terms—Search, real-time search, bidirectional search, problem solving, real-time problem solving, organizational problem solving, heuristic depression.

1 INTRODUCTION

SUPPOSE there are two robots trying to meet in a fairly complex maze: one is starting from the entrance and the other from the exit. Each robot always knows its current location in the maze, and can communicate with the other robot; thus, each robot always knows its goal location. Even though the robots do not have a map of the maze, they can gather information around them through various sensors.

For further sensing, however, the robots are required to physically move (as opposed to state expansion): planning and execution must be interleaved. In such a situation, how should the robots behave to efficiently meet with each other? Should they negotiate their actions, or make decisions independently? Is the two robot organization really superior to a single robot one? These are some of the research issues of *real-time bidirectional search*, which will be investigated throughout this paper.

In traditional *off-line unidirectional search*, the problem solver located at the initial state expands its wave front toward the goal state. The computational complexity of this search can be represented by B^D , where B indicates the number of operators (branching factors) applicable at each state, and D indicates the depth of the search. On the other hand, *off-line bidirectional search* aims at improving the problem solving efficiency by starting from both the initial and goal states. If the two wave fronts can meet at the depth of $D/2$, the total amount of computation becomes $2 \times B^{D/2}$, which is significantly less than B^D . However, if heuristic search, such

as A^* , is employed, the performance cannot easily be improved because of the difficulty faced by the two wave fronts in trying to meet in the middle of the problem space. This phenomenon was observed by Pohl during his research on the *BHPA (Bidirectional Heuristic Path Algorithm)* [19].

De Champeaux thus proposed an algorithm called the *BHFFA (Bidirectional, Heuristic, Front-to-Front Algorithm)* [2], [3], where two problem solvers coordinate their search directions by communicating their wave fronts in order to shorten the distance between the two fronts. Though this algorithm can reduce the number of expanded states, the coordination cost cancels the advantage. This is because the wave fronts expand on the order of B^D . The history of bidirectional search makes us aware of the difficulties existing in multi-agent problem solving.

All previous research on bidirectional search focused on off-line search. The challenge of this paper is to study *real-time bidirectional search (RTBS)*, and to investigate its performance. *Real-time search* always computes a plausible next move and physically executes that move in constant time, while *off-line search* computes the entire solution path before executing the first step in the path. Real-time search is effective in uncertain situations, even though the lookahead ability of the problem solver is limited (note: the sensing ability of mobile robots is always physically limited); due to the limitation, planning and execution must be interleaved [7], [21]. Real-time search is also promising in dynamic situations, where problems change even as they are being solved. Since real-time search takes a constant time for each move, if the speed of the problem solver can be made faster than the speed of any problem change, we can expect any problem to be solved eventually.

Thus, RTBS can be viewed as coordinated problem solving in uncertain and dynamic situations, and as a for-

• The author is with the Department of Information Science, Kyoto University, Kyoto, 606-01, Japan. E-mail: ishida@kuis.kyoto-u.ac.jp.

Manuscript received Mar. 20, 1995; revised Feb. 9, 1996.

Recommended for acceptance by R. Kasturi.

For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number P96023.

mal step towards organizational problem solving [4], [5], [6], [10], viewing DAI (Distributed Artificial Intelligence) problems as distributed search [17]. In RTBS, two problem solvers starting from the initial and goal states physically move toward each other. As a result, unlike the off-line bidirectional search (such as BHFFA), the coordination cost is expected to be limited within some constant time. Since the planning time is also limited, however, the moves of the two problem solvers may be inefficient.

This paper proposes two kinds of RTBS algorithms and compares them to *real-time unidirectional search (RTUS)*. One is called *centralized RTBS* where the best action is selected from among all possible moves of the two problem solvers, and the other is called *decoupled RTBS* where the two problem solvers independently make their own decisions. These algorithms are developed based on available real-time search algorithms, i.e., *RTA** (*Real-Time A**), *LRTA** (*Learning Real-Time A**) [15] and *MTS* (*Moving Target Search*) [8], [9], [13]. An experimental evaluation has been made by using mazes and n -puzzles, which are the most common examples in the area of search.

The evaluation results will show that, in clear situations, (i.e., heuristic functions return accurate values), decoupled RTBS performs better than centralized RTBS, while in uncertain situations (i.e., heuristic functions return inaccurate values), the latter becomes more efficient. Surprisingly enough, compared to RTUS, RTBS dramatically reduces the number of moves for 15- and 24-puzzles,¹ but increases it for randomly generated mazes.

This paper is not intended to stress the superiority of RTBS, even though RTBS efficiently solves n -puzzles. The motivation is rather to analyze RTBS performance to understand the theory behind the performance of coordinated problem solving. The performance of real-time search heavily depends on the *heuristic topography*, the topography composed by the problem space and heuristic values. More specifically, the problem solving performance is affected by the shape of a *heuristic depression*, i.e., a set of states whose heuristic values are less than or equal to those of directly surrounding states. It will be shown that the effectiveness of RTBS is determined by the topographical changes between the real-time unidirectional and bidirectional search problem spaces.

2 REAL-TIME SEARCH ALGORITHMS

This section briefly reviews available real-time search algorithms, e.g., *RTA** and *LRTA** for fixed goal problems, and *MTS* for moving goal problems. The problem space is represented as a connected graph. The graph is undirected, allowing the problem solver to move along an edge in both directions. To simplify the following discussion, we assume a unit cost for all edges in the graph. The problem solver does not have a map of the problem space, but can utilize a *heuristic function* that returns an estimate of the distance to the goal. The heuristic function must be *admissible*, meaning that it must never overestimate the actual distance [18].

1. RTBS can solve 35-, 48-, and 63-puzzles, which cannot be solved by conventional real-time search algorithms within a reasonable amount of time.

2.1 Fixed Goal Problems

*Real-Time-A** (*RTA**), *learning-Real-Time-A** (*LRTA**) [15], and their extensions [22] are real-time search algorithms for fixed goal problems. They commit to individual moves in constant time, and interleave the computation of each move with its execution. Initially, estimated distances are obtained by evaluating the heuristic function. Through repeated exploration of the space, however, more accurate values are learned until they eventually equal the exact distances to the goal. In the following description, let x be the current position of the problem solver, and $h(x)$ be the current heuristic estimate of the distance from x to the goal.

The *LRTA** algorithm repeats the following steps until the problem solver reaches the goal state.

[*LRTA**]

- 1) Calculate $h(x')$ for each neighbor x' of x .
- 2) Update the value of $h(x)$ as follows:

$$h(x) \leftarrow \min_{x'} \{h(x') + 1\}$$
- 3) Move to the neighbor x' with the minimum $h(x')$, i.e., assign the value of x' to x . (Ties are broken randomly.)

From the current position x of the problem solver, *Step2* of *LRTA** calculates $h(x') + 1$ for each neighbor x' of the current state, where "1" is the cost of the edge from x to x' . This is allowed, because $h(x') + 1$ represents lower bounds on the actual distance to the goal through each of the neighbors, and the actual distance from x must be at least as large as the smallest of these estimates. Since $h(x')$ is not overestimated, neither can $h(x)$ be overestimated.

