Solving Problems under Uncertainty Paradigm

Makoto Sakuta*, Hiroyuki Iida*, Jin Yoshimura**
Departments of Computer Science* and Systems Engineering**, Shizuoka University

Abstract— **This paper proposes a novel paradigm for solving search problems with uncertainty (hereafter Uncertainty Paradigm). A search under this paradigm is denoted as Uncertainty-Paradigm Search (UPS). UPS is a new forward calculation method of true solutions for dynamic games. In UPS, a metaposition instead of a normal position is employed as a node of a game tree as well as a metamove instead of a normal move. A metaposition is a hybrid of possible positions, while a metamove is a hybrid of possible moves that is used as an arc between a metaposition and its metaposition successor. Deterministic solving of search problems with uncertainty is thus resolved into a plain AND/OR tree search for multi-person games as well as for single-agent puzzles. Theoretically, UPS is applicable in solving any puzzles or endgame problems for incomplete-information games. We implemented UPS in a computer program of a chess-like game with uncertainty, i.e., Kriegspiel-like variant of Japanese chess, and Counterfeit-Coin Problem. We then demonstrated the effectiveness of the proposed idea.**

Keywords— **problem solving, game-tree search, uncertainty paradigm, uncertainty paradigm search, dynamic game.**

I. Introduction

We first claim the problem statement on the solving problems under uncertainty situation, and then discuss the conventional approach for solving problems concerned.

A. The problem statement

A game has been recognized a model of many social and economic competitive activities. Many studies have focused on the analysis of games without uncertainty, which are complete-information games of no chance. In the domain of computer-game research, most efforts have indeed been devoted to programming perfect-information games such as chess.

However, *uncertainty* is known to be a major factor in social and economic competitive behaviors. Therefore, games with uncertainty are much better than games without uncertainty (e.g., complete-information games) for reflecting realistic social and economic competitive models. The study on incomplete-information games or more generally problems with uncertainty is thus becoming more and more important.

When solving problems with uncertainty such as mating problems of games with incomplete information, the conventional game theory suggests us to build a search tree as a game in extensive form and combine it with an information set, and possibly find a reasonable strategy. The information set is somewhat apart from the tree structure, so the search cannot be resolved into a plain tree search.

Then, a research question arises.

How can we solve problems with uncertainty in the framework of a plain tree search?

B. General backgrounds of decisions and games under uncertainty

Human decisions under uncertainty has been extensively studied in many social sciences (e.g., economics, psychology, management), and engineering sciences (e.g., operations research, control theory). In the social sciences, such optimization methods include decision theory and game theory. These techniques have been applied in almost every area of the social sciences, including economics, politics, psychology, sociology, and management sciences. In the operations research/engineering sciences, dynamic programming, and stochastic modeling have been developed and applied almost exclusively to physical/mechanical controls (see [17][18][45]). Expected utility theory: In all optimization methods, expected utility theory is commonly considered to be a basic principle of human decision making ([9][39][42][34]). The basic principle of expected utility is very simple: Given the utility of wealth, rational choice should maximize expected utility rather than expected monetary gain. Expected utility theory has been studied extensively and is well formulated mathematically while its axiomatic foundation was developed ([10][34]). It has also been applied to many situations involving human decisions, e.g., economic wealth optimization, social choice theory, social psychology, risk assessment, criminology (e.g., [9][14][16][19][38][43][48]). Expected utility theory is often considered a central principle of human decision under uncertainty and risk ([11][25][39]).

In expected utility theory, decision a tree is often built to assess the value of each options. In such a tree some branches (arcs) are often characterized with probability and the overall value (utility, that is evaluation function) of each options are evaluated by the weighted average of final outcomes with its probability. Here we evaluate the original options based on the probabilistic tree. Thus expected utility is a type of probabilistic tree search, concerning the original options.

Dynamic optimization: Dynamic optimization is a method of optimizing decisions from time to time, in contrast with "static" optimization, in which only one-time decision is considered ([27][45]). Dynamic optimization finds optimal paths or trajectories, whereas static models find optimal points. Since behavior also changes between time periods, dynamic optimization is a useful technique to describe the dynamics and flexibility of behavior ([27]). Dynamic optimization has been developed mostly in engineering and applied mathematics, and its application to social/economic sciences is rather limited (but see [8][26]).

Corresponding author. Email: iida@cs.inf.shizuoka.ac.jp.

There are two techniques of dynamic optimization: dynamic programming for discrete state variables, and optimal control theory for continuous state variables ([36][45]). Dynamic programming is a simple algorithm based on the principle of optimality, and is directly applicable to computer programming ([4][6]).

Dynamic optimization is the type of probabilistic search tree with optimal decisions at every time. Here the tree is vastly large because of exponential branching processes. Therefore, it is impossible to evaluate the optimal path (sequential decisions) from the beginning of the time steps (forward calculation). By backward calculation, it is only possible to find the optimal decisions, trimming all the unnecessary (suboptimal) paths.

Game theory: Game theory has a background identical to that of expected utility theory ([34]). Therefore, game theory is often considered a subset of decision theory ([24]). Game theory considers a play against other players, in contrast to (statistical) decision theory that describes a play against uncertain environments. Game theory has been extensively studied in social science contexts ([43][44]). It has also been applied to animal conflicts and cooperation, leading to the concept of evolutionarily stable strategies, or ESS's (see [28][29][46]).

Dynamic games combining games and dynamic optimization have also been developed for behavioral problems ([20][27][49]). Here we should note that the conventional dynamic game assumes that the optimal game solution is valid in each step of dynamic programming calculation in order to get an ad-hoc solution. However, in true dynamic games, decisions have to be made simultaneously under the changing game situations.

Thus probabilistic tree search is fundamental to decision theory and dynamic optimization (programming). The difference between decision theory and dynamic programming is that decision theory only concerns one-time decision and dynamic programming deals with sequences of decisions, Dynamic programming is also characterized with backward calculation to get the optimal solution. Game theory is the basic principle to get the plain tree search assuming the optimal moves by the opponent.

C. Conventional tree search and its criticism

We here summarize the conventional approach for finding a reasonable strategy in playing incomplete-information games, based on game theory[35].

A game in extensive form is represented by a topological tree or game tree Γ. It is a finite collection of nodes, called vertices, connected by lines, called arcs. In this paper, a vertex denotes a *position* and an arc denotes a *move*. A game has the starting vertex A, starting point or initial position.

For n-player game with chance, there are $n+1$ player sets as subsets of vertexes:

$S_0, S_1, \cdots, S_n$: player sets

Here, S_0 is related to chance moves. A probability distribution is defined at each vertex of S_0 among the immediate successors of this vertex. $S_1, \cdots, S_n$ are related to personal moves with correspond to the n players.

For each player i, information set S_i^j is defined as a set of vertexes that are undistinguishable for player i. S_i^j is then a subset of S_i. Each player knows only to which information set he belongs, but he cannot see his internal location within the information set.

Player i is said to have perfect information in Γ if his information sets S_i^j each consists of one element. The game Γ is said to have perfect information if each player has perfect information in Γ. In this paper, we discuss about a game for which at least one player has one or more information sets with more-than-one element, i.e., game with incomplete-information.

A node of game tree is a position in the conventional approach based on game theory. However, since it is impossible to solve problems with uncertainty only using the information of plain tree, the information sets are used somewhat apart from the tree structure. For example, the two positions that are not siblings may even result in the same information set. Therefore, the search cannot be resolved into plain tree search, so developing a program for solving problems with uncertainty is usually much more difficult than doing so for perfect-information problems.

The idea of information sets has been incorporated in many studies on incomplete-information games ([12][22]).

The present contribution explores the deterministic solving of search problems with uncertainty, which can be resolved into a plain AND/OR tree search. We propose a novel paradigm for solving search problems with uncertainty (Uncertainty Paradigm). A search under this paradigm, denoted as Uncertainty-Paradigm Search (UPS), is discussed. It thus enables us to deterministically solve problems with uncertainty.

II. Uncertainty Paradigm

In this section, the notion of uncertainty paradigm is introduced with the concept of metaposition and metamove.

A. Uncertainty Paradigm

The idea of uncertainty paradigm is in a way a paradigm shift in game theory. This resembles the paradigm shift from classical dynamics to quantum dynamics, so several terms used here to discuss the uncertainty paradigm are derived from quantum mechanics or quantum chemistry.

Insightfully, the uncertainty paradigm recognizes uncertain situations as it really is, like in the same way we perceive the universe under uncertainty principles. Under the uncertainty paradigm, an uncertain situation is a hybrid of several certain situations. The uncertain situation is a delocalized state, while the definite situations in the uncertain situation are localized states.

