Mathematical Problems of Computer Science 43, 62--75, 2015.

Structuring of Goals and Plans for Personalized Planning and Integrated Testing of Plans

Sedrak H. Grigoryan

Institute for Informatics and Automation Problems of NAS RA
e-mail: addressforsd@gmail.com

Abstract

We study competition problems defined in the class where Space of Solutions is a Reproducible Game Tree (RGT). Personalized Planning and Integrated Testing algorithms were developed for searching optimal strategies in RGT problems. Hereinafter we develop structures for plans and goals in PPIT, construct strategy searching algorithms by plans and demonstrate their adequacy for chess endgame examples.

Keywords: Strategy, Plan, Competition, Chess, Goal.

1. Introduction

1.1. In [2] the variety of problems was identified as a class where *Space* of possible *Solutions* can be ***specified*** by *Reproducible* combinatorial *Game Trees* (RGT) and ***unified*** algorithms and software were developed, ***RGT Solver***, for elaborating optimal strategies for any input specified problem of the class.

The RGT is a spacious class of problems with only a few following requirements to belong to:

- there are (a) interacting actors (players, competitors, etc.) performing (b) identified types of actions in the (c) specified moments of time and (d) specified types of situations
- there are identified benefits for each of the actors
- the situations the actors act in and transformed after the actions can be specified by certain rules, regularities.

Many security and competition problems belong to RGT class. Specifically, these are network Intrusion Protection (IP), Management in oligopoly competitions and Chess-like combinatorial problems, many other security problems such as Computer Terrorism Countermeasures, Disaster Forecast and Prevention, Information Security.

1.2. Unified RGT specification of problems makes possible to design a unified Solver for the problems of the class.

Solver of the RGT problems is a package [7] aimed to acquire strategic expert knowledge to become comparable with a human in solving hard combinatorial competing and combating problems. In fact, the following three tasks of expert knowledge acquisition can be identified in the process:

- construction of the package of programs *sufficient* to acquire the ***meanings*** of the units of vocabulary (UV) of problems
- construction of procedures for *regular* acquisition of the meanings of UV by the package
- provision of means *measuring the effectiveness* of solutions of RGT problems.

The limitations in designing effective package were formulated as follows:

- be able to store typical categories of communalized knowledge as well as the personalized one and depend on them in strategy formation
- be able to test approximate knowledge-based hypothesis on strategies in questioned situation by reliable means, for example, using game tree search techniques.

The second task of *acquisition of complex expert knowledge* was planned to solve in the following two stages:

- *proving the sufficiency,* i.e. proving that Solver, in principle, can acquire the meanings of expert knowledge of an intensive RGT problem, e.g., for the kernel RGT chess game
- *ensuring regularity,* i.e. to develop procedures for regular acquisition of RGT problems and meanings of UV of those problems.

1.3. Regular improvement of Solver by expert knowledge is studied for chess, where the problems of knowledge representation and consistent inclusion into the programs stay central since the pioneering work by Shannon in 1950.

Players indicate and communicate chess knowledge by units of vocabulary and are able to form corresponding contents. Whether it is possible to form equal contents by computers remains questionionable.

The approaches to regular inclusion of chess knowledge into strategy formation process are described in [5]. Then try to bring common handbook knowledge to cut the search in the game tree. The frontiers of those approaches can be revealed by understanding the role and proportion of the personalized chess expertise compared with the common, communicable one.

1.4. Studies of knowledge-based strategies in the Institute for Informatics and Automation Problems of the National Academy of Sciences of the Republic of Armenia have been started in 1961 and noticeable results were published in the Laboratories of “Mathematical Logic” and “Cognitive Algorithms and Models” led by I. Zaslavski [1] and E. Pogossian [2, 3, 6].

Designed and developed PPIT (Personalized planning and integrated testing) [2, 6] algorithms indicate the optimal strategy by effective usage of expert knowledge. The algorithms had been tested for a variety of problems, for chess, Reti and Nodarishvili etudes [6], for intrusion protection problems [3]. In the PPIT algorithms predefined set of knowledge was used which was strongly specific to the solving problem and did not provide a generic and regular way to define knowledge and reveal strategies from them. This approach reduced the abilities of algorithm execution, since it required writing a new program to solve each certain situation and each of them was useful only for the given situations, so the program developed for Reti etude could be used only to solve this etude.

In the RGT Solver strategy searching algorithms were not yet suggested to provide general solution while plans used in PPIT algorithms are only generic descriptions of strategies.

In the following we describe planning-based strategy searching algorithms within the frame of Solver package.