*RTA**, on the other hand, updates the value of $h(x)$ in a different way from *LRTA**. In *Step2* of *RTA**, instead of setting $h(x)$ to the smallest value of $h(x') + 1$ for all neighbors x' , the second smallest value is assigned to $h(x)$. Thus, *RTA** learns more efficiently than *LRTA**, but can overestimate heuristic distances. The *RTA** algorithm is shown below. Note that *secondmin* represents the function that returns the second smallest value.

[*RTA**]

- 1) Calculate $h(x')$ for each neighbor x' of x .
- 2) Update the value of $h(x)$ as follows:

$$h(x) \leftarrow \text{secondmin}_{x'} \{h(x') + 1\}$$
- 3) Move to the neighbor x' with the minimum $h(x')$, i.e., assign the value of x' to x . (Ties are broken randomly.)

In a finite problem space with positive edge costs, in which there exists a path from every node to the goal, and starting with nonnegative admissible initial heuristic values, both *RTA** and *LRTA** are *complete* in the sense that they will eventually reach the goal.

Since the second smallest values are always maintained, *RTA** can make locally optimal decisions:² each move made by *RTA** is along a path whose estimated cost toward the goal is a minimum based on the already-obtained information. *LRTA** behaves less efficiently than *RTA**. Since *LRTA** never overestimates, however, it *learns* the optimal

2. This result is for the case of trees and cannot extend to general graphs with cycles [15].

solutions through repeated trials: the heuristic values will eventually converge to their exact values along every optimal path.

2.2 Moving Goal Problems

Moving Target Search (MTS) [1], [8], [9], [12], [13] handles situations, in which the location of the goal possibly changes during the course of the search. Off-line search cannot handle moving goal problems, because the goal can change each time before the path toward the goal is found. Thus, only real-time search, which interleaves planning and execution, is feasible.

The moving goal is called a *target*. It is assumed that the problem solver and the target move alternately, and can traverse at most one edge in each turn. The speed of the target is reduced by assuming that periodically the target will make no move, and remain at its current state. The problem solver has no control over the movements of the target, but always knows the target's current position.

At the initialization of the algorithm, the current state of the problem solver is assigned to x , and the current state of the target to y . Let $h(x, y)$ be the current heuristic estimate of the distance between x and y . The MTS algorithm repeatedly performs the following steps until the problem solver and the target occupy the same state.

[MTS]

When the problem solver moves:

- 1) Calculate $h(x', y)$ for each neighbor x' of x .
- 2) Update the value of $h(x, y)$ as follows:

$$h(x, y) \leftarrow \max \left\{ \begin{array}{l} h(x, y) \\ \min_{x'} \{h(x', y) + 1\} \end{array} \right\}$$

- 3) Move to the neighbor x' with the minimum $h(x', y)$, i.e., assign the value of x' to x . (Ties are broken randomly.)

When the target moves:

- 1) Calculate $h(x, y')$ for the target's new position y' .
- 2) Update the value of $h(x, y)$ as follows:

$$h(x, y) \leftarrow \max \left\{ \begin{array}{l} h(x, y) \\ h(x, y') - 1 \end{array} \right\}$$

- 3) Reflect the target's move to the problem solver's goal, i.e., assign the value of y' to y .

Careful readers might be aware that MTS is based on LRTA*, but not on RTA*. This is because, in the case of moving goals, the search process toward the target's particular position is often interrupted by the target's moves, and restarted when the target revisits the same position. MTS extends LRTA*, because LRTA* never overestimates, and can accumulate multiple search trials.

In a finite problem space, in which a path exists between every pair of nodes, starting with non-negative admissible initial heuristic values, the MTS algorithm is *complete* in the sense that the problem solver executing MTS is guaranteed to eventually reach the target, if the target periodically skips moves.

2.3 Real-Time Search Behavior

The performance of real-time search heavily depends on the topography of the estimated problem space. This is because, in real-time search, the problem solver always commits to its decision, which will seriously affect the consequent problem solving process.

Fig. 1 represents the real-time search behavior in a one-dimensional problem space. The x -axis represents the positions of the problem solver and the goal (i.e., target), while the y -axis represents the estimated distance between the problem solver and the goal. In this case, the goal is assumed to be fixed at position q . An example set of initial heuristic values is plotted with a wide line, and the arrows indicate the moves of the problem solver. Incremental updates of estimated distances are indicated by the dark boxes.

To explain the problem solver's behavior, we define a *heuristic depression* with respect to a single goal state as follows.

[Heuristic Depression]

A *heuristic depression* is a set of connected states with heuristic values less than or equal to those of the set of immediate and completely surrounding states. A heuristic depression is *locally maximal*, when no single surrounding state can be added to the set; in this case, the set satisfies the condition of a heuristic depression.

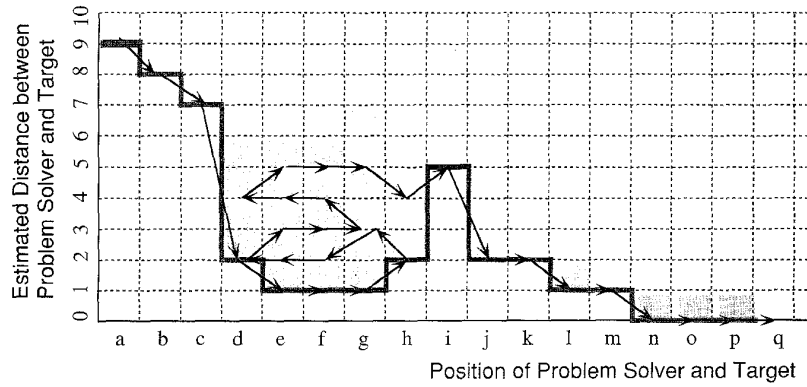
For example, in Fig. 1, the positions from d to h form a locally maximal heuristic depression. This is because their heuristic values are less than those of the surrounding states c and i ; if c or i is added to the set, the set does not satisfy the condition of heuristic depression. Note that no depression exists in the actual distance. However, as the situation becomes uncertain, heuristic values differ significantly from the actual distances, and so heuristic depressions tend to appear frequently in the problem space.

As described in Fig. 1, the problem solver performing real-time search repeatedly moves along the slope of heuristic values. If an erroneous decision is made at the boundary of a heuristic depression, the problem solver cannot stop moving toward the bottom of the depression. When placed in a heuristic depression, the problem solver often finds no way of decreasing the heuristic difference, and recognizes that its heuristic values are inaccurate. In such cases, the problem solver cannot reach the target without "filling" the depression by repeatedly updating the heuristic values. To summarize, *the performance bottleneck of real-time search results from its inefficiency in filling heuristic depressions*. This fact will be the key to understanding the RTBS performance described in Section 4.

3 RTBS ALGORITHMS

3.1 Framework

Pohl proposed the framework of off-line bidirectional search: the *control strategy* first selects *forward* or *backward* search, and then performs the actual state expansion [19]. We here propose the framework of RTBS algorithms, which inherits the framework of off-line bidirectional search. The difference from Pohl's framework is that, in RTBS, the forward and backward operations are not state expansions but physical moves of the problem solvers.



