Two is not Always Better than One: Experiences in Real-Time Bidirectional Search

Toru Ishida
Department of Information Science
Kyoto University
Kyoto, 606, JAPAN
ishida@kuis.kyoto-u.ac.jp

Abstract

This paper investigates *real-time bidirectional search (RTBS)* algorithms, where two problem solving agents, starting from the initial and goal states, physically move toward each other. To evaluate the RTBS performance, two kinds of algorithms, *centralized RTBS* and *decoupled RTBS*, are proposed and are compared to *real-time unidirectional search (RTUS)*. 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 RTUS for 15- and 24-puzzles but not for randomly generated mazes. It will be shown that the selection of the multi-agent 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 overcomes the deficit.

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 organizational performance issues of *real-time bidirectional search*, which will be investigated throughout this paper.

In traditional *off-line bidirectional search* [Pohl, 1971; de Champeaux and Sint, 1977; de Champeaux, 1983], the problem solver located at the initial state expands its wave front toward the goal state. 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 paper studies *real-time bidirectional search (RTBS)*, and investigates 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); this limitation means that planning and execution must be interleaved. 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 formal step towards organizational problem solving [Fox, 1981; Ishida *et al.*, 1992], viewing DAI problems as distributed search [Lesser, 1990]. In RTBS, two problem solvers starting from the initial and goal states physically move toward each other. Unlike the off-line bidirectional search, 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 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 based on available RTUS algorithms, i.e., *RTA* (Real-Time A*)*, *LRTA* (Learning Real-Time A*)* [Korf, 1990] and *MTS (Moving Target Search)* [Ishida and Korf,

1991; 1995; Ishida, 1992a]. An experimental evaluation has been made using mazes and n-puzzles, the most common examples in the area of search.

RTBS Algorithms

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 [Pohl, 1971]. 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.

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.

In RTBS, control strategies are crucial for characterizing the autonomy of the problem solvers: RTBS algorithms can be classified into the following two categories depending on the autonomy of the problem solvers. In *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. In *decoupled control*, on the other hand, 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 this section, the control strategy simply selects forward and backward moves alternately.

In addition to 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. In *shared learning*, the forward and backward problem solvers share heuristic values, $h(x,y)$, and maintain the values together. In *distributed learning*, 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.

In the following two subsections, only the two extremes of the four possible combinations (*centralized control with shared learning* and *decoupled control with distributed learning*) will be investigated. We simply call the two extremes *centralized RTBS* and *decoupled RTBS*. The 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.

Let us take an n-puzzle example. The RTBS algorithm utilizes two game boards. At the beginning, one plate indicates the initial state and the other indicates the goal state. The aim in this case is to achieve identical 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 plate independently.

Centralized RTBS

Below, we describe a centralized RTBS algorithm called LRTA*/B, which is a bidirectional version of LRTA*. The positions of two problem solvers are represented by x (forward) and y (backward), 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:

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

   (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.)

RTA*/B is similar to LRTA*/B, but it utilizes the second minimum value for updating heuristic distances. Both LRTA*/B and RTA*/B are complete as follows.

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.

The proof is obtained by showing that the centralized RTBS algorithms is equivalent to applying the corresponding RTUS algorithms to the *RTBS problem space*.

Decoupled RTBS

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 $h_b(x,y)$ may 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:

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

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

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

(c) 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:

(a) Calculate $h_b(x',y)$ for the forward problem solver's new position x'.

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

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

(c) 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:* Analogous to the forward move, described above.

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. The same assumption is even required in MTS/B, however, where the two problem solvers try to meet each other.

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.

Theorem 2 demonstrates the completeness of MTS/B. The proof can be derived like that for MTS [Ishida and Korf, 1991].

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.

Computational Complexity

Figure 1(a) illustrates the search tree for RTUS. Each node represents a position of the problem solver. An example path from the initial state to the goal state is represented by the thicker 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 RTBS performance is to assume that RTBS algorithms solve a totally different problem from RTUS; i.e., the difference between RTUS and RTBS is not the number of problem solvers, but their problem spaces.

Let an RTBS state be a pair of RTUS states of two problem solvers: where x and y are RTUS states. Then the pair of RTUS states (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 be the RTUS initial state, and g be the RTUS goal state; then, the pair of RTUS states (i,g) becomes the RTBS initial state. The goal state of RTBS requires both problem solvers to share the same RTUS state, i.e., consequently, all (x,y), where $x = y$, are RTBS goal states. Thus, the RTBS goal state is not unique, i.e., when there is one RTUS goal state and 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.