In the first section we consider structuring of plans and goals. We need these structures for strategy generation algorithms. In the second one the algorithm that searches for a strategy to accomplish the plan and in the last section an example demonstrating adequacy of structures and strategy searching algorithm are described.

2. Contributing to Personalized Planning and Integrated Testing (PPIT) Algorithms

2.1. State of the Art

2.1.1. The Basics of PPIT

For the strategy construction we use PPIT algorithm, which creates strategies using plans. Plans are certain general descriptions of strategies. For some positions in chess plans might be occupying the center or the corners of the board. Each plan represents a hierarchy of goals. Those are the goals which a player tries to achieve in current situation while playing by the plan. The essences of the plans are to select the goals which get the maximal profit.
The PPIT program was designed as a composition of the following basic units:
Reducing Hopeless Plans (RHP)
Choosing Plans with Max Utility (CPMU)
Generating Moves by a Plan (GMP)
Given a questioned position P1 and a store of plans, RHP recommends to CPMU a list L1 of plans promising by some not necessary proved reasons to be analyzed in P1. The core of the unit is knowledge in classification of chess positions allowing identifying the niche in the store of knowledge the most relevant for analysis the position. If the store of knowledge is rich and P1 is identified properly it can provide a ready-to-use portion of knowledge to direct further game playing process by GMP unit. Otherwise, RHP, realizing a reduced version of CPMU, identifies L1 and passes the control to CPMU.
CPMU recommends to GMP to continue to play by current plan if L1 coincides with list L0 of plans formed in the previous position P0 and changes in P1 are not essential enough to influence the utility of current plan.
If changes in P1 are essential, CPMU analyzes L1 completely to find a plan with max utility and to address it to GMP as a new current plan. Otherwise, CPMU forms a new complementary list L1/ L1*L0 from the plans of L1 have not been analyzed, yet, in L0, finds a plan with the best utility in that list and comparing it with the utility of the current plan recommends one of them with a higher utility.
To calculate utilities of the attribute, goal and plan type units of chess knowledge, we represent them as operators over the corresponding arguments as follows:

- ➢ for basic attributes the arguments are characteristics of the states of squares in the questioned positions, including data on captures of pieces, threats, occupations, etc.;
- ➢ for composed attributes, including concepts and goals, the arguments are subsets of values of basic attributes relevant to the analyzed positions;
- ➢ for plans the arguments are utilities of the goals associated with the realization of those plans.

Utilities of arguments of basic attributes are calculated by the trajectory-zones based technique (TZT) [4, 6] originally suggested to estimate utilities of captures only of the opponent pieces. For example, to choose capture with max utility TZT chains the moves to each piece of the opponent (trajectories) without accounting possible handicaps for real capturing then using all available knowledge “plays the zones” of the game tree induced by the trajectories followed by estimation of their values to choose the best.
The utility of units of knowledge the operators assemble from the utilities of the corresponding arguments in some predetermined ordering. Thus, each operator can provide by a request the arguments which are analyzed at the moment.

For example, realizing the current plan the shell can determine the goal in the agenda which in turn determines basic attributes to be considered followed by indication of the arguments of those attributes.
Utility estimation operators rely on the principle of integration of all diversity of units of knowledge the shell possesses at the moment. In fact, the operators represent a kind of expert knowledge with a variety of mechanisms and leverages to make them better. Along with dynamically changed parametric values of pieces they can include rules, positions with known values and strategies to realize them, other combinatorial structures. To estimate expected utilities the operators take into account the cost of resources necessary to get them.

2.1.2. What has been done

In the initial C++ realization the units of knowledge are realized as OO classes with specialized interfaces for each type of knowledge and one common for the shell itself.

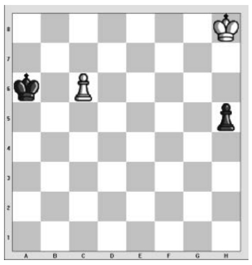
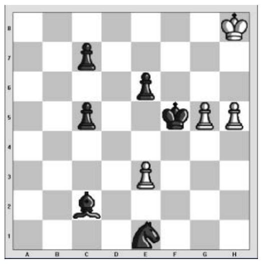
Fig 1. Reti and Nodareishvili etudes.