The problem solver is initially positioned at "a," and the target at "q."
 → Moves of the problem solver
 — Initial heuristic value
 - - - Learned distance

Fig. 1. Real-time search behavior.

In RTBS, the following steps are repeatedly executed until the two problem solvers meet in the problem space.

- 1) *Control strategy:*
 Select a forward (*Step2*) or backward move (*Step3*).
- 2) *Forward move:*
 The problem solver starting from the initial state (i.e., the *forward problem solver*) moves toward the problem solver starting from the goal state.
- 3) *Backward move:*
 The problem solver starting from the goal state (i.e., the *backward problem solver*) moves toward the problem solver starting from the initial state.

3.2 Classification of RTBS Algorithms

In RTBS, control strategies are crucial for characterizing the autonomy of the problem solvers.³ RTBS algorithms can be classified into the following two categories depending on the autonomy of the problem solvers.

Centralized control:

Control strategy evaluates all possible forward and backward moves, and selects the best one. As a result, the moves of the two problem solvers are completely controlled by the centralized decisions. When adopting centralized control, forward and backward moves are not always performed alternately.

Decoupled control:

Control strategy employs the minimal control necessary to guarantee the termination of the algorithm. As a result, the two problem solvers make their decisions independently. In the algorithms described in 3.4, the control strategy simply selects forward and backward moves alternately.

3. Pohl discussed control strategies for off-line bidirectional search [19]. Several strategies are proposed including *cardinality comparison*, which equalizes the number of expanded states of the two problem solvers.

Besides the type of control, the RTBS algorithms can be further classified from the information sharing point of view, i.e., how two problem solvers share heuristic distances.

Shared heuristic information:

The forward and backward problem solvers share heuristic values, $h(x, y)$, and maintain the values together.

Distributed heuristic information:

Each of the forward and backward problem solvers maintains its own heuristic values, say $h_f(x, y)$ and $h_b(x, y)$. The initial values of $h_f(x, y)$ and $h_b(x, y)$ might be different, because each problem solver can employ its own heuristic function.

Table 1 summarizes the classification of the RTBS algorithms. In the following two subsections, only two extremes in four possible combinations (*centralized control with shared heuristic information* and *decoupled control with distributed heuristic information*) will be investigated. The algorithms called RTA*/B, LRTA*/B and MTS/B will be proposed based on available real-time search algorithms. We simply call the two extremes *centralized RTBS* and *decoupled RTBS*. Two combinations that remain will not be discussed, because they can be generated rather straightforwardly, and their individual performance lies somewhere between the two extremes.

TABLE 1
 CLASSIFICATION OF RTBS ALGORITHMS

Control \ Information	Shared	Distributed
Centralized	RTA*/B, LRTA*/B	—
Decoupled	—	MTS/B

Let us take an n -puzzle example. The RTUS algorithm utilizes a single game board, and interleaves both planning

and execution; it evaluates all possible actions at a current puzzle state and physically performs the best action (slides one of the movable tiles). By repeating these steps, the algorithm eventually achieves the goal state. On the other hand, the RTBS algorithm utilizes two game boards. At the beginning, one board indicates the initial state and the other indicates the goal state. What is pursued in this case is to equalize the two puzzle states. Centralized RTBS behaves as if one person operates both game boards, while decoupled RTBS behaves as if each of two people operates his/her own game board independently.⁴

3.3 Centralized RTBS

In centralized RTBS, the control strategy selects the best action from among all the possible forward and backward moves to minimize the estimated distance to the goal state.

Below, we describe two centralized RTBS algorithms called LRTA*/B and RTA*/B, which are based on LRTA* and RTA*, respectively. The positions of two problem solvers are represented by x and y , while the estimated distance between two problem solvers is represented by $h(x, y)$.

[LRTA*/B]

1) Control strategy:

- a) Calculate $h(x', y)$ for each neighbor x' of x .
Calculate $h(x, y')$ for each neighbor y' of y .

- b) Update the value of $h(x, y)$ as follows:

$$h(x, y) \leftarrow \min_{x', y'} \begin{cases} h(x', y) + 1 \\ h(x, y') + 1 \end{cases}$$

- c) If $\min_x h(x', y) < \min_y h(x, y')$ select a forward move.
If $\min_x h(x', y) > \min_y h(x, y')$ select a backward move.
Otherwise, select a forward or backward move randomly.

2) Forward move:

Move the forward problem solver to the neighbor x' with the minimum $h(x', y)$, i.e., assign the value of x' to x . (Ties are broken randomly.)

3) Backward move:

Move the backward problem solver to the neighbor y' with the minimum $h(x, y')$, i.e., assign the value of y' to y . (Ties are broken randomly.)

The RTA*/B algorithm is presented below. The difference between LRTA*/B and RTA*/B is that the latter utilizes the second minimum value for updating heuristic distances.

[RTA*/B]

1) Control strategy:

- a) Calculate $h(x', y)$ for each neighbor x' of x .
Calculate $h(x, y')$ for each neighbor y' of y .

- b) Update the value of $h(x, y)$ as follows:

$$h(x, y) \leftarrow \text{secondmin}_{x', y'} \begin{cases} h(x', y) + 1 \\ h(x, y') + 1 \end{cases}$$

- c) If $\min_x h(x', y) < \min_y h(x, y')$ select a forward move.
If $\min_x h(x', y) > \min_y h(x, y')$ select a backward move. Otherwise, select a forward or backward move randomly.

2) Forward move:

Move the forward problem solver to the neighbor x' with the minimum $h(x', y)$, i.e., assign the value of x' to x . (Ties are broken randomly.)

3) Backward move:

Move the backward problem solver to the neighbor y' with the minimum $h(x, y')$, i.e., assign the value of y' to y . (Ties are broken randomly.)

Both LRTA*/B and RTA*/B are complete. The proof is obtained by showing that the centralized RTBS algorithms are the results of applying the corresponding RTUS algorithms to the RTBS problem space.

THEOREM 1. *In a finite problem space with positive edge costs, in which there exists a path from every node to the goal, and starting with non-negative admissible initial heuristic values, both RTA*/B and LRTA*/B are complete in the sense that the forward and backward problem solvers will eventually share the same state.*

PROOF. We first define the RTBS problem space as follows.

- 1) Let a RTBS state be a pair of locations of two problem solvers: Let x and y be RTUS states; then the pair of RTUS states (x, y) becomes an RTBS state. Let i and g be the RTUS initial and goal states; then, the pair of RTUS states (i, g) becomes the RTBS initial state. The goal state of bidirectional search requires two problem solvers to share the same RTUS states; i.e., consequently, all (x, y) , where $x = y$, become the RTBS goal states.
- 2) The distance to the goal state can be represented by the distance between a pair of current locations of two problem solvers, x and y ; the estimated distance to the goal state is represented by $h(x, y)$.
- 3) There is a state transition in the RTBS problem space when one of the problem solvers moves. Thus, the branching factor in RTBS is the sum of the branching factors of two problem solvers.