In this paper, we assume a player in a game/puzzle to be the *solver* and other player(s) to be the *opponent*.
In order to solve problems with uncertainty deterministically, the solver has to resolve all situations irrelevant to the responses of the opponent. We also assume that the

opponent selects the optimal choice at any situation. This means that the opponent has the complete information, which may correspond to the best defense model in the incomplete-information games[12].

B. Metaposition and Metamove

We introduce the notion of *metaposition* and *metamove* in relation to the uncertainty paradigm.

B.1 Metaposition

A metaposition is a hybrid of positions that are undistinguishable for the solver.

Let Φ be a metaposition, φ_i be a position in the metaposition, and p_i be a coefficient that indicates probability. The metaposition is then represented as:

$$\Phi = \sum_{i}^{n_p} p_i \varphi_i. \tag{1}$$

Equation(1) describes the relation between the localized states φ_i and the delocalized states Φ. The number of the possible positions, denoted as n_p in Equation(1), can be recognized as the *uncertainty index*.

For deterministic solving, the probability coefficients are meaningless. Then, each coefficient denoted as p_i in Equation(1) is to be omitted. We thus have Equation(2). In future, when extending the proposed paradigm to probabilistic solving, p_i will have an important role.

$$\Phi = \sum_{i}^{n_p} \varphi_i \tag{2}$$

B.2 Metamove

A metamove is a move between two metapositions. We distinguish two cases: (1) metamove with complete information, and (2) metamove with incomplete information.

In the first case, the solver can see the definite metamoves with perfect information. In this case a metamove is one definite move among all legal moves. For instance, a pass move, if the rules allow players to play, is also a metamove. Moreover, a possibly illegal move, if the rules allow players to play, is also a metamove. Note that playing an illegal move will be important to gain information.

In the second case, the solver can only see clues of a metamove with incomplete information. In this case, a metamove is a hybrid of all possible moves at all positions in certain metapositions. Note that the solver cannot distinguish each metamove as well as metaposition.

In many games or puzzles, a metamove of the solver usually consists of a definite move. However, a metamove of the opponent is a definite move in some games while a hybrid metamove in the other games.

For every position φ_i of a metaposition, let n_{m_i} be the number of possible moves at position φ_i. There exists a set of possible moves:

$$M_i = \{m_{ij} \mid \text{possible moves of } \varphi_i,\ 1 \le j \le n_{m_i}\}$$

In some games where the solver can see the definite metamoves with complete information, a metamove is a definite move among all legal moves. For instance, if the rules allow each player to play a pass move, it can be a metamove. If the rules allow each to play a possibly illegal move, it also can be a metamove.

Let us classify two cases: (1) metamoves of the solver and (2) metamoves of the opponent.

For the first case, i.e., at a metaposition where the solver is to move, only the moves that are common in all positions form a metamove in the deterministic solving. Hence, the set of possible moves at metapositions is the intersection of all M_i. The intersection $\mathcal{M}^{solver}$ is then given by Equation(3).

$$\mathcal{M}^{solver} = \bigcap_{i}^{n_p} M_i \tag{3}$$

If the set of possible moves at a metaposition is empty, i.e., $\mathcal{M}^{solver} = \emptyset$, then the metaposition is a terminal node. Moreover, if it is a goal metaposition, the node is considered as being proved. Otherwise, the node is considered as being disproved.

Therefore, the metamove of the solver with complete information is an element of the set of possible moves at the metaposition. Thus we see the relation indicated by Equation(4).

$$\mu = m \in \mathcal{M}^{solver} \tag{4}$$

For the second case, i.e., at a metaposition where the opponent is to move, the solver knows the metamove after it has been made. So a metaposition may converge into a metaposition with less uncertainty according to whether the definite move that has been played is the allowed move. This converging is the special case of splitting of metapositions described later. However, unlike the normal splitting, it is not necessary to solve positions for which the move is not allowed since the solver can see that the opponent has already played it.

The set of possible moves at a metaposition where the opponent is to move, is the union of all M_i. A metamove of the opponent with complete information is an element of the set of possible moves at the metaposition. The union $\mathcal{M}^{opponent}$ is then given by Equation(5).

$$\mathcal{M}^{opponent} = \bigcup_{i}^{n_p} M_i \tag{5}$$

Thus we observe the relation indicated by Equation(6).

$$\mu = m \in \mathcal{M}^{opponent} \tag{6}$$

The solver can only see clues of the opponent's moves with incomplete information. In this case, the set of possible moves at a metaposition is the sum of all M_i, and the opponent's metamove is the hybrid of all possible moves at all positions in the metaposition. The solver cannot distinguish each move in a metamove as well as a metaposition.

$$\mathcal{M}^{uncertain} = \{m_{ij} \mid \text{possible moves of } \varphi_i, \; 1 \leq j \leq n_{m_i}, \; 1 \leq i \leq n_p\} \qquad (7)$$

Let μ be a metamove, n_m be the number of possible moves in the metamove, and q_{ij} be the coefficient that indicates probability of each move. Metamove μ is then given by Equation(8).

$$\mu = \sum_i^{n_p} \sum_j^{n_{m_i}} q_{ij} m_{ij} \qquad (8)$$

The number of possible moves in the metamove, denoted as n_m, is given by Equation(9).

$$n_m = |\mathcal{M}^{uncertain}| = \sum_i^{n_p} n_{m_i} \qquad (9)$$

For deterministic solving, the probability coefficients are meaningless and therefore can be omitted. Hence, we obtain simply Equation(10).

$$\mu = \sum_i^{n_p} \sum_j^{n_{m_i}} m_{ij} \qquad (10)$$

C. Diffusing and splitting of metaposition

The notion of metaposition is similar to the information set in game theory. In our approach, a metaposition is employed as a single node in a search tree. There is only one arc between two metapositions connected by a metamove, while there are possibly several arcs between two information sets connected by several moves. This is an essential difference from the concept of the information set.

Let φ and Φ be a position and metaposition, respectively. Then, $m(\varphi)$ denotes a position obtained after move m is played at position φ. Similarly, $\mu(\Phi)$ denotes a metaposition obtained after metamove μ is played at metaposition Φ.
Here to play a metamove at a metaposition is represented as:

$$\Phi^{child} = \mu(\Phi)$$

When a metamove μ at metaposition Φ is a definite move m, making the metamove is identical to making the move m at each position in the metaposition. Such relation is given by Equation(11).

$$\Phi^{child} = \sum_i^{n_p} m(\varphi_i) = \sum_i^{n_p^{child}} \varphi_i^{child} \qquad (11)$$

In this case, diffusing does not occur. On the contrary some converging may occur since the several $m(\varphi_i)$ happen to be the same position. In addition, as described before, there may occur converging of the metaposition after the definite opponent metamove. Hence, $n_p^{child} \leq n_p$.

When metamove μ at metaposition Φ is a hybrid of several moves, making the metamove corresponds to making all moves at each position of the metaposition. Consequently, the number of positions in the metaposition increases. This is a diffusion of the metaposition. In addition, some converging may occur since the several $m_{ij}(\varphi_i)$ happen to be the same position in this case. The relation is thus indicated by Equation(12).

$$\Phi^{child} = \sum_i^{n_p} \sum_j^{n_{m_i}} m_{ij}(\varphi_i) = \sum_i^{n_p^{child}} \varphi_i^{child} \qquad (12)$$

The number of positions in the metaposition increases up to n_p^{child}, i.e., at most n_m, given by n_m/n_p times as many as that of the parent metaposition n_p.

If the *solver* cannot find any clue in such metamoves, the uncertainty of the metaposition soon explodes combinatorially. However, there must be observations or clues that help the *solver*. By these observations a metaposition can be split into several metapositions with less uncertainty, i.e., metapositions that have less positions in it.

We define the **observable** as an element of which value the *solver* can get as a clue. An observable o always acts on the metaposition (and the positions in the metaposition) and returns the corresponding value automatically.