The Solver is experimented in solving Reti and Nodareishvili etudes (Fig 1.) required by Botvinnik[4, 5] intensive expert knowledge-based analysis not available to conventional chess programs.
Experiments with these etudes proved that the shells, in principle, can acquire the contents of units of vocabulary used by chess players and allow tuning them properly to solve expert knowledge intensive chess problems.
The initial implementation of the PPIT algorithm used knowledge units that were hardcoded as C++ language classes. The approach didn't allow adding expert knowledge in a regular way – there wasn't any regular method for formalization and representation of the expert knowledge.
To achieve a regularity of expert knowledge acquisition for RGT problems a graphical language similar to the UML, using which experts have possibility to formalize and insert meanings of the communicable knowledge into the Solver.
The constituents of the Interface have been designed for specifying both game attributes and rules. It was designed to acquire an expert knowledge in a form of patterns (abstracts). Abstracts are used to define classes as well as operations, thereby providing a considerable uniformity of the structure of the language [7, 10].

2.1.3. What We are Going to Do

We are developing algorithms and structures of strategy construction in the Solver package by putting the stress on GMP module of PPIT algorithms first. So for the current state of development we suppose that we already have plans defined in the Solver and we just need to execute the defined plan. Plans are being defined by experts.
In PPIT Plan is defined as a set of Goals. We will describe their definitions below.

As mentioned above the third module of PPIT GMP chooses the best move from a plan. We meet the following issues

1. Goals' and Plans' structures need to be generic and need to allow definition of the goals independent of the problems they relate to.
2. An algorithm needs to be developed to search for strategies using defined plans and goals. The algorithm needs to be generic and allow constructing strategies for any of defined plans.

2.2. Structuring Goals and Plans

As we used to do before in our research, now we're going to apply all the defined and developed to the chess as a classical example of RGT game.

The goal in general needs to have the following structure.

A. It needs a preCondition situation, for which this goal is applicable, because there are situations where a goal is not achievable, e.g., if the situation contains only two kings and a pawn, a goal like "make check with the queen" can't be applied. This basically defines the pattern of situations where goal is meaningful. Note that for some goals the preCondition can be any situation, so this is not obligatory to define some pattern in preCondition.

B. It needs to have a postCondition situation. This is the situation which appears when the goal is achieved, e.g., if the goal is "make check with the queen", after it is achieved the opponent king is under check of queen in the given situation, this describes the postCondition situation. This defines the pattern of achieved by the goal situations. Similar to preCondition, postCondition also can be any situation.

C. For some goals the depth of game tree needs to be more than one move, e.g., if the goal is "make perpetual check", we need to construct a tree and make several moves to see if this goal can be achieved.

D. Goals need to have some evaluation. There are goals like "put mate" or "avoid stalemate" where there are only two evaluation states, which indicate whether the goal is achieved or not, but there are some goals which do not show "an achieved" or "not" result, they show how good the goal is achieved, e.g., a goal "keep king closer to the opponent king" goal does need some criterion to define that the lesser distance between kings is, the better is the goal evaluation. For that purpose we define evaluator, which is a set of prioritized criteria that are being defined to evaluate the goal. For the above example only one criterion exists and it is the distance between two kings.

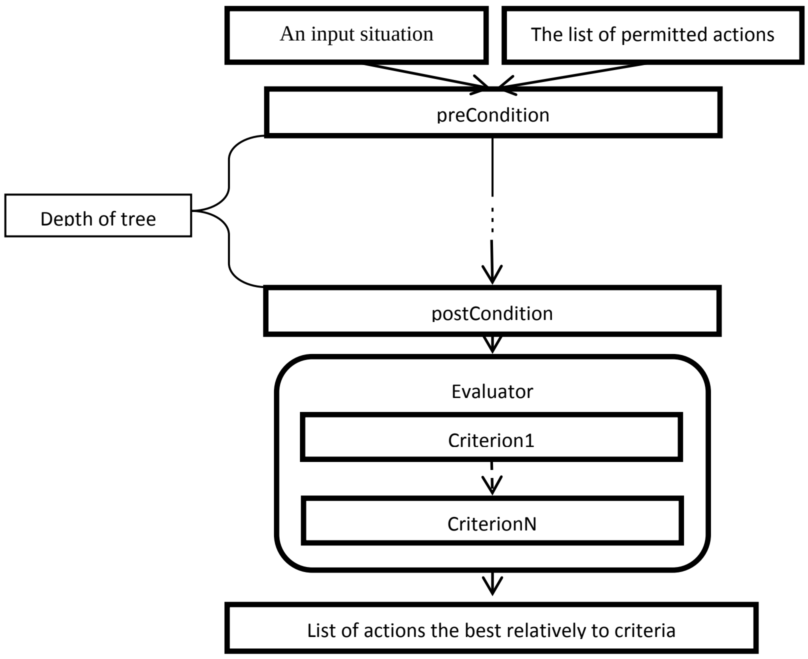

Fig 2. The structure of goals, their inputs and outputs.