Based on the above mapping between the real-time unidirectional and bidirectional search problem spaces, LRTA*/B and RTA*/B can be seen as the result of applying LRTA* and RTA* to the RTBS problem space. For example, *Step1(a)* and *Step1(b)* of LRTA*/B correspond to *Step1* and *Step2* of LRTA*. *Step1(c)*, *Step2* and *Step3* of LRTA*/B correspond to *Step3* of LRTA*. As described in Section 2, since LRTA* and RTA* are complete, LRTA*/B and RTA*/B are complete in the RTBS problem space.

3.4 Decoupled RTBS

In decoupled RTBS, the control strategy merely selects the forward or backward problem solver alternately. As a result, each problem solver independently makes decisions based on its own heuristic information.

4. In a human problem solving process, n -puzzles are more efficiently solved by setting appropriate subgoals (i.e., dressing tiles column by column). This paper, however, will not discuss subgoaling techniques.

At this time, MTS is the only algorithm that can handle situations in which both the problem solver and the goal move. Therefore, decoupled RTBS should employ MTS for both forward and backward moves. In the following description, x and y indicate the positions of the forward and backward problem solvers, while $h_f(x, y)$ and $h_b(x, y)$ indicate the heuristic distances estimated by the forward and backward problem solvers, respectively. Note that the initial values of $h_f(x, y)$ and $h_b(x, y)$ might be different, because the two problem solvers can utilize different heuristic functions.

[MTS/B]

1) Control strategy:

Select a forward or backward move alternately.

2) Forward move:

For the forward problem solver:

- Calculate $h_f(x', y)$ for each neighbor x' of x .
- Update the value of $h_f(x, y)$ as follows:

$$h_f(x, y) \leftarrow \max \left\{ \begin{array}{l} h_f(x, y) \\ \min_{x'} \{h_f(x', y) + 1\} \end{array} \right\}$$

- Move to the neighbor x' with the minimum $h_f(x', y)$, i.e., assign the value of x' to x . (Ties are broken randomly.)

For the backward problem solver:

- Calculate $h_b(x', y)$ for the forward problem solver's new position x' .
- Update the value of $h_b(x, y)$ as follows:

$$h_b(x, y) \leftarrow \max \left\{ \begin{array}{l} h_b(x, y) \\ h_b(x', y) - 1 \end{array} \right\}$$

- Reflect the forward problem solver's move to the backward problem solver's goal, i.e., assign the value of x' to x .

3) Backward move:

For the backward problem solver:

- Calculate $h_b(x, y')$ for each neighbor y' of y .
- Update the value of $h_b(x, y)$ as follows:

$$h_b(x, y) \leftarrow \max \left\{ \begin{array}{l} h_b(x, y) \\ \min_{y'} \{h_b(x, y') + 1\} \end{array} \right\}$$

- Move to the neighbor y' with the minimum $h_b(x, y')$, i.e., assign the value of y' to y . (Ties are broken randomly.)

For the forward problem solver:

- Calculate $h_f(x, y')$ for the backward problem solver's new position y' .
- Update the value of $h_f(x, y)$ as follows:

$$h_f(x, y) \leftarrow \max \left\{ \begin{array}{l} h_f(x, y) \\ h_f(x, y') - 1 \end{array} \right\}$$

- Reflect the backward problem solver's move to the forward problem solver's goal, i.e., assign the value of y' to y .

As described in 2.2, the original MTS algorithm assumes that the problem solver moves faster than the target. This

assumption was introduced to prevent the target from escaping the problem solver forever. Note that the same assumption is even required in MTS/B, however, where the two problem solvers try to meet each other.

Fig. 2 illustrates the case where two problem solvers fall into an infinite loop, moving alternately at the same speed. Fig. 2a represents the situation often occurring in mazes when utilizing the Manhattan distance as a heuristic function. Two problem solvers move to the opposite sides of the same wall. The problem solvers are currently located at x_1 and y_1 . The surrounding states of the problem solvers are indicated by x_0 to x_3 and y_0 to y_3 , respectively. Suppose the problem solver located at x' attempts to move. Since the heuristic values at two neighboring states are $h(x_0, y_1) = 2$ and $h(x_2, y_1) = 2$, the problem solver updates $h(x_1, y_1) = 3$ and moves toward x_0 or x_2 . The tie is broken randomly. Starting from Fig. 2a, the two problem solvers repeatedly move around on four states, and update the corresponding heuristic values. The heuristic values eventually become like Fig. 2b. The updated distances are indicated by bold characters. The two problem solvers then alternately move forever.

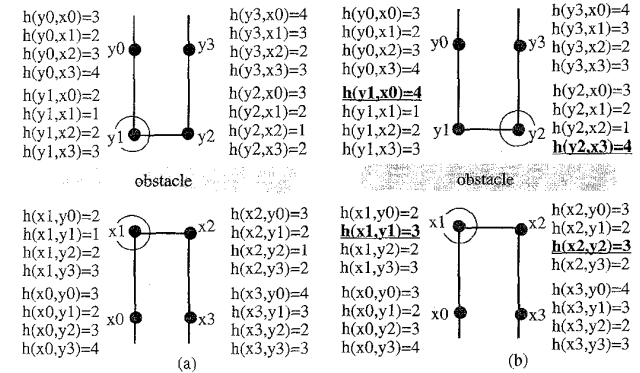


Fig. 2. Example of an infinite loop.

The completeness of MTS/B can be easily derived like that for MTS.

THEOREM 2. *In a finite problem space, in which a path exists between every pair of nodes, starting with non-negative admissible initial heuristic values, the forward and backward problem solvers executing MTS/B will eventually share the same state, if the two problem solvers move alternately, and the forward (or backward) problem solver periodically skips some of its moves.*

PROOF. It has been proven that MTS is complete assuming that the problem solver and the target move alternately, and the target periodically skips some of its moves. In MTS/B, two problem solvers independently perform MTS. Thus, it is sufficient to show that one of the two problem solvers satisfies the condition of the completeness of MTS. One of the two problem solvers, each of which performs forward or backward moves, is complete, because one of them skips its moves periodically. \square

4 PERFORMANCE ANALYSIS

This section examines the performance of the RTBS algorithms. The computational complexity will be investigated first, and then the actual performance will be measured on typical example problems.

4.1 Computational Complexity

Fig. 3a illustrates the search tree for real-time unidirectional search. Each node represents a position of the problem solver. An example path from the initial state to the goal state is represented by the wide line. Let B_f be the number of operators (branching factor) for the forward move, B_b be the number of operators for the backward move, and D be the number of moves before reaching the goal state. Then, the number of generated states can be represented by $B_f \times D$.

The key to understanding the real-time bidirectional search performance is to assume that RTBS algorithms solve a totally different problem from RTUS; i.e., the difference between the real-time unidirectional and bidirectional search is not the number of problem solvers, but their problem spaces.