Figure 1(b) 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 the moves of the two problem solvers before meeting each other. Then, the number of generated states can be represented by $(B_f+B_b)\times D$.

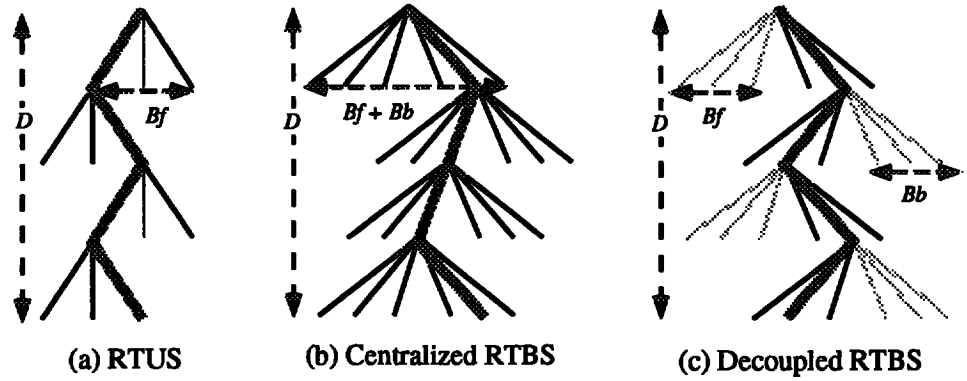


Figure 1: Comparison of Computational Complexities

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. Figure 1(c) 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). Therefore the selected action may not be the best among all the possible $B_f + B_b$ moves of the two problem solvers. Let D be the sum of the moves of the two problem solvers. Since decoupled RTBS uses partial information, D increases in contrast 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 in contrast to centralized RTBS.

Measurement on Typical Problems

The RTUS and RTBS algorithms are applied to two typical path finding problems.

(1) *Randomly Generated Mazes*

Mazes have been used for testing search algorithms, in particular bidirectional search [Kwa, 1989]. In this measurement, we represent a maze as a grid space, replacing an 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 up, down, right or left in the grid space. The Manhattan distance is used as the heuristic function.

(2) *n-Puzzles*

n-puzzle problems, especially 15-puzzles, have been appearing repeatedly in bidirectional search literature [Pohl, 1971; de Champeaux *et al.* 1977; Politowski and Pohl, 1985]. In this evaluation, with 15-puzzles, the first ten problems of the 100 problems presented in [Korf, 1985] are used. A further evaluation is based on ten randomly generated 24-puzzles. The distance between two n-puzzle states is estimated by summing the Manhattan distances between the locations of all the tiles.

Figures 2 to 4 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 time taken by the two problem solvers. Note that *the number of generated states determines the planning time, and the number of moves determines the execution time.* Programs are written in Allegro Common Lisp running on a SparcStation2. The results in Figures 2 and 3 are obtained by averaging 100 trials, while Figure 4 uses only 30 trials due to the limitations of computing resource.

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 (i.e., heuristic functions return accurate values), decoupled RTBS performs better.* From the maze experiences in Figure 2, when the obstacle ratio is low, there is not much difference in the number of moves between centralized and decoupled RTBS. This means that, in clear situations, two problem solvers can independently make decisions without losing efficiency. However, since centralized RTBS expands twice as many states as decoupled RTBS, the planning time for decoupled RTBS is almost 1/2 that of centralized RTBS.
- *In uncertain situations (i.e., heuristic functions return inaccurate values), centralized RTBS performs*