Given a position φ and metaposition Φ, $o(\varphi)$ represents a value returned by acting on position φ. Similarly, $o(\Phi)$ represents a value returned by acting on metaposition $\Phi = \sum_i^{n_p} \varphi_i$. Then, $o(\Phi)$ is classified as follows:

$$o(\Phi) = \begin{cases} v & \text{if for all } \; i \;\; (1 \leq i \leq n_p) \quad o(\varphi_i) = v \\ \text{uncertain} & \text{otherwise} \end{cases}$$

Let n_o be the number of observables, and o_l be each observable. Let $(t_{s1}, t_{s2}, \ldots, t_{sn_o})$ be a n_o-tuple of the indexes of observable values that represents s-th split $(1 \leq s \leq n_s^{child})$. For each observable o_l, let nv_l be the number of possible values for the observable, and $\{v_{lt} \mid 1 \leq t \leq nv_l\}$ be a set of possible values for the observable. The splitting of a metaposition is then represented as:

$$\Phi_s^{child} = \sum_{i_s}^{n_{p_s}^{child}} \varphi_{i_s}^{child} \qquad (1 \leq s \leq n_s^{child})$$

$$\text{such that} \;\; o_l(\Phi_s^{child}) = v_{lt_{sl}}$$

$$1 \leq n_s^{child} \leq \prod_l^{n_o} nv_l$$

$$\sum_s^{n_s^{child}} n_{p_s}^{child} = n_p^{child}$$

So the metaposition Φ^{child} splits into n_s^{child}-metapositions, i.e., $\{\Phi_1^{child}, \ldots, \Phi_{n_s^{child}}^{child}\}$. If there is only one n_o-tuple of possible indexes of observables for all positions in the child metaposition, the splitting does not occur, namely, $n_s^{child} = 1$.

The *solver* can guess which position actually is in a metaposition and which position is not by collecting the clues or observations as much as possible. By these observations the metaposition splits into several metapositions according as whether each position in the metaposition conforms to each observation or not.

Since the *solver* should accept this splitting in a passive manner, he has to solve all the split metapositions. Hence, it is an AND-splitting of metapositions even in case of the single-agent puzzles.

III. Uncertainty Paradigm Search

Various problems with uncertainty can be handled under the uncertainty paradigm. Here we categorize such problems in the viewpoint of the number of players participated in a game/puzzle.

Single-agent puzzles: In single-agent puzzles with complete information, search tree is an OR graph, which can be often recognized as an OR tree. The algorithm A* is well-known and representative for solving OR graph. However, in case of incomplete information under the uncertainty paradigm, a metaposition may AND-split into some metapositions while every node is basically an OR-node. So the search tree is resolved into AND/OR graph or tree. Since a move of the solver is usually definite in single-agent puzzles, the diffusion of metaposition does not occur. Therefore, the uncertainty does not increase at all but only decreases. If there might exist a single-agent puzzle with uncertain moves, the diffusion would occur of course in such a puzzle.

Two-person games or adversary-agent puzzles: In adversary-agent puzzles with complete information, search tree is an AND/OR graph, which can be often recognized as AND/OR tree. This AND/OR tree has the feature that OR nodes and AND nodes appear by turns. However, in case of incomplete information under the uncertainty paradigm, a metaposition may AND-split into some metapositions in both the solver's turn and opponent's turn, while every node of the solver is basically an OR node and every node of the opponent being an AND node. So the search tree is resolved into more generic AND/OR graph or tree. In this AND/OR tree, OR nodes and AND nodes do not necessarily appear by turns.

n-person games or n-agent puzzles ($n \geq 3$) Every node where the solver is to move is basically an OR node, while every node where the opponent is to move is basically an AND node. In addition, a metaposition may often AND-split into some metapositions. The search tree is also resolved into AND/OR graph or tree. However, the diffusion of metapositions is generally heavy and AND nodes have the large proportion of the whole nodes in this AND/OR tree. Therefore, it seems that solving such a problem is much harder.

Consequently, when solving all games/puzzles with uncertainty, search tree under the uncertainty paradigm is AND/OR tree, which we denote as Uncertainty AND/OR tree.

A. Idea of uncertainty paradigm search

A search tree under the uncertainty paradigm is a plain AND/OR tree. Likewise, search under the uncertainty paradigm is plain AND/OR tree search, which we denote as Uncertainty Paradigm Search (UPS).

Since UPS is AND/OR tree search, the result of the search is one of 'proved', 'disproved', or 'unsolved'. When a problem is 'proved', it turns out to be solved deterministically. When the problem is 'disproved', it is truly disproved or it is unsolvable deterministically. In this case, there may be another optimal solution using probabilistic search. If the problem is 'unsolved', there occurs some limitation for searching, for instance, lack of memory resource.

Theoretically, UPS is applicable to deterministically solving any puzzles or mating problems of incomplete-information games. Although there is a severe restriction of practical application in many domains, the advantage of UPS is that we can exploit other search algorithms along with UPS. For AND/OR tree search, several efficient algorithms are known, e.g., pn-search[2], [3], PDS[31], [32] and df-pn[33]. UPS can be implemented using such algorithms specialized for UPS as well as simple depth-first search with/without iterative deepening.

There is no essential difference whether a game has chance or not when solving it deterministically, since a game of chance is recognized as a game of no chance by giving one or more chances to players who make chance metamoves. In any case, the solver has to solve all metapositions that occur after chance metamoves. The same rule applies to the opponent.

B. The domain of applicability

Under the uncertainty paradigm, there are two kinds of sources for the complexity of searching. One is the explosion of the number of nodes, and another is the explosion of uncertainty, i.e., the number of positions in metapositions. The number of nodes depends on the branching factor at the solver's turn and the AND-splitting of metaposition. Explosion of uncertainty is critically dependent on the existence of the useful observations and fairly less possible moves at the opponent's turn, regardless of whether or not the metaposition explodes combinatorially in UPS. If $n_p^{child}/n_p \gg n_s^{child}$ for most of metamoves, then uncertainty clearly explodes soon. UPS is practically inapplicable to domains in which such combinatorial explosion occurs. However, selective search algorithms may give some help in such case, which we describe later.

Thus the applicable domains at present are:

single-agent problems with uncertainty
adversary-agent problems with uncertainty

UPS is applicable for most single-agent problems with uncertainty. For single-agent problems, this paradigm is natural in a way. Because a metamove of the solver is definite in most single-agent problems, it can be recognized as a plain move in such problems. Therefore, there is less difference between the uncertain paradigm and the conventional approach.

On the other hand, the uncertain paradigm performs well for adversary-agent problems, e.g., for mating problems of Kriegspiel-like games. Without using this paradigm the searching code would be much complicated and could hardly use efficient search algorithms such as PDS. However, when applying UPS to problems with much uncertainty, the uncertain paradigm has a severe restriction because of the combinatorial explosion. For instance, UPS cannot be applied to playing the actual Kriegspiel game by any means.

Note that we cannot show at present an UPS application example for multi-player games to be solved deterministically under the uncertainty paradigm. It can be applicable at least for some limited endgame in card games such as Bridge or Poker.

C. An example of uncertainty paradigm search

We explain the essence of the uncertainty paradigm, showing an example in the domain of Tsuitate-Tsume-Shogi (TTS). We then compare it with the case of solving by the conventional approach. The rules of TTS are given in Section IV.

Let us shown, in Figure 1, a problem of TTS and the essential part of its solution tree. This figure is depicted under the uncertainty paradigm. Note that a node is a metaposition. A metaposition that has more than one position is encircled, while a metaposition that has only one position is not encircled. The number at the lower right indicates the count of possible positions.

The node at the upper left represents the initial position of a given problem. At the position, the solver plays a check 1. Silver(銀) 2d-1e. After which there are four responses (a1 to a4) of the opponent.

(a1) If the opponent plays 1. ... King(玉) 1d-1e which captures Silver on 1e, the solver comes to know the response since Silver on 1e disappeared from the board. The information enables the solver to lead the position to a checkmate.

(a2-a4) If no capture happens, there are three possibilities, i.e., these (encircled) three positions are shown at the second ply in the figure. Note that the metaposition is diffused to one with three positions.

Then the solver plays a dropping move 2. Rook(飛) 2c'. If it is a foul move by dropping a piece at non-empty square (noticed by an umpire), the solver is able to know where the King is sitting, i.e., 2c. The information enables the solver to lead the position to a checkmate. Otherwise, the metaposition is reduced to one with two positions, where there are nine possible responses (b1 to b9) of the opponent.

(b1) If the opponent plays a move of King while capturing Rook on 2c, i.e., 2. ... King x1c-2c, then the solver is able to distinguish the move and the current position. This information enables the solver to lead the position to a checkmate.

(b2) If the opponent plays a move of King while capturing Silver on 1e, i.e., 2. ... King x2e-1e, then the solver is able to distinguish the move and the current position. The information also enables the solver to lead the position to a checkmate.

(b3-b9) If no capture happens after the response of the opponent, there are seven possibilities, i.e., the metaposition has seven positions. Of the seven possibilities, six are dropping moves at square 2d to interpose the check by Rook on 2c. In the figure, the interposing moves are represented by dropping a Pawn, i.e,, 2. ... Pawn 2d'. Another possibility is a move 2. ... King 1c-1b.