Let x and y be locations of two problem solvers. Then the pair of locations (x, y) becomes an RTBS state. Thus, when the number of RTUS states is n , the number of RTBS states becomes n^2 . Let i and g be the RTUS initial and goal states; then (i, g) becomes the RTBS initial state. The goal state of RTBS requires both problem solvers to share the same RTUS state. Thus, the RTBS goal state is not unique, i.e., when there are n RTUS states, there are n RTBS goal states. Each state transition in the RTBS problem space corresponds to a move by one of the problem solvers. Thus, the branching factor in RTBS is the sum of the branching factors of the two problem solvers.

Fig. 3b represents the search tree for centralized RTBS. At each move, the best action is selected from all possible $B_f + B_b$ moves of the two problem solvers. Let D be the sum of moves of the two problem solvers before meeting with each other. Then, the number of generated states can be represented by $(B_f + B_b) \times D$.

In decoupled RTBS, two problem solvers independently make their own decisions and alternately move toward the other problem solver. Let us assume, however, that even in decoupled RTBS, the two problem solvers move in an RTBS problem space. Fig. 3c represents the search tree for decoupled RTBS. Each problem solver selects the best action from possible B_f or B_b moves represented by solid edges, but does not examine the moves of the other problem solver represented by dashed edges. The figure shows that the selected action might not be the best among possible $B_f + B_b$ moves of the two problem solvers. Let D be the sum of moves of the two problem solvers. Since decoupled RTBS utilizes partial information, D increases (as opposed to centralized RTBS). On the other hand, the number of generated states can be represented by $\{(B_f + B_b)/2\} \times D$, which can decrease (as opposed to centralized RTBS).

The relations among the number of moves and the number of generated states are summarized as follows. Note that *the number of generated states determines the planning time, and the number of moves determines the execution time.*

- In centralized RTBS, the number of moves is smaller than that in decoupled RTBS. This is because, when assuming that both centralized and decoupled RTBS share the same problem space, centralized RTBS utilizes more information than decoupled RTBS.
- In centralized RTBS, the number of generated states per move is double that in RTUS or decoupled RTBS. This is because the branching factor of centralized RTBS is the sum of the branching factors of the two problem solvers.

In both centralized and decoupled RTBS, the cost of visiting one state is larger than that in RTUS as follows.

- In centralized RTBS, the CPU time per the number of generated states is about two fold that in RTUS. This is because an RTBS state is composed by a pair of RTUS states.
- In decoupled RTBS, the CPU time per the number of generated states is about two fold that in RTUS. This is because, for each move in decoupled RTBS, heuristic distances are updated by two problem solvers.

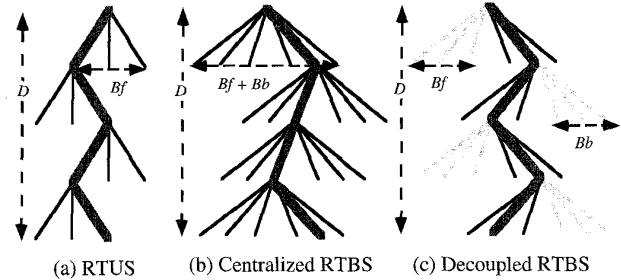


Fig. 3. Comparison of computational complexities.

4.2 Measurement on Typical Problems

The real-time unidirectional and bidirectional search algorithms (RTA*, LRTA*, RTA*/B, LRTA*/B and MTS/B) are applied to the following two typical path finding problems.

1) Randomly Generated Mazes

Mazes have been utilized in the area of search including bidirectional search [16]. In this measurement, we represent a maze as a grid space, replacing the arbitrary number of junctions by obstacles. With a high obstacle ratio (more than 20%), the obstacles combine and form walls of various shapes. The complexity of the maze rapidly increases as the ratio increases from 25% to 35%. At the start, two problem solvers are placed so that the Manhattan distance between them is 50 (i.e., the direct path between the initial and goal states consists of 50 edges). Both problem solvers can move down/up/right/left in the grid space. The Manhattan distance is used as a heuristic function.

2) n -Puzzles

n -puzzle problems, especially 15-puzzle ones have repeatedly been appearing in bidirectional search literatures [2], [19], [20]. In this evaluation, as 15-puzzles, the first ten problems of the 100 problems presented in [14] are utilized. Ten problems of 8- and

24-puzzles are randomly generated. The distances between two n -puzzle states are estimated by summing up the Manhattan distances between the present and goal locations of all tiles.

Figs. 4, 5, 6, and 7 display the experimental results. The x -axis represents the obstacle ratio in mazes and the problem ID in n -puzzles. The y -axis represents the total number of moves, generated states, and the CPU time⁵ taken by two problem solvers. Programs are written in Allegro Common Lisp running on a SUN SS-2. All numbers in Figs. 4, 5, and 6 are obtained by averaging 100 trials, while the numbers in Fig. 7 are obtained by averaging 30 trials due to the limitation of computing resources. The major results obtained from this experiment are described as follows.

4.2.1 Centralized vs. Decoupled Control

The superiority or inferiority of centralized and decoupled RTBS depends on whether the situation is clear or uncertain, i.e., whether the heuristic function returns accurate values or not.

- In clear situations, decoupled RTBS performs better.

From the maze experiences, when the obstacle ratio is low, there is not much difference in the number of moves between centralized/decoupled RTBS. This means that, in clear situations, two problem solvers can independently make decisions without losing efficiency. However, since centralized RTBS expands states double that of decoupled RTBS, the planning time for decoupled RTBS is almost 1/2 that of centralized RTBS.⁶

- In uncertain situations, centralized RTBS performs better.

It is clearly observed from the maze experiences that, as the situation becomes uncertain (the obstacle ratio exceeds 20%), centralized RTBS becomes much more efficient than decoupled RTBS. Our n -puzzle experiences⁷ also show that the number of moves for centralized RTBS is about 1/2 that of decoupled RTBS.

These results are supported by the discussion of computational complexity in Section 4.1.

4.2.2 Unidirectional vs. Bidirectional Search

By comparing real-time unidirectional and bidirectional search performances, interesting observations can be made as follows.

- In clear situations, both search algorithms have a similar performance.

Our maze experiences show that, in clear situations, where the obstacle ratio is less than 10%, the number of moves in RTUS and RTBS are not much different. This is because, in clear situations, near

5. Garbage collection time is included.

6. A further speedup of decoupled RTBS can be expected by utilizing two parallel processors.

7. In mazes, when the obstacle ratio becomes 35%, while the optimal path length is 60 to 80, the number of moves becomes 400 to 4,000, i.e., 7 to 50 times longer than the optimal path length. Similarly, in 15-puzzles, while the optimal path length is 40 to 60, 2,000 to 5,000 moves are required, i.e., 50 to 80 times longer than the optimal path length. These numbers show that the situation in 15-puzzles is as uncertain as that in fairly complex mazes.

optimal solutions can be easily obtained by applying various search techniques.