From the described above we reveal that the goal consists of preCondition and postCondition, which are situations (in the Solver we define these situations as composite abstracts), depth of three, which is a number that defines how deep the tree can be constructed for checking if the postCondition is achieved (by default it is 1) and the evaluator which evaluates how good the goal is achieved.

Also one important point we need to define the concept of absolute goal (which is just a flag on the goal), like mate in chess and indicates that the game is over.

The plan structure is basically defined in previous works of our team and nothing more is required. It consists of prioritized goals.

2.3. Searching Strategies by Plans

Now when we have the structures of goals and plans, we can define how the algorithm should work to find the best move from the given plan.

As described above the goal and the plan are completely generic in their structures regardless of the problem they solve in RGT class and can be defined by a user, not only injected initially for a certain problem.

The algorithm we have developed to execute the plans and to choose the best action by the defined plan is the following.

As said above plan is a set of prioritized goals, we need to run over the goals and find the move which best satisfies the highest priority goal.

The algorithm initially requires input situation (IS). For IS Solver does matching and finds the list of active abstracts [8], where there are also actions active in that situation (the actions that are possible to perform in IS), let’s call the list of active in IS actions <A>. Let’s assume we have Plan Pl which has G1 to Gn goals in it (G1 has the highest priority and Gn has the lowest priority). For the given P1 plan, the algorithm will take goals from the highest to the lowest priority and do the following procedure.

1. Passing list of actions <A_{i-1}> (for G1 <A> is passed instead of <A_{i-1}>). If the list is empty then nothing can be done for this goal, just returning, else if the list has only one action, then the list is returned and the procedure is stopped, as nothing to do if only one action can be done, no need of further processing, we just do the action.
2. preCondition of the goal is checked against IS. If IS satisfies the pattern defined in the preCondition all actions in the action set <A_{i-1}> are applied to the IS situation and postCondition of current G_i is checked to be achieved. The goal is being evaluated by the criterion defined in the evaluator if there are any and the actions which satisfy the goal best are being returned in the list <A_i> (this list will be used in the next goal processing). An important point here is that if the goal is absolute and the list of actions achieving this goal is not empty, then the procedure is stopped after this step and the list of actions is returned.
3. If the returned list <A_i> is empty, <A_{i-1}> list is being used instead, otherwise if the returned list has only one item in it, the list is just returned and the procedure is stopped.
4. New Actions list is passed to the next goal and the procedure is being done for it from the beginning (1 to 4).
5. When the procedure is done for all goals or stopped somewhere while performing 1-4 steps, it returns the list of actions, which indicate the best actions to achieve the plan in the current situation. Any of those actions brings to the best move selection and thus brings to the best strategy for the given P1 plan.

Any action from the returned list of actions is being selected (we just select the first one) and applied to the IS. New situation is achieved after opponent's action, so we already have a changed situation, a new input situation. The plan execution starts again for the new situation and a new best move is selected for the plan. The algorithm is stopped when the highest priority goal is achieved or is not achievable at all (e.g., we have already put mate or no mate can be achieved), which means that either the strategy for the given plan already worked or cannot be achieved anymore.

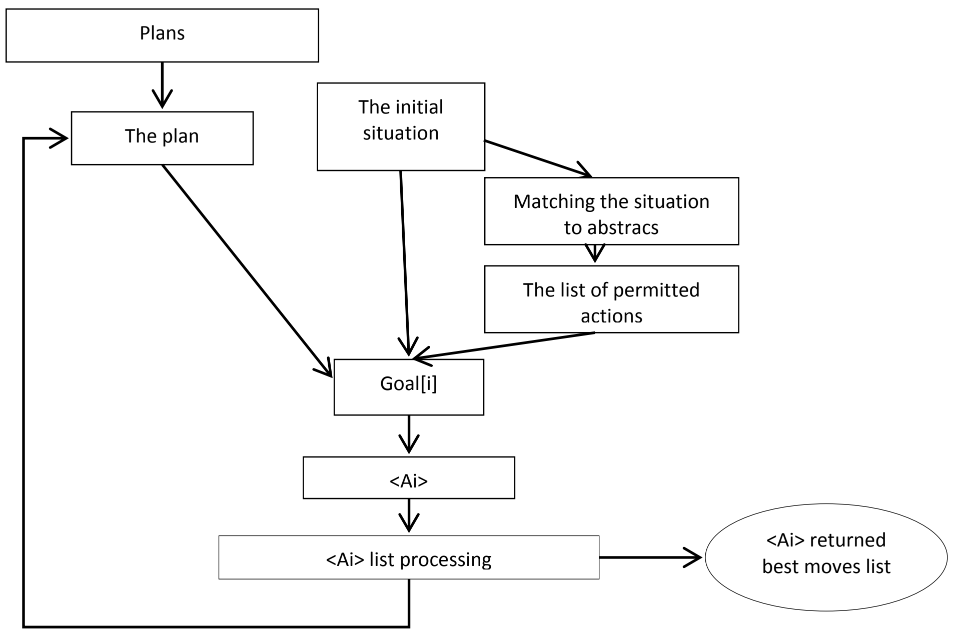
Plans
The plan
The initial situation
Matching the situation to abstracs
The list of permitted actions
Goal[i]
<Ai>
<Ai> list processing
<Ai> returned best moves list

Fig 3. The schema of searching the best moves.