Now the solver is to move at the positions (encircled) which we show at the 5th ply from the top in the figure. There is no common check at these positions in this metaposition. Howver, 3. Rook 3d-1d is a metamove while utilizing a foul move, If it is a foul move, the solver obtain the exact information of the present position. This enables him to lead to a checkmate. Otherwise, the solver comes to know that the metaposition has only one position. After which, there are six interposing possible moves and one capturing move.

(c1-c6) If there is no capture, the solver comes to confirm that the response of the opponent was an interposing move such as 3. ... Pawn 1c'. This information is enough for the solver to lead the position to a checkmate.

(c7) Otherwise, the response of the opponent is a move of King while taking Rook on 2c, i.e, 3. ... King x1b-2c. At which, the metaposition has only one position, then solver can lead the position to a checkmate, by 4. Rook 1d-2d King 2c-1b 5. Rook 2d-2b+ (promoting) mate.

After all, the principal variation from the viewpoint of the longest step has nine plies and no foul whereas the allowed foul should be one for solving. The principal variation of the solution is:

1. 銀1e (−) 2.飛*2c (−) 3.飛1d 玉x2c 4. 飛2d (−) 5.飛2b+ mate.

'(−)' indicates a metamove after which no capturing has occurred. Incidentally, our program solves this problem in a few seconds.

In order to distinguish the essential difference between the conventional approach and the proposed uncertainty paradigm, we show the structures of the solution tree in both cases.

Let us show, in Figure 2, an illustration of the solution tree by the conventional approach. A node of game tree is a position, and a move between a parent position and a child position is indicated by a line. Thus the game tree has more nodes than the tree under the uncertainty paradigm. The information sets are used instead of metapositions. In Figure 2, an information set with more than one position is encircled.

In the figure, an OR node represents a position where the solver is to move, while an AND node is a position where the opponent is to move. Then, OR nodes and AND nodes appear by turns. These features conform to the common adversary-agent problems. However, only the structure of this tree does not have the sufficient information about the

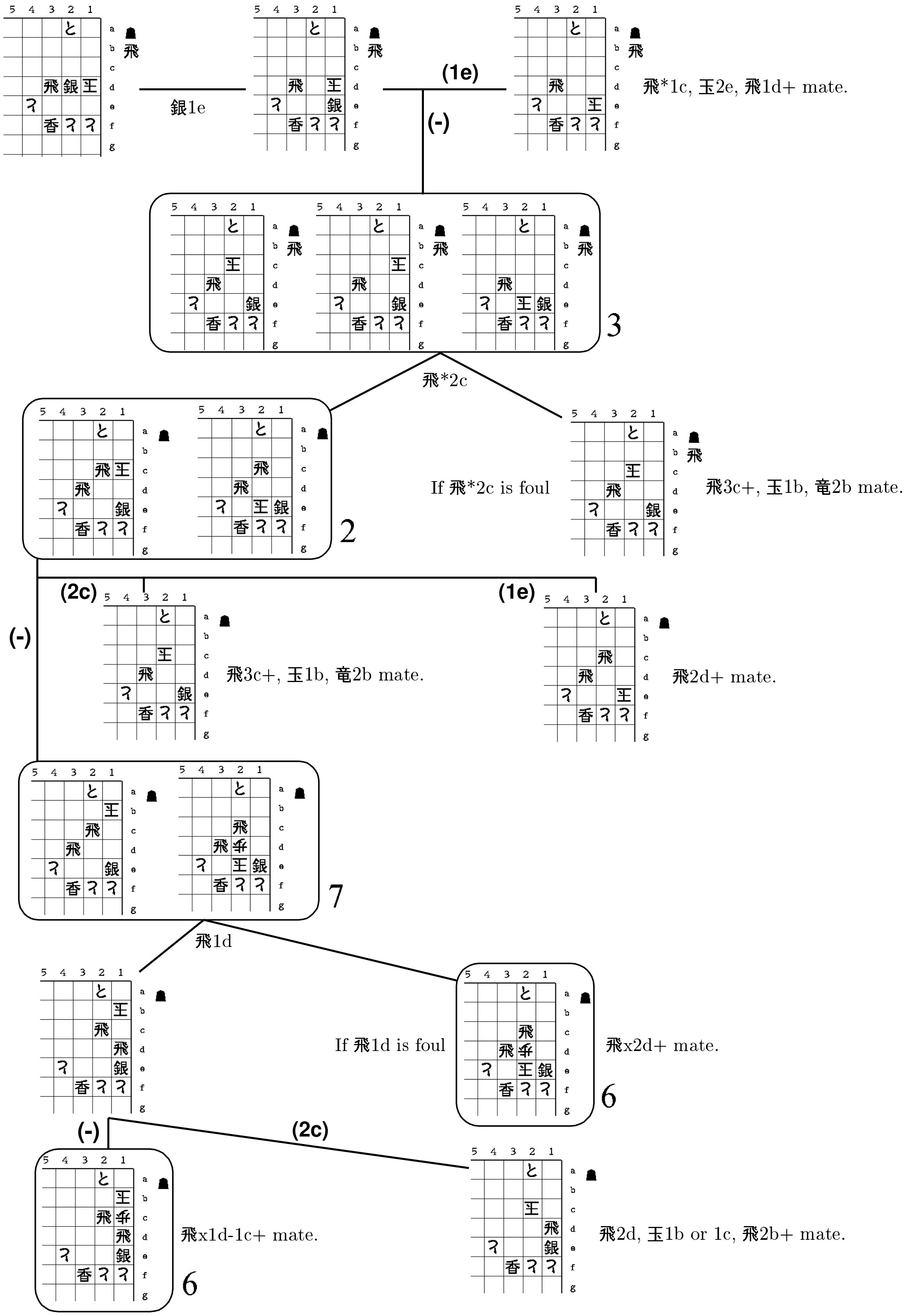

Fig. 1. Illustration of the essential part of solution tree for a sample TTS problem.

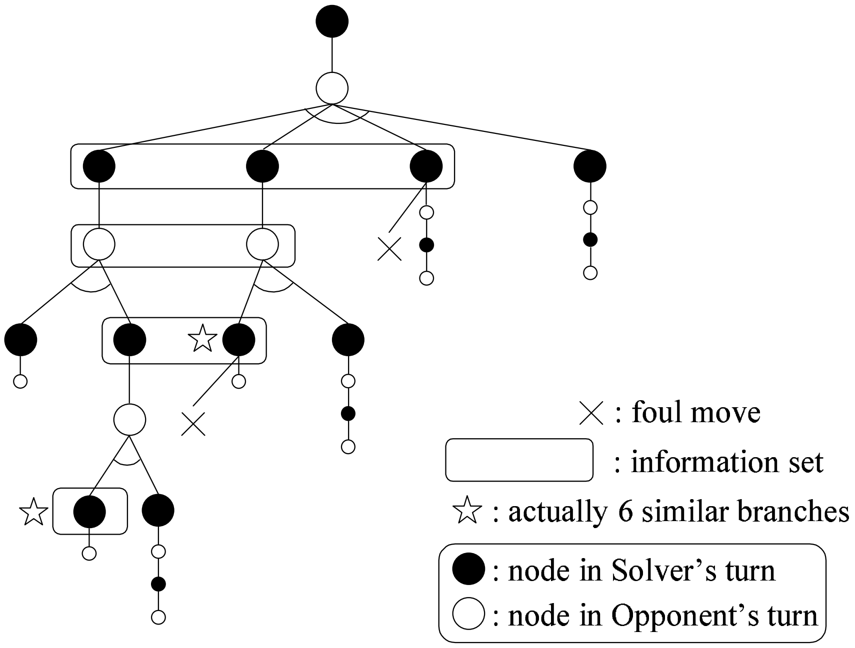

Fig. 2. Illustration of the solution tree with information sets (by the conventional approach).

problem. In addition to the information of the structure of the plain game tree, the information about the information sets should be stored. Moreover, the structure of the information sets is somewhat apart from the tree structure. For instance, two positions that are not siblings can be in the same information set. Hence, a game tree cannot be resolved into a plain AND/OR tree in the conventional approach.

Let us show, in Figure 3, an illustration of the solution tree under the uncertainty paradigm. A node in the figure represents a metaposition, and a metamove between a parent metaposition and a child metaposition is indicated by a line.

Since there is no need of extra information for solving a problem, the problem is resolved into the search of a plain AND/OR tree under the uncertainty paradigm. Nodes where the solver is to move are basically OR nodes, but OR branches often split into some AND branches. For instance, there are twelve possible moves, i.e., twelve OR branches at the initial position. Therefore, this AND/OR tree is more generic than that of the common adversary-agent problems. The difference from the common adversary-agent problems is that OR nodes and AND nodes do not appear by turns in this AND/OR tree.