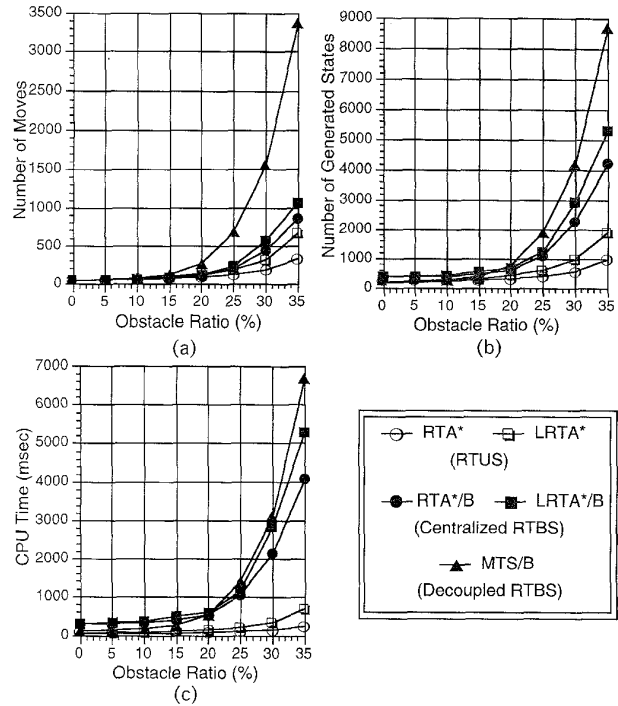


Fig. 4. Performance for mazes.

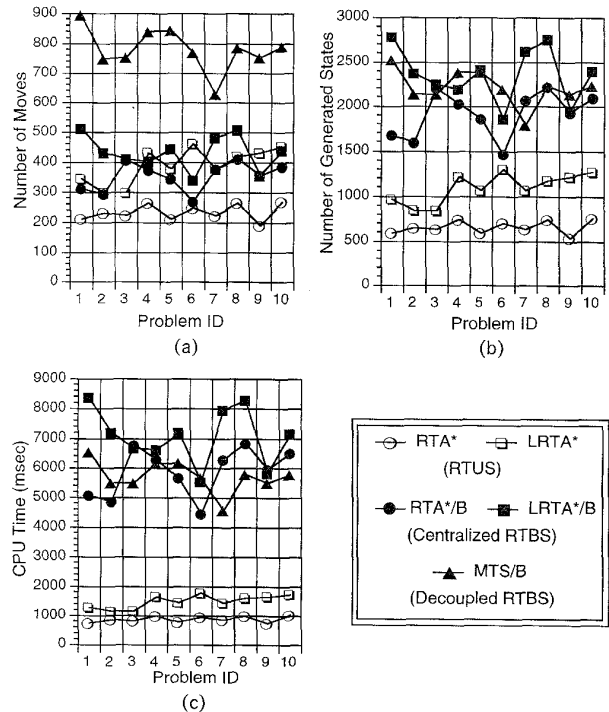


Fig. 5. Performance for 8-puzzles.

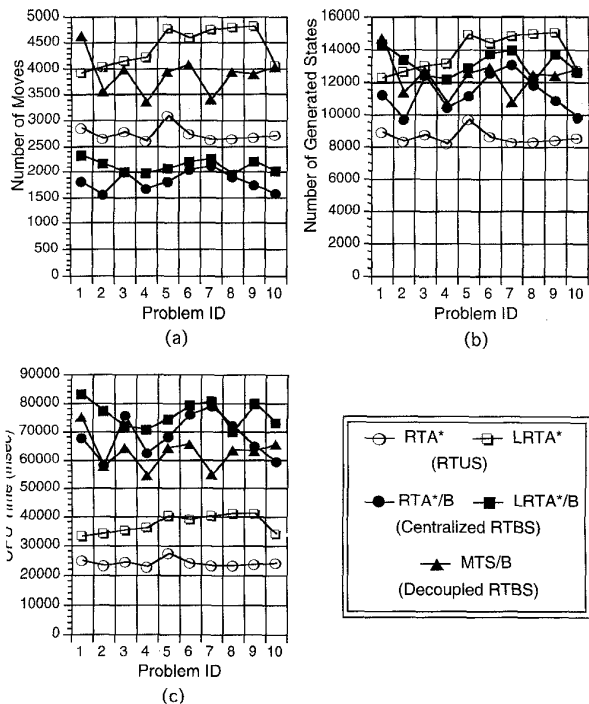


Fig. 6. Performance for 15-puzzles.

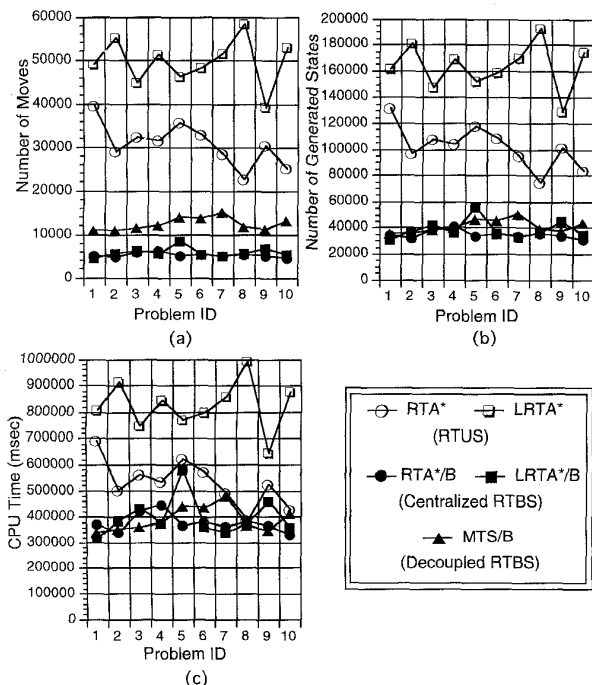


Fig. 7. Performance for 24-puzzles.

- In uncertain situations, the superiority or inferiority of real-time unidirectional and bidirectional search heavily depends on the applied problems.

RTBS can solve 15- and 24-puzzles with less moves than RTUS: the number of moves for centralized

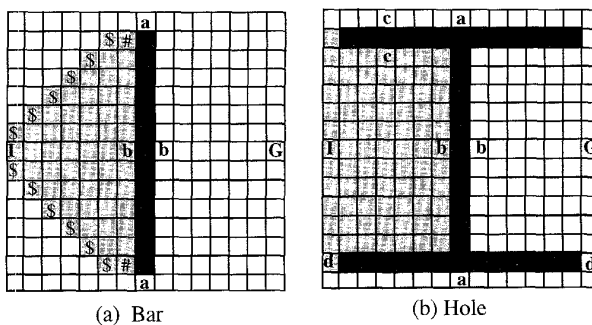
RTBS is around 1/2 in 15-puzzles and 1/6 in 24-puzzles that for RTUS. In mazes, however, when the obstacle ratio becomes more than 10%, RTBS cannot perform better than RTUS. As the obstacle ratio increases, this tendency becomes even more clear: the number of moves for RTBS is roughly doubled compared to RTUS.⁸

Why is RTBS effective for 15- and 24-puzzles, but not for randomly generated mazes? The results cannot be explained by the previous discussion of computational complexity. Though we do not have enough theory to support the results, the next chapter investigates the mechanism behind the effectiveness of RTBS.