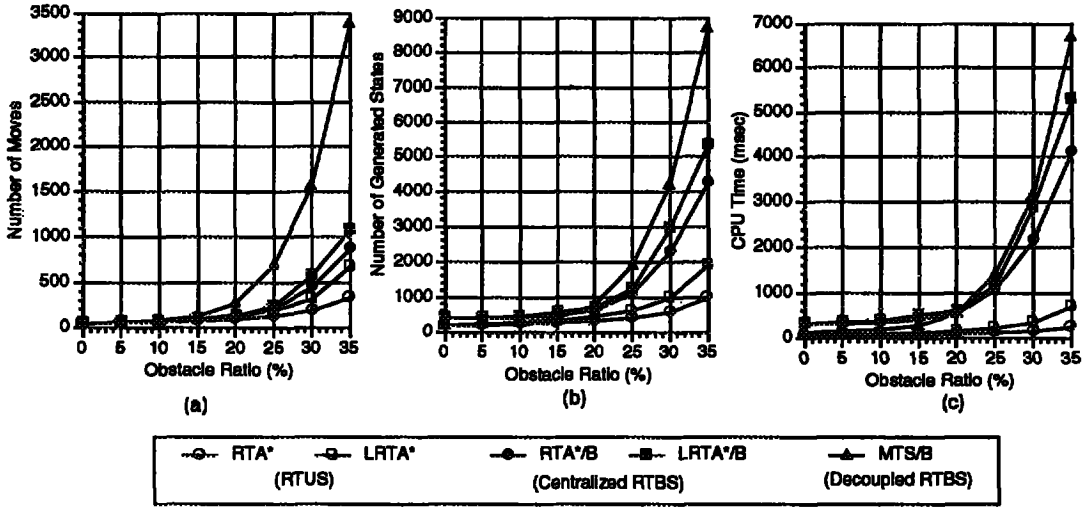


Figure 2: Performance for Randomly Generated Mazes

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. The n-puzzle experiences in Figures 3 and 4,[1] also show that the number of moves for centralized RTBS is about 1/2 that of decoupled RTBS.

By comparing RTUS and RTBS performances, interesting observations can be made.

- *In clear situations, both RTUS and RTBS have a similar performance.* Our maze experiences in Figure 2 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 optimal solutions can be easily obtained by applying various search techniques.
- *In uncertain situations, the superiority or inferiority of RTUS and RTBS relies heavily on the problem type.* RTBS can solve 15- and 24-puzzles with fewer moves than RTUS as in Figures 3 and 4; the number of moves for centralized RTBS is approximately halved for 15-puzzles and reduced by a factor of 6 for 24-puzzles compared to RTUS. In randomly generated mazes, however, when the obstacle ratio becomes more than 10%, Figure 2 shows that 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.

[1] In mazes, when the obstacle ratio becomes 35%, while the optimal path length is 60 to 80, the number of moves becomes 400 to 4000, 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, 2000 to 5000 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.

Heuristic Topographies

The performance of real-time search depends greatly on the topography of the estimated problem space. This is because, in real-time search, the problem solver always commits to its decision, and thus will seriously affect the consequent problem solving process. To explain the problem solver's behavior, we define a *heuristic depression*, with respect to a single goal state, as a set of connected states whose heuristic values are less than or equal to those of the set of immediate and completely surrounding states. 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 more frequently in the problem space.

The real-time problem solver 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.

We believe the RTBS performance, measured in the previous section, results from the difference between