IV. Applying Uncertainty Paradigm Search

Application examples of uncertainty paradigm search are shown in two domains: Kriegspiel-like game and Counterfeit Coin Problems.

A. Kriegspiel-like game

As an application of the uncertainty paradigm search to adversary-agent problems with uncertainty, we have chosen the domain of Tsuitate Tsume Shogi (TTS), endgame problems of a Japanese variant of Kriegspiel-like games.

Let us give a short summary of the TTS rules. The rules of TTS seem almost similar with Kriegspiel[37]. TTS requires two players, an umpire, two Shogi (Japanese chess) boards with a set of pieces, and a screen. Both players are unable to see the other side because of the screen. The principle of the game is that each player moves normally but is not told the opponent's moves which he attempts to discover through judicious play.

We should notice that a move of TTS includes not only *capture* but also *drop*. After one plays a move on his own board, the umpire approves the move to provide information to both players as required by the rules. Most important information is given by the umpire saying *check*, *illegal*, and *Black/White has played*. Since the umpire cannot tell other information aloud, he performs capturing on the board involved instead of the players.

Each player is allowed to make illegal moves within a certain number of times, typically eight times during a game. If one has made illegal moves more than the allowed number of times, he has to lose the game on fouls. A game also ends up when the king of any player is in mate.

A.1 Tsuitate Tsume Shogi (TTS)

Let us distinguish two players in TTS: the attacker and the defender, which corresponds to the solver and the opponent in UPS. The goal of solving a TTS problem is to mate the king of the defender after the sequence of check and its response. In TTS, the attacker is unable to obtain the definite information on the defender's responses, while the defender has the complete information on both sides. This corresponds to deterministic solving under the uncertainty paradigm. The attacker must lead to a checkmate whatever moves the defender may make.

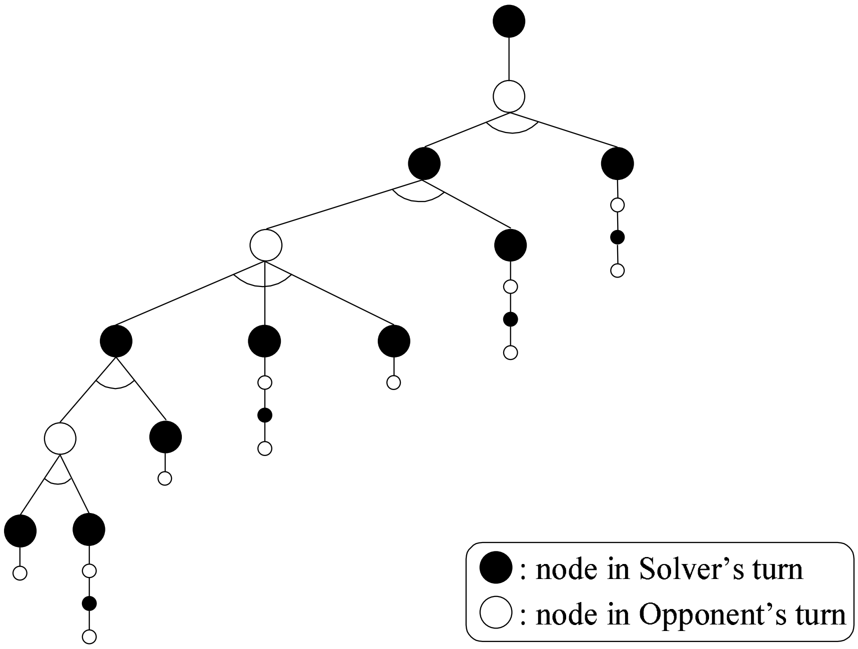

Fig. 3. Illustration of the solution tree under uncertainty paradigm proposed.

The attacker will guess the position of the defender's pieces as the mating search progresses by trying moves that can be either legal or illegal with respect to the whole position. In this sense, to make some (but limited) illegal moves is useful to gain insight into the position.

It is meaningless to try an explicit nonsense move, because it is illegal at all positions in a metaposition and the *solver* cannot get any information by trying it. No splitting of the metaposition occurs and the situation is absolutely the same before trying that move except increasing the count of fouls.

We introduce a notion so-called *foul move* as a subset of illegal moves. The foul move is a move that seems to be legal on the board of the solver, but illegal on the board with pieces of both sides.

A.2 Solving TTS problems

TTS has uncertainty, so the information sets of TTS often consist of many positions. Consequently, it is necessary to collect any clues of the current position for solving just the same as the problem of Kriegspiel. These are the common features of incomplete-information games.

In TTS, a metamove of the attacker is one of the definite moves, while a metamove of the defender is the uncertain hybrid of moves.

At a certain metaposition where the attacker is to move, possible moves are the moves that are either checks or foul moves at each position in the metaposition. One of the moves is a metamove to the metaposition. However, it is hopeless to make a metamove that is illegal at all positions in a metaposition, since the attacker is unable to obtain any information. No splitting of the metaposition occurs and the situation is completely the same before making that metamove except increasing the count of fouls. Therefore, the attacker should not play such a metamove that is a foul move at all positions in a metaposition.

At a certain metaposition where the defender is to move, possible moves are all the legal moves of all positions in the metaposition. The metamove is the hybrid of all possible moves.

The attacker will guess both the defender's moves and the current position through judicious play. Thus, a metaposition is split into some metapositions with less uncertainty. Eventually, TTS problem is resolved into the search of a generic AND/OR tree under the uncertainty paradigm.

A.3 Implementations and Experiments

We have implemented a computer program that can solve TTS problems[41] under the uncertainty paradigm. In our implementation for solving TTS problems, a metaposition has been represented as an array of possible positions in the metaposition.

In a metaposition, after a certain metamove, there happens to be some same positions. In this case, the redundant positions are discarded and the position count is reduced. These reductions usually speed up solving, but in some problems it is faster not to perform the reduction of positions. So there is an option switch for this in our program.

UPS has been implemented as a simple depth-first search for AND/OR tree using iterative deepening. We presently use a quite inefficient iterative deepening. Iteration is doubly nested to find a solution with the shortest steps (depths) and the least fouls. The search depth is employed as the outer iteration while the allowed count of fouls as the inner iteration.

After deciding the steps and fouls, the search with the multiple iterative deepening at OR nodes is performed to determine the principal variation of the solution. The so-

lution is determined by the following way.
First find the solution(s) with the shortest steps. If there are some solutions, find the solution(s) with the least numbers of fouls. If there are still some solutions, find the solution(s) with the smallest set of pieces in the attacker's hand at the position in mate.
The metaposition which contains a part of the principal variation is selected at the OR node, while the metaposition which contains the worst variation is selected at the AND node or at AND-splitting.

Our program was coded in C++ (Visual C++ 6.0). Experiments were performed under the following environment: Gateway2000, G6/GP6 Series (TB298-0109) Pentium II 450MHz, RAM: 384MB Windows 98.

We used a test set that contains 39 TTS problems[21], and then we first tried to solve the first 15 problems. They vary from some easy problems with 7 or 9 steps to some hard problems with 19 steps (3 permitted fouls).

A.4 Results and Discussions

Results of the experiments are shown in Table I. The program solved all problems. The number of metapositions and positions, denoted as M and P in Table I, is counted after defining the solvable search depth and fouls.

Table I show us that the average uncertainty index which is indicated by the average number of positions in a metaposition, roughly ranges from 1.4 to 6. The maximum uncertainty index that is indicated by the maximum number of positions in a metaposition varied widely from 9 to 1112.

The effective branching factors were calculated supposing the worst cases and turned out to be roughly ranged from 1.9 to 2.7. For references the effective branching factors based on the counts of positions are also shown. They are a little larger than those of the former ones are. Assuming the best cut-off cases, these values would be roughly square as large, ranged from 3.6 to 7.3. By allowing the attacker to try foul moves, the number of possible moves is not so small. Consequently, a quantity of computing is also not so small. There seems to have the tendency that the branching factor is larger for hard problems with relatively long steps.

In our current implementation, the most of solving time is the part defining the solvable depths and fouls by the doubly nested iterative deepening. By using some efficient algorithm such as PDS, the total numbers of generated metapositions and positions (denoted as TM and TP in Table I) would nearly come down to the total number of metapositions and positions (denoted as M and P in Table I) by a factor of one. Consequently, the total solving time, denoted as TT in Table I, would be reduced to the values of the time for defining the principal variation, denoted as TB in Table I.