5 HEURISTIC TOPOGRAPHIES

The RTBS performance, measured in 4.2, is considered to result from the difference between the RTUS and RTBS problem spaces. Our expectation is that the effectiveness of RTBS can be determined by comparing the two problem spaces. Unfortunately, the problem spaces (especially the RTBS problem spaces) of example problems, which are utilized in Section 4, are too large to examine with existing computer systems. (Appendix A will present, however, the collected information concerning the heuristic topography of randomly generated mazes and 8-puzzles.) This section thus provides a complete example of topographical changes between the RTUS and RTBS problem spaces to provide better understanding of the RTBS performance.

Fig. 8 displays two RTUS problem spaces by 15 × 15 mazes. Fig. 8a represents the maze with a single wall (we call this maze *Bar*), while Fig. 8b includes three walls (we call this maze *Hole*). In both mazes, the initial state is represented by *I*, and the goal state by *G*.



- I : Initial state
- G : Goal state
- : Obstacles
- ▒ : Heuristic depressions in the RTUS problem space
- a,b,c,d : Pairs of problem solvers' states resulting from heuristic depressions in the RTBS problem space
- \$.# : Circumference (both \$ and #) and exit (#)

Fig. 8. Two mazes.

8. This fact shows that two problem solvers (one from the entrance and the other from the exit) cannot cooperatively solve a maze. The most efficient way is allowing only one problem solver to move. This phenomenon was observed again and again in our previous research on MTS [8], [9].

The locally maximal heuristic depressions in the RTUS problem spaces are represented by dark areas. In both mazes, heuristic depressions are spread between the initial state and walls. Table 2 summarizes the *width*, *depth*, *capacity*, *circumference*, *exit*, and *number* of heuristic depressions. The *width* represents the number of states in a heuristic depression. The *depth* indicates the maximum difference between the heuristic value of any state in the depression and that of the *exit*. Since the heuristic depression is defined as a set of states whose heuristic distances can be equal to those of surrounding states, the *depth* might be 0. Note that, to reflect the width of the depression to the capacity, the depth is defined as $\sum_i (\text{depth}(i) + 1)$ where i represents a state in the depression. The *circumference* represents the number of states, each of which belongs to the depression but is adjacent to the outside of the depression. We call a state an *exit* of a depression, if the state is a part of the circumference, and if its heuristic value is equal to that of the adjacent outside state. In Fig. 8a, for example, the *circumference* is represented by \$ and #, and the exit by #.

TABLE 2
HEURISTIC DEPRESSIONS IN THE TWO MAZES

Maze \ Problem Space	Bar					Hole					
	Width	Depth	Capacity	Circumference (Exit)	Number	Width	Depth	Capacity	Circumference (Exit)	Number	
RTUS	61	7	201	14 (2)	1	79	13	641	2 (2)	1	
RTBS	a	1	1	2	1 (1)	2	37	1	50	26 (2)	2
	b	63	1	76	52 (2)	2	2381	8	7379	579 (4)	2
	c						23	1	29	18 (1)	8
	d						1	1	2	1 (1)	4

Recall that the RTBS state is composed by a combination of two RTUS states. Thus, the heuristic depressions in the RTBS problem space are also caused by the positions of two problem solvers. For example, a pair of positions across the wall creates a heuristic depression. Table 2 represents all locally maximal heuristic depressions in the RTUS and RTBS problem spaces. To compute all depressions, an exhaustive search was performed in both problem spaces. Pairs of problem solver positions, which cause the heuristic depressions, are indicated as *a*, *b*, *c*, *d* both in Fig. 8 and Table 2. The major observations obtained from Table 2 are as follows.

- In the RTUS problem space, each maze contains one deep heuristic depression.
- In the RTBS problem space, the heuristic topography changes significantly. In *Bar*, both the depth and capacity of the depression decrease dramatically, compared to the RTUS problem space. On the other hand, in *Hole*, the depth decreases, while the capacity increases on the order of square.

Intuitively, as the width and depth of heuristic depressions increase, it becomes difficult to get out from them. In such a situation, the capacity can indicate the cost of the search. However, wide and shallow depressions with a long circumference (especially with a wide exit) are not too

hard to get out, even if the depressions are considerably wide. In RTBS, since the problem space is widened, the numbers of states in heuristic depressions also increase. This leads to a problem that is difficult to solve. Contrarily, since various paths are created, heuristic depressions in RTBS become shallow. This leads to a problem that is easy to solve. These effects are combined and appear differently in different problems. We measured the number of moves of the RTBS algorithms on two mazes in Fig. 8. In *Bar*, the number of moves of LRTA*/B is 2/3 to 1/2 that of LRTA*, while in *Hole*, it is 10 times greater than that of LRTA*.

6 CONCLUSION

Three complete algorithms for *real-time bidirectional search* (RTBS) have been proposed: RTA*/B and LRTA*/B as centralized RTBS algorithms which perform efficiently in uncertain situations, and MTS/B as a decoupled RTBS algorithm which is superior in clear situations. Compared to *real-time unidirectional search* (RTUS), RTBS significantly reduces the number of moves in 15- and 24-puzzles, but increases it in randomly generated mazes. It appears that the superiority or inferiority of the RTUS and RTBS algorithms depends on the topographical changes between the RTUS and RTBS problem spaces.

Several issues still remain unresolved. In previous research, where off-line search has primarily been studied, heuristic functions have been investigated from the probabilistic point of view [18]. However, as shown in this paper, the performance of real-time search is sensitive to heuristic depressions. This is because, in real-time search, erroneous decisions seriously affect the consequent problem solving behavior. We are now aware that heuristic topography is crucial to understanding the real-time search performance. It is also required to establish a theory behind the topographical changes of problem spaces.

This work can be considered as the first step toward coordinated problem solving. RTBS is not the only way for organizing two problem solvers [11]. Another possible way is to have both problem solvers start from the initial state and move toward the goal state. The problem space then becomes different from the RTBS case. This means that the selection of the problem solving organization is the selection of the problem space, which determines the baseline of the organizational efficiency; once a difficult problem space is selected, the local coordination among problem solvers hardly overcome the deficit. The theory behind the RTBS performance will provide a computational basis for coordinating and organizing multiple problem solvers.

Let us revisit the example at the beginning of this paper. The two robots first make decisions independently to move toward each other. However, the problem is hardly solved. To overcome this inefficiency, the robots then introduce centralized decision making to choose the appropriate robot to move next. They think that two is better than one, because the two robot organization has more freedom for selecting actions; better actions can be selected through sufficient coordination. However, the result appears miserable. The robots are not aware of the changes that have occurred in their problem space.

APPENDIX A HEURISTIC DEPRESSIONS IN MAZES AND n -PUZZLES

Why could RTBS be effective for n -puzzles, but not for randomly generated mazes? Surprisingly enough, these problems have been repeatedly utilized without discussing differences. This appendix examines heuristic depressions in these problems.