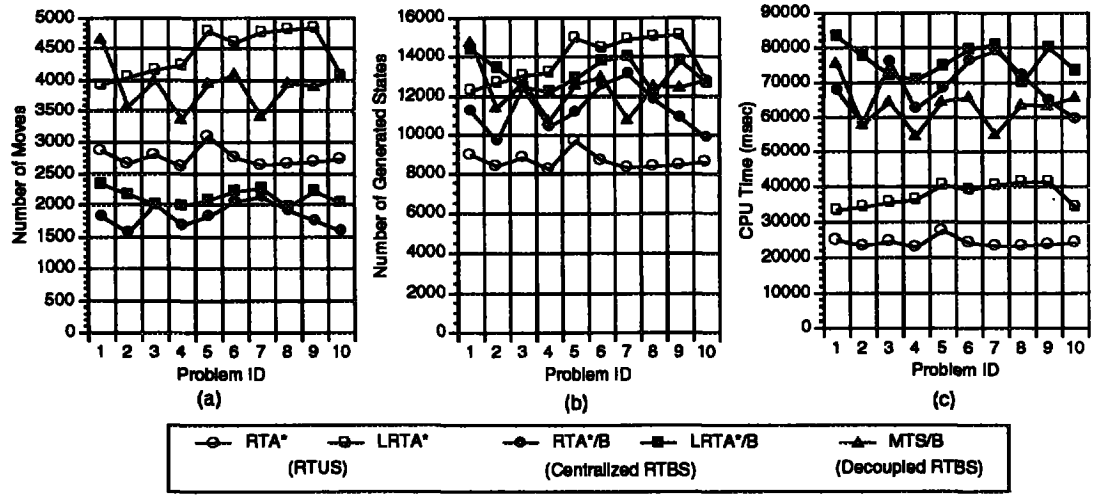


Figure 3: Performance for 15-Puzzles

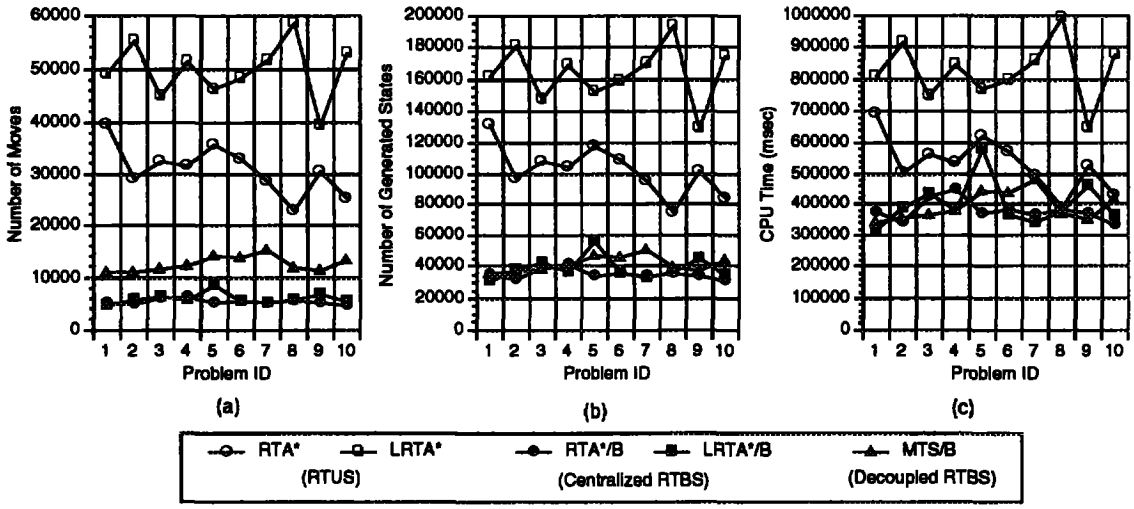
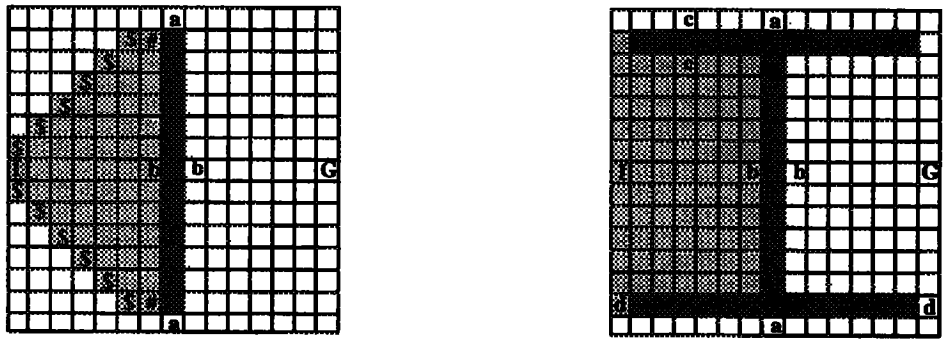


Figure 4: Performance for 24-Puzzles

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 the example problems are too large to examine with existing computer systems. This section therfore provides a complete example of topographical changes between the RTUS and RTBS problem spaces to provide better understanding of the RTBS performance.

Figure 5 displays two RTUS problem spaces of 15 × 15 mazes. Figure 5(a) represents the maze with a single wall (we call this maze *Bar*), while Figure 5(b) includes three walls (we call this maze *Hole*). In both mazes, the initial state is represented by I, and the goal state by G.

The 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 1 summarizes the *width*, *depth*, *capacity*, *circumference*, *exit* and *number* of heuristic depressions. The *width* represents the number of states in a heuristic 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. 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 are less than or equal to those of surrounding states, the *depth* can be 0. The *capacity* is defined as $\sum_i (depth(i) + 1)$ where i represents a state in the depression. This is to reflect the width of the depression to the capacity. The *circumference* represents the number of states, each of which belongs to the depres-



(a) Bar (b) Hole

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

Figure 5: Two Mazes

sion but is adjacent to the outside of the depression. In Figure 5(a), for example, the *circumference* is represented by $ and #, and the *exit* by #.

Recall that the RTBS state is composed of a combination of two RTUS states. Thus, the heuristic depressions in the RTBS problem space are caused by the positions of two problem solvers. For example, a pair of positions across the wall creates a heuristic depression. Table 1 represents all locally maximal heuristic depressions in the RTUS and RTBS problem spaces. To compute all depressions, an exhaustive search was performed in both the RTUS and RTBS problem spaces. Pairs of problem solver positions, which cause the heuristic depressions, are indicated as a, b, c, d both in Figure 5 and Table 1. The major observations from Table 1 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 by the factor of 10^2.