B. Counterfeit Coin Problems

As a single-agent problem with uncertainty, we have chosen the domain of *Counterfeit Coin Problems*.
The problem is described as follows. There are several coins that are not distinguishable from the outer aspects, but one of them may be counterfeit. The weight of the counterfeit coin is different from that of the genuine coin, however, unfortunately it is unknown whether the counterfeit is lighter or heavier. There is a balance that can only show that the left side is heavier than the right side, or even, or the right is heavier than the left. How can you distinguish the counterfeit coin and determine whether it is lighter or heavier using the balance *via.* the least weighing?

There are variations of the rules according to whether it is possible that all coins are genuine, and according to whether there exists an absolutely genuine coin other than the suspicious coins.

This problem is well known and there exists the perfect heuristics for solving by solely using mathematical and information-theoretical consideration. There are a lot of studies on this problem, e.g., in [5][23][30][47], although most of those studies handled the problem mathematically or information-theoretically but no data of the brute-force search is shown.

Here we tackle this problem without any heuristics or with least heuristics while applying Uncertainty Paradigm Search (UPS).

There are two classes of solving algorithms considered: sequential algorithms and predetermined algorithms[1], [7]. Sequential algorithms allow the weighing performed on the next step in accordance to the previous results of weighing, while all the weighing have to be predefined before starting in predetermined algorithms.
In addition, there are two types of the purposes of problems: the worst-case problems and the average-case problems. The worst-case problem needs to minimize the maximum length of the solution tree by considering all possibilities, while the average-case problem minimizes the average length of solution given a certain probability distribution. We focus on the sequential solving algorithm for the worst-case problem, which fits well into the uncertain paradigm.

B.1 Solving Counterfeit Coin Problems

The initial state of a given problem is uncertain, so this can be recognized as a metaposition. The weighing by the balance is recognized as a metamove of the solver and this is actually some definite move. The result of weighing is the observation. Three cases are possible: the balance is even, the right side is down, or the left side is down.

The purpose of the problem is to find a solution tree that has the least height. All metamoves are definite in this problem. Hence, as described earlier, a metaposition does not diffuse at all but splits by observations into at most three metapositions.

At first, we have solved problems without using any heuristics but performing the brute-force search of a game tree with full width. The number of possible moves increases rapidly as the number of coins is getting more. Consequently, the search space increases explosively. UPS has been implemented as a simple depth-first search for AND/OR tree using iterative deepening. The search depth is started from 2 and increased by one at each time of iteration.

No.	ST	F	TM	TP	M	P	MP	AP	BM	BP	TB	TT
1	7	0	5668	14844	608	1244	46	2.05	2.50	2.77	0.15	1.67
2	13	1	236503	956259	28543	73494	138	2.57	2.20	2.37	10.60	118.5
3	9	2	44978	280136	6263	26513	94	4.23	2.64	3.10	3.70	32.11
4	9	0	6205	9683	1241	1761	9	1.42	2.21	2.29	0.20	1.09
5	11	0	18534	33841	2901	4578	30	1.58	2.06	2.15	0.54	3.91
6	11	0	62099	177146	11831	36773	59	3.11	2.35	2.60	4.19	20.09
7	11	1	59834	140533	6088	10413	46	1.71	2.21	2.32	1.37	16.02
8	13	3	1769090	13909600	166303	880752	214	5.30	2.52	2.87	219.3	2151
9	11	1	217325	409273	60581	107893	59	1.78	2.72	2.87	14.64	47.96
10	13	1	40910	302404	24389	126538	592	5.19	2.18	2.47	19.91	37.41
11	15	1	989726	7406880	393206	2274680	756	5.78	2.36	2.65	563.6	1184
12	19	2	1387210	7289530	292411	1852760	1112	6.34	1.94	2.14	411.8	982.50
13	19	3	28345500	171844000	12651700	65214600	751	5.15	2.36	2.58	139740	199839
14	19	2	20198300	82664900	10647300	30074200	1090	2.82	2.34	2.48	17798	35742
15	11	3	220245	968951	50570	185196	423	3.66	2.68	3.01	25.02	113.5

ST represents the number of steps of the solution. Similarly, we use abbreviation as follows. F: count of fouls appeared in the principal variation, TM: total number of metapositions generated, TP: total number of positions generated, M: total number of metapositions generated, P: total number of positions generated, MP: the maximum number of positions in a metaposition, AP: the average number of positions in a metaposition, BM: effective branching factor based on metapositions, BP: effective branching factor based on positions, TB: time for determining the principal variation, and TT: total time to solve (seconds)

TABLE I
Results of solving TTS problems.

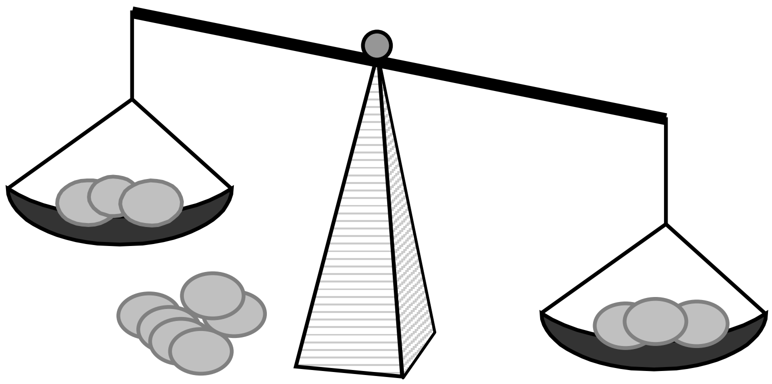

Fig. 4. Typical Counterfeit Coin Problem.

Secondly, we have incorporated the minimum knowledge from information theory into the brute-force search. Since the number of splitting a metaposition is at most three, one weighing can distinguish at most three cases. Therefore, we have cut irrelevant metapositions using the following criterion:
If the number of possible positions in a metaposition is greater than the power of 3 to the remaining search depth, the metaposition is unsolvable.
In addition, expanding this idea, for every metamove of a metaposition, the maximum number of possible positions is obtained and checked. If the maximum number is greater than the power of 3 to the remaining search depth, the metamove is not relevant and cut off. This is an information-theoretical and correct cut-off. A metaposition has been represented directly as itself in the implementation of this problem. Experiments were performed in the same environment mentioned in Section IV-A.3.

The search depth of iterative deepening is started from the integer not greater than the logarithms to the base 3 of the number of possible positions at the initial state.

B.2 Results and Discussions

We examine the two types of problems: (1) to find one counterfeit among n suspicious coins without a genuine coin, and (2) to do so with a genuine coin.
In both types, we assume that all coins might be genuine. The results by the brute-force search without any knowledge are shown in Table II and Table III respectively.

When using no heuristics, the number of nodes in the search explodes quickly as the number of coins increases. A program based on this concept is unable to solve the problem in practical time when the number of coins N exceeds 30. In the case of 30 coins, the maximum number of metamoves (denoted as MM in Table III) increased up to

N	D	T	TM	M	LM	AS	AM	MM	B
3	2	0.00079	11	11	7	1.25	2.00	3	2.65
4	3	0.0018	22	14	9	0.93	2.80	5	2.08
5	3	0.0029	36	21	13	1.12	3.25	11	2.35
6	3	0.0043	41	30	20	1.47	4.70	9	2.71
7	3	0.011	106	87	67	1.75	7.90	23	4.06
8	3	0.016	118	102	79	2.00	10.39	35	4.29
9	3	0.047	396	371	320	2.32	13.65	48	6.84
10	3	0.062	438	418	362	2.39	16.48	68	7.13
11	3	0.18	1247	1216	1098	2.54	22.54	86	10.32
12	3	0.21	1352	1326	1199	2.56	25.01	112	10.62
13	4	1.4	8107	1855	1631	2.68	21.99	146	6.35
14	4	2.4	12714	2910	2605	2.71	27.07	184	7.14
15	4	3.2	14801	3886	3507	2.73	31.86	232	7.70
16	4	13.7	72050	54964	51607	2.74	52.36	281	15.07
17	4	17.9	85828	67734	63955	2.77	62.28	341	15.90
18	4	45.9	224245	196220	185703	2.77	65.10	418	20.76
19	4	64.6	270695	242187	230314	2.79	76.84	498	21.91
20	4	127.2	528517	484439	460891	2.79	78.63	598	26.06
21	4	170.9	640575	597361	570651	2.81	91.64	696	27.48
22	4	278.3	1069510	1003009	958646	2.81	95.30	816	31.29
23	4	359.9	1295227	1231988	1181509	2.83	109.47	958	32.97
24	4	563.6	2032985	1935732	1856244	2.83	111.81	1108	36.91
25	4	759.7	2437945	2347826	2257910	2.84	127.01	1283	38.76
26	4	1200	3720302	3582083	3444058	2.84	128.46	1461	43.08
27	4	1496	4394876	4269579	4115229	2.85	144.68	1666	45.04
28	4	2302	6701694	6509570	6271178	2.85	143.69	1907	50.04
29	4	2768	7767040	7596216	7333808	2.86	160.78	2154	52.04
30	4	4399	11978736	11717248	11306090	2.86	157.54	2441	57.99