1) Maze

Table 3a represents heuristic depressions in a randomly generated maze. The maze is created by replacing 35% of the states with obstacles. The initial and goal states are placed so that the Manhattan distance between them is 50 (i.e., the direct path consists of 50 edges). In Table 3a, h represents the heuristic distance between the exit of a heuristic depression and the goal. The row for "0-5" represents the widest heuristic depression, whose heuristic value at the exit is between 0 and 5. From this table, it is observed that heuristic depressions can exist everywhere in the problem space. Compared to n -puzzles, it is observed that the depressions are deep and their exits are quite narrow. The topography of a randomly generated maze is like steep mountains.

TABLE 3
HEURISTIC DEPRESSIONS IN MAZES AND 8-PUZZLES

h	Width	Depth	Capacity	Circumference (Exit)	Number
0-5	1	1	2	1 (1)	1
6-10	6	2	10	3 (1)	1
11-15	12	6	45	3 (1)	1
16-20	22	5	60	7 (1)	1
21-25	46	7	192	7 (1)	1
26-30	14	5	44	4 (1)	1
31-35	16	4	42	4 (1)	1
36-40	18	10	119	2 (1)	1
41-45	35	6	124	5 (1)	1
46-50	25	9	114	6 (1)	1
51-55	36	7	129	6 (1)	1

(a) Maze

h	Width	Depth	Capacity	Circumference (Exit)	Number
5	43	2	65	23 (1)	2
6	531	4	883	238 (12)	1
7	2994	4	4894	1420 (50)	1
8	5231	4	8097	2546 (254)	1
9	789	3	1176	392 (90)	1
10	562	2	802	292 (40)	1
11	81	1	108	48 (28)	1
12	71	2	85	38 (4)	1
13	81	3	124	42 (8)	1
14	18	1	21	11 (7)	2
15	17	1	18	10 (8)	1
16	4	0	4	3 (3)	2

(b) 8-Puzzle

2) n -Puzzle

Table 3b represents heuristic depressions in an 8-puzzle. In Table 3b, h also represents the distance between the exit of a heuristic depression and the goal. Each row shows the widest heuristic depression where the heuristic distance from its exit to the goal is h . From this table, it is observed that wide depressions spread at the middle elevation. In n -puzzles, swapping two pairs of tiles can create a somewhat deep depression. However, Table 3b shows that the average depth is less than 2, which means that the deep area is quite limited in a depression. The circumference of depressions (especially the exit) is substantially wide compared to a maze. Wide and shallow depressions exist with a large circumference; the topography of an 8-puzzle is like a vast plateau. The

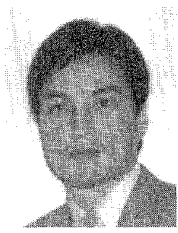
heuristic topography of 15- and 24-puzzles are considered to be similar, but cannot be confirmed because of the sizes of their problem spaces.

ACKNOWLEDGMENTS

The author wishes to thank Richard Korf, Kazuhiro Kuwabara, and Makoto Yokoo for their helpful discussions. This paper is the extended version of the author's previous conference paper [12].

REFERENCES

- [1] F. Chimura and M. Tokoro, "The Trailblazer Search: A New Method for Searching and Capturing Moving Targets," *AAAI-94*, pp. 1,347-1,352, 1994.
- [2] D. de Champeaux and L. Sint, "An Improved Bidirectional Heuristic Search Algorithm," *J. ACM*, vol. 24, no. 2, pp. 177-191, 1977.
- [3] D. de Champeaux, "Bidirectional Heuristic Search Again," *J. ACM*, vol. 30, no. 1, pp. 22-32, 1983.
- [4] E.H. Durfee, V.R. Lesser, and D.D., Corkill, "Coherent Cooperation among Communicating Problem Solvers," *IEEE Trans. Computers*, vol. 36, pp. 1,275-1,291, 1987.
- [5] M.S. Fox, "An Organizational View of Distributed Systems," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 11, no. 1, pp. 70-80, 1981.
- [6] L. Gasser, N. Rouquette, R. Hill, and J. Lieb, "Representing and Using Organizational Knowledge in DAI Systems," L. Gasser and M.N. Huhns, eds., *Distributed Artificial Intelligence*, vol. 2. London: Pitman, pp. 55-78, 1989.
- [7] M.P. Georgeff and A.L. Lansky, "Reactive Reasoning and Planning," *AAAI-87*, pp. 677-682, 1987.
- [8] T. Ishida and R.E. Korf, "Moving Target Search," *IJCAI-91*, pp. 204-210, 1991.
- [9] T. Ishida, "Moving Target Search with Intelligence," *AAAI-92*, pp. 525-532, 1992.
- [10] T. Ishida, L. Gasser, and M. Yokoo, "Organization Self-Design of Distributed Production Systems," *IEEE Trans. Knowledge and Data Engineering*, vol. 4, no. 2, pp. 123-134, 1992.
- [11] T. Ishida, "Towards Organizational Problem Solving," *IEEE Int'l Conf. Robotics and Automation*, pp. 839-845, 1993.
- [12] T. Ishida, "Two is not Always Better than One: Experiences in Real-Time Bidirectional Search," *Int'l Conf. Multi-Agent Systems (ICMAS-95)*, pp. 185-192, 1995.
- [13] T. Ishida and R.E. Korf, "A Moving Target Search: A Real-Time Search for Changing Goals," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 6, pp. 609-619, 1995.
- [14] R.E. Korf, "Depth-first Iterative-Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence*, vol. 27, no. 1, pp. 97-109, 1985.
- [15] R.E. Korf, "Real-Time Heuristic Search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189-211, 1990.
- [16] J.B.H. Kwa, "An Admissible Bidirectional Staged Heuristic Search Algorithm," *Artificial Intelligence*, vol. 38, pp. 95-109, 1989.
- [17] V.R. Lesser, "An Overview of DAI: Viewing Distributed AI as Distributed Search," *JSAI J.*, vol. 5, no. 4, pp. 392-400, 1990.
- [18] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, Mass.: Addison-Wesley, 1984.
- [19] I. Pohl, "Bi-directional Search," *Machine Intelligence*, vol. 6, pp. 127-140, 1971.
- [20] G. Politowski and I. Pohl, "D-node Retargeting in Bi-directional Heuristic Search," *AAAI-84*, pp. 274-277, 1984.
- [21] M.E. Pollack and M. Ringuette, "Introducing the Tileworld: Experimentally Evaluating Agent Architectures," *AAAI-90*, pp. 183-189, 1990.
- [22] S. Russell and E. Wefald, *Do the Right Thing*. MIT Press, 1991.



Toru Ishida received the BE, MEng, and DEng degrees from Kyoto University, Kyoto, Japan, in 1976, 1978, and 1989, respectively. He is currently a professor at the Department of Information Science, Kyoto University, Japan. From 1978 to 1993, he was a research scientist at the NTT Laboratories. He was also a visiting research scientist at the Department of Computer Science, Columbia University from 1983 to 1984. Since 1985 he has been working in the area of distributed artificial intelligence. Dr.

Ishida is a member of IPSJ, JSAI, JSSST, IEICE, ACM, and AAAI.