Why is RTBS effective for 15- and 24-puzzles, but not for randomly generated mazes? Intuitively, as the width and depth of heuristic depressions increase, it becomes difficult to get out of 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 fairly wide. In RTBS, since the problem space is widened, the number of states in heuristic depressions also increases. This leads to a problem that is difficult to solve. On the other hand, 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 RTUS and RTBS algorithms on the two mazes in Figure 5. In *Bar*, the number of moves of LRTA*/B is between a half and two third compared to LRTA*, while in *Hole*, it is 10 times greater than that of LRTA*.

Conclusion

This paper has proposed *real-time bidirectional search (RTBS)* algorithms, and has investigated their performance. 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 [Pearl, 1984]. However, as shown in this paper, the performance of real-time search is sensitive to heuristic depressions. We are now aware that heuristic topography is crucial to understanding the real-time search performance, and that we must establish a theory behind the topographical changes between the RTUS and RTBS problem spaces.

This work can be considered as the first step toward organizational problem solving [Ishida, 1992b; Ishida, 1993]. Note that RTBS is not the only way to organize two problem solvers. Another possible way is to have

Maze / Problem Space		Bar Width	Bar Depth	Bar Capacity	Bar Circumference (Exit)	Bar Number	Hole Width	Hole Depth	Hole Capacity	Hole Circumference (Exit)	Hole 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

Table 1: Heuristic Depressions in the Two Mazes

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 multi-agent 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 overcomes the deficit. The theory behind the RTBS performance will provide a computational basis for coordinating and organizing multiple problem solving agents.

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.

Acknowledgments

The author thanks Richard Korf, Kazuhiro Kuwabara and Makoto Yokoo for their helpful discussions.

References

[de Champeaux and Sint, 1977] D. de Champeaux and L. Sint, "An Improved Bidirectional Heuristic Search Algorithm, " *J. ACM*, Vol. 24, No. 2, pp. 177 - 191, 1977.

[de Champeaux, 1983] D. de Champeaux, "Bidirectional Heuristic Search Again, " *J. ACM*, Vol. 30, No. 1, pp. 22-32, 1983.

[Fox, 1981] M. S. Fox, "An Organizational View of Distributed Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 11, No. 1, pp. 70–80, 1981.

[Ishida and Korf, 1991] T. Ishida and R. E. Korf, "Moving Target Search," *IJCAI-91*, pp. 204-210, 1991.

[Ishida, 1992a] T. Ishida, "Moving Target Search with Intelligence," *AAAI-92*, pp. 525-532, 1992.

[Ishida, 1992b] T. Ishida, "The Tower of Babel: Towards Organization-Centered Problem Solving," *11th Distributed Artificial Intelligence Workshop*, pp. 141-153, 1992.

[Ishida *et al.*, 1992] T. Ishida, L. Gasser and M. Yokoo, "Organization Self-Design of Distributed Production Systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 2, pp. 123-134, 1992.

[Ishida, 1993] T. Ishida, "Towards Organizational Problem Solving," *IEEE International Conference on Robotics and Automation*, pp. 839-845, 1993.

[Ishida and Korf, 1995] T. Ishida and R. E. Korf, "A Moving Target Search: A Real-Time Search for Changing Goals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995 (to appear).

[Korf, 1985] R. E. Korf, "Depth-first Iterative-Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence*, Vol. 27, No. 1, pp. 97-109, 1985.

[Korf, 1990] R. E. Korf, "Real-Time Heuristic Search," *Artificial Intelligence*, Vol. 42, No. 2-3, pp. 189-211. 1990.

[Kwa, 1989] J. B. H. Kwa, "An Admissible Bidirectional Staged Heuristic Search Algorithm," *Artificial Intelligence*, Vol. 38, pp. 95-109, 1989.

[Lesser, 1990] V. R. Lesser, "An Overview of DAI: Viewing Distributed AI as Distributed Search," JSAI Journal, Vol. 5, No. 4, pp392-400, 1990.

[Pearl, 1984] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, Mass., 1984.

[Pohl, 1971] I. Pohl, "Bi-directional Search," *Machine Intelligence*, Vol. 6, pp. 127-140, 1971.

[Politowski and Pohl, 1985] G. Politowski and I. Pohl, "D-node Retargetting in Bi-directional Heuristic Search," *AAAI-84*, pp. 274-277, 1984.