N: number of suspicious coins, D: solution depth, T: solving time in seconds, TM: total count of generated metapositions, M: count of metapositions in success iteration, LM: count of terminal metapositions, AS: average number of splits, AM: average number of metamoves, MM: maximum number of metamoves, B: effective branching factor

TABLE II

Results of the performance by brute-force search

N	D	T	TM	M	LM	AS	AM	MM	B
3	2	0.0012	14	14	9	1.47	3.00	4	3.00
4	2	0.0017	19	19	13	1.68	4.67	8	3.61
5	3	0.0065	59	26	18	1.59	5.13	16	2.62
6	3	0.014	126	88	68	2.21	6.75	23	4.08
7	3	0.029	218	175	140	2.39	9.37	35	5.19
8	3	0.049	353	302	249	2.45	11.62	48	6.29
9	3	0.085	539	482	407	2.55	14.71	68	7.41
10	3	0.14	897	832	720	2.59	17.24	86	8.96
11	3	0.23	1247	1176	1031	2.63	21.51	112	10.10
12	3	0.39	2148	2067	1851	2.66	25.12	146	12.28
13	3	0.59	2803	2716	2458	2.69	31.64	184	13.50
14	4	6.2	30134	3882	3505	2.73	32.02	232	7.69
15	4	17.9	89699	55718	52308	2.75	52.64	281	15.12
16	4	37.3	187937	142694	134679	2.76	59.13	341	19.16
17	4	73.3	353670	296829	281059	2.77	65.93	418	23.02
18	4	131.4	573353	499612	474457	2.78	71.97	498	26.25
19	4	217.9	913848	823325	783658	2.79	79.32	598	29.75
20	4	340.1	1324856	1209804	1154237	2.80	86.48	696	32.78
21	4	507.0	1938164	1799646	1720512	2.81	95.45	816	36.22
22	4	722.0	2642740	2470165	2366616	2.82	104.42	958	39.22
23	4	1026	3672140	3467752	3328787	2.83	115.54	1108	42.71
24	4	1494	4827063	4576225	4401203	2.84	126.31	1283	45.80
25	4	2172	6530354	6237159	6007829	2.85	138.37	1461	49.51
26	4	2906	8362854	8008139	7727429	2.86	151.46	1666	52.72
27	4	4067	11134716	10724633	10360607	2.86	163.73	1907	56.73
28	4	5250	14006040	13515535	13078076	2.87	178.95	2154	60.14
29	4	7384	18490245	17928409	17361181	2.87	190.23	2441	64.55
30	4	9666	22970892	22306158	21632516	2.88	207.24	2727	68.20

TABLE III

Results of the performance by brute-force search (with one genuine coin)

2441 or 2727, the effective branching factor (B) is roughly 50 to 70, where the number of nodes (M) is greater than 10^7. Therefore, there are a lot of failure branches in the search tree. However, it is very important to solve problems without using any knowledge in the viewpoint of artificial intelligence. Therefore, this problem may be a good subject for machine learning as a future work.

Secondly, the results using the minimum knowledge from information theory are shown in Table IV and Table V respectively. When using the minimum knowledge from information theory, all metamoves that are recognized as relevant are the right metamoves to be lead to a successful terminal node. Therefore, there is no failure branch in the search. The heuristics used is turned out to be crucial for this problem. However, as the number of coins is becoming larger, the number of candidate metamoves to be checked (MC) is becoming extremely larger. Therefore, for problems that have more than a thousand coins, it takes much time to solve them. Of course, as mentioned before, since there is a perfect heuristics for solving this problem, we can make a program to solve a problem immediately even though the number of coins is greater than a thousand.

N	D	T	M	LM	AS	MC
3	2	0.00093	11	7	2.500	1
4	3	0.0012	14	9	2.600	2
12	3	0.0027	38	25	2.846	27
13	4	0.0030	42	27	2.733	11
39	4	0.0099	119	79	2.950	1070
40	5	0.0093	124	81	2.860	715
120	5	0.149	362	241	2.983	71343
121	6	0.133	367	243	2.952	61997
363	6	17.27	1091	727	2.995	5426531
364	7	16.36	1096	729	2.984	5176768
1092	7	3289.0	3278	2185	2.998	430963047
1093	8	3226.0	3283	2187	2.995	424227731

N: number of suspicious coins, D: solution depth, T: solving time in seconds, M: count of metapositions, LM: count of terminal metapositions, AS: average number of splits, MC: maximum count of combinations searching a metamove

TABLE IV

Results of the performance by search with the minimum knowledge

N	D	T	M	LM	AS	MC
4	2	0.0010	13	9	3	4
5	3	0.0012	16	11	3	4
13	3	0.0028	40	27	3	32
14	4	0.0031	43	29	3	32
40	4	0.0099	121	81	3	1124
41	5	0.0096	124	83	3	1124
121	5	0.149	364	243	3	71608
122	6	0.150	367	245	3	71608
364	6	17.29	1093	729	3	5434304
365	7	17.31	1096	731	3	5434304
1093	7	3289.6	3280	2187	3	431114692
1094	8	3290.9	3283	2189	3	431114692

TABLE V

Results of the performance by search with the minimum knowledge (with one genuine coin)

As shown in the experimental results of counterfeit coin problems, we confirm the effectiveness of the proposed uncertainty paradigm in the domain of single-agent problems with uncertainty.

V. Concluding Remarks

We proposed a novel paradigm for solving problems with uncertainty, which we call Uncertainty Paradigm, and Uncertainty Paradigm Search (UPS). Under this paradigm a game tree of any problem with uncertainty is resolved into a plain AND/OR tree in the viewpoint of deterministic solving. Therefore, search algorithms of AND/OR tree can simply be adapted for such problems as well as using efficient algorithms for AND/OR tree search. Successful applications of this paradigm to adversary-agent problems, Tsuitate-Tsume-Shogi, and the single-agent problem, Counterfeit Coin Problem, were shown.

However, UPS has severe restriction of uncertainty. To reduce the combinatorial explosion of uncertainty, the co-use of this paradigm with the Partition Search[13] and the descriptive representation of a metaposition are possible for enhancement.

When we consider search trees under uncertainty, uncertainty paradigm with metaposition and metamove is the only unique way to include multiple positions in one node and multiple moves in one arc. In this sense, it is strictly different from search trees of conventional theories, such as decision theory, dynamic programming and game theory. Even though UPS is limited in use, it calculates the optimal solution forwardly. When we deal with dynamic decisions, any kind of conventional trees with single positions face with the combinatorial explosions of branching. This is why dynamic programming is only calculated backward. UPS is the first method of forward calculation that avoids the combinatorial explosions of branching using meta-concepts. This method may provide a true solution for dynamic game without essentially invalid assumptions used in conventional dynamic games with ado-hoc assumption.

In the current development, UPS reduces a plain and/or tree without probability weights because we evaluate all the possible options. In future, we should include the probability coefficients as used in decision theory and dynamic programming.

A. Possible Enhancements

There are several enhancements for UPS. In this paper, a simple depth-first iterative deepening with full-width search is performed in UPS. Such a simple iterative deepening is not practical for a problem with large search space.

First, we did not use any transposition table while searching. The efficiency could be much higher to use transposition tables. The use of the transposition table is a well-known and effective method in searching a game tree. The Zobrist method, the fast method of coding positions using the pseudo-random numbers is known for board games[50]. However, under the uncertainty paradigm, it is not directly applicable for coding a metaposition. We consider the use of the CRC (or checksum) of position codes as a code of a metaposition.

Secondly, our program was able to successfully solve the test set of Tsuitate-Tsume-Shogi problems, except several hard problems that have relatively long steps or considerably large branching factors. To solve such problems in a short time, we should examine the other search algorithms of the best-first manner such as pn-search, PDS and df-pn, where each algorithm must be modified for UPS. We compared these algorithms in the domain of adversary-agent problems with complete information while using 6x6 Othello and Tsume-Shogi. We then confirmed that PDS outperformed the other algorithms in terms of solving speed and ability[40]. Since UPS is an AND/OR tree search as well as the search of adversary-agent problems with complete information, these algorithms represented by PDS would work efficiently in UPS. However, some modifications or specialization of these algorithms will be required in UPS because there are additional factors derived from uncertainty.

Thirdly, to reduce the combinatorial explosion, some enhancements are possible for UPS. For reducing nodes that need to be visited or to be stored in game-tree search, the search technique called Partition Search is well known. This technique reduces the nodes drastically by recognizing the similar positions that would come to be the same result as the single node by dependency analysis. This method combines several positions into one super node, so in this meaning there is a common point with the concept of metapositions. However, a node of a search tree is a position after all and the structure of the search tree is essentially same as that in the simple search except the number of nodes. On the other hand, a node of a search tree is a metaposition and the tree structure is quite different in UPS. So the partition search is based on the other concept that is different from the uncertain paradigm. Nevertheless, it is possible to use the partition search under the uncertain paradigm. It may be useful to reduce the number of positions in a metaposition. In this case dependency analysis would be performed for positions in a metaposition.

There are other modifications for the implementation of metapositions. The metaposition can be implemented as an array of possible positions, or as a metaposition itself according to the problem concerned. However, these implementations are not practical for problems with much uncertainty. For example, instead of storing each position, we can store one start position and the move sequence to each position. This modification would considerably reduce the restriction of memory space. Moreover, to deal with problems with heavy uncertainty, it could be effective to use a descriptive representation of metapositions.

Acknowledgements

We would like to thank Jonathan Schaeffer for his advice on the partition search and the implementation of the metaposition. This work was in part supported by the grant-in-aid of the Ministry of Culture in Japan to H.I.(grant no.11695046).

References

[1] M. Aigner, *Combinatorial Search*, John Wiley & Sons Ltd, and B.G. Teubner, Stuttgart, 1988.

[2] L.V. Allis, M.van der Meulen and H.J.van den Herik, Proof-number search, *Artificial Intelligence* 66(1994) 91-124.

[3] L.V. Allis, *Searching for Solutions in Games and Artificial Intelligence*, Ph.D. thesis, Department of Computer Science, University of Limburg, The Netherlands, 1994.

[4] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[5] R. Bellman, B. Gluss, On Various Versions of the Defective Coin Problem, *Information and Control* 4(1961) 118–131.

[6] R. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, 1961.

[7] A.D. Bonis, L. Gargano, U. Vaccaro, Optimal Detection of a Counterfeit Coin with Multi-Arms Balances, *Discrete Applied Mathematics* 61(1995) 121–131.

[8] C. W. Clark, *Mathematical Bioeconomics: The Optimal Management of Renewable Resources*, 2nd ed, Wiley-Interscience, New York, 1991.

[9] M.H. DeGroot, *Optimal Statistical Decisions*, McGraw-Hill, New York, 1970.

[10] P.C. Fishburn, *Nonlinear Preference and Utility Theory*, Johns Hopkins University Press, Baltimore, Maryland, 1988.

[11] P.C. Fishburn, Foundations of decision analysis: Along the way, *Management Science* 35(1989) 387–405.
Pages,

[12] I. Frank, D. Basin, Search in games with incomplete information: a case study using Bridge card play, *Artificial Intelligence* 100(1998) 87-123.

[13] M.L. Ginsberg, Partition Search, in: *Proceedings the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996, pp.228–233.

[14] M.S. Greenberg and R. B. Ruback, *Social Psychology of The Criminal Justice System*, Brooks/Cole, Monterey, California, 1982.

[15] R. Grimbergen, A Survey of Tsume-Shogi Programs Using Variable-Depth Search, in: H.J. Herik, H. Iida (Eds.), *Proceedings First International Conference on Computers and Games (CG'98)*, Lecture Notes in Computer Science, 1558, Springer, Heidelberg, 1999, pp.300–317.

[16] J. C. Harsanyi, *Rational Behavior and Bargaining Equilibrium in Games and Social Situations*, Cambridge University Press, 1977.

[17] D.P. Heyman and M.J. Sobel, *Stochastic Models in Operations Research*, Volume I, *Stochastic processes and Operating Characteristics*, McGraw-Hill, New York, 1982.

[18] D.P. Heyman and M.J. Sobel, *Stochastic Models in Operations Research*, Volume II, *Stochastic Optimization*, McGraw-Hill, New York, 1984.

[19] R.M. Hogarth and M. W. Reder eds, *Rational Choice: The Contrast Between Economics and Psychology*, University of Chicago Press, 1986.

[20] A.I. Houston and J.M. McNamara, Singing to attract a mate-a stochastic dynamic game, *J. Thoer. Biol.* 129(1987) 57–68.

[21] T. Kato (Ed.), *Collection of Kapitein Documents No.1*, Explanations and Problems of Tsuitate-Tsume-Shogi (in Japanese), 1995.

[22] D. Koller, A. Pfeffer, Representations and Solutions for Game-Theoretic Problems, *Artificial Intelligence* 94(1997) 167–215.

[23] N. Linial, M. Tarsi, The Counterfeit Coin Problem Revisited, *SIAM J. Comput.* 11(1982) 409–415.

[24] R.D. Luce and H. Raiffa, *Games and Decisions*, Wiley, New York, 1957.

[25] M.J. Machina, Choice under uncertainty: Problems solved and unsolved, *Economic Perspectives* 1(1987) 121–154.

[26] M. Mangel, *Decision and Control in Uncertain Resource Systems*, Academic Press, New York, 1985.

[27] M. Mangel and C. W. Clark, *Dynamic Modeling in Behavioral Ecology*, Princeton University Press, 1988.

[28] J. Maynard Smith, The theory of games and the evolution of animal conflicts, *J. theor. Biol* 47(1974) 209–221.

[29] J. Maynard Smith, *Evolution and the Theory of Games*, Cambridge University Press, 1982.

[30] D.G. Mead, The Average Number of Weighings to Locate a Counterfeit Coin, *IEEE Trans. Inform. Theory* 25(1979) 616–617.

[31] A. Nagai, A new AND/OR Tree Search Algorithm Using Proof Number and Disproof Number, in: *Proceedings of Complex Games Lab Workshop*, ETL, Tsukuba, 1998, pp.40–45.

[32] A. Nagai, A New Depth-First-Search Algorithm for AND/OR Trees, M.Sc. thesis, Department of Information Science, University of Tokyo, Japan, 1999.

[33] A. Nagai, H. Imai, Application of df-pn+ to Othello Endgames, in: *Proceedings of the 5th Game Programming Workshop*, Hakone, Japan, 1999, pp.16–23.

[34] J.J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, Princeton University Press, Princeton, New Jersey, 1944.

[35] G. Owen, *Game Theory*, Academic Press, New York, 1995.

[36] D.A. Pierre, *Optimization Theory With Applications*, John Wiley, New York, 1969.

[37] D.B. Pritchard, *The Encyclopedia of Chess Variants*, Games & Puzzles Publications, Godalming, UK, 1994.

[38] H. Raiffa and R. Schlaifer, *Applied Statistical Decision Theory*, Harvard University Press, Cambridge, Massachusetts, 1961.

[39] H. Raiffa, *Decision Analysis*, Addison-Wesley, Reading, Massachusetts, 1968.

[40] M. Sakuta and H. Iida, An Empirical Comparison of Depth-First AND/OR Tree Search Algorithms on 6x6 Othello and Tsume-Shogi, in: H.J. Herik (Ed.), *Advances in Computer Chess 9*, in press.

[41] M. Sakuta and H. Iida, Solving Problems with Uncertainty: A case study using Tuitate-Tsume-Shogi, in: Proceedings of the 5th Game Programming Workshop, Hakone, Japan, 1999, pp.145–152.

[42] L.J. Savage, *The Foundations of Statistics*, Dover, New York, 1954.

[43] M. Shubik ed., *Game Theory and Related Approaches to Social Behavior*, Wiley, New York, 1964.

[44] M. Shubik, *Game theory in the Social Sciences: Concepts and Solutions*, The MIT Press, 1982.

[45] H.M. Wagner, *Principles of Operations Research*, 2nd ed., Prentice/Hall, London, 1975.

[46] D.S. Wilson, *The Natural Selection of Populations and Communities*, Benjamin/Cummings, Menlo Park, California, 1980.

[47] K. Winkelmann, An Improved Strategy for a Counterfeit Coin Problem, *IEEE Trans. Inform. Theory* 28(1981) 120–122.

[48] D. von Winterfeldt and W. Edwards, *Decision Analysis and Behavioral Research*, Cambridge University Press, 1986.

[49] J. Yoshimura and C. W. Clark, Individual adaptations in stochastic environments, *Evolutionary Ecology* 5(1991) 173–192.

[50] A.L. Zobrist, A New Hashing Method with Applications for Game Playing, *Technical Report #88*, Computer Science Department, The University of Wisconsin, Madison WI, USA, 1970. Reprinted in: *ICCA Journal* 13(2) (1990) 69–73.