3. Testing Adequacy of PPIT for Chess Endgames

3.1. Planning Chess Endgames

3.1.1. Planning "rook against king" endgame

Previously the strategy description language was defined in [9], where exact algorithms were used to define each plan and its realization. For the demonstration of the language adequacy "rook against king" chess endgame was described. For the demonstration of our algorithms we will also consider chess endgames, like "rook against king" or "two rooks against king".
To simplify the definitions we just assume our color is predefined and is white. We will try to define only the mate on one direction to make it simpler, the same is done in [9]. Let's take vertical direction only for our future definitions. Similarly we will be able to define putting mate on horizontal direction. Which one to choose vertical or horizontal is a job for another module in PPIT algorithms expected to be developed in the scope of Solver during the future steps of our research. Currently it will just construct strategy with the given certain plan.
A plan for the "rook against king" endgame will have the below goals

1. Put mate
2. Avoid stalemate (note that this is quite important because some situations can appear with stalemate and we need to avoid it)
3. Escape rook from attack
4. Push king to the edge (without putting rook under attack)
5. Make a waiting move when preOpposition appears
6. Bring white king closer to the black king

The definition of each goal is described in details.

1. Putting mate - preCondition is any situation, and postCondition is a situation where there's mate, the depth is 1, this is absolute goal. There is no evaluator defined for this goal.
2. Avoid stalemate - preCondition is again any situation and the postCondition is a situation where no stalemate appears. The depth is 1 and no evaluator again.
3. Escape rook from attack - the preCondition is "rook under kings attack" abstract, so the goal is applicable only for situations where the rook is under the opponent king's attack. The postCondition is a situation where rook is not under attack and the vertical coordinate of the rook is not changed. It has a depth value 1 and the evaluator will have one criterion defined which calculates the distance of the rook and opponent king by vertical direction.
4. Push king to the edge- preCondition can be any situation and postCondition is "rook is not under attack" situation and depth is 2. The evaluator has two criteria. First is: moves of opponent king are closer to the edge are better (this basically means the horizontal distance of opponent king from the edge is calculated and for each action the value of criterion is calculated as the highest value of king's distance from the edge). The second criterion for this goal evaluator is the number of actions opponent king can do, and the better action is the action which allows fewer number of actions by opponent king.
5. Make a waiting move when preOpposition appears - preCondition is preOpposition situation. PreOppositionByVertical abstract in the Solver can be defined as below.
   This is a virtual abstract which has two attributes – black and white kings. It must have 4 specifications

A. Whiteking.cordX = BlackKing.cordX + 2
whiteking.cordY = blackking.cordY + 1
B. Whiteking.cordX = BlackKing.cordX + 2
whiteking.cordY = blackking.cordY – 1
C. Whiteking.cordX = BlackKing.cordX - 2
whiteking.cordY = blackking.cordY + 1
D. Whiteking.cordX = BlackKing.cordX - 2
whiteking.cordY = blackking.cordY – 1

which is complete enough to define the precondition of preOpposition.
The postCondition is a situation where the king position is not changed and the rook vertical coordinate is not changed. Depth of goal is 1. The evaluator again has one criterion, which shows the distance of the rook from the opponent king.

6. Bring white king closer to the black king, but avoid opposition – preCondition and is any situation and postCondition is a situation where no opposition appears, depth is 1. The evaluator has one criterion, which defines the distance of the king from the opponent king to be minimal. We can calculate this by the following formula “(king.cordX-opponentKing.cordX)^2 + (king.cordY-opponentKing.cordY)^2”.

3.1.2. Planning “two rooks against king” Endgame

A winning plan for chess endgame “two rooks against king” will be

1. Put mate
2. Avoid stalemate
3. Escape rook from attack
4. Push king to the edge, where postCondition will be two rooks on the board and the criterion of evaluator will be only opponent king’s distance from edge is minimal.
5. Escape rook which vertical coordinate is different from opponent king’s coordinate by 1 (rook.y = king.y + 1 or rook.y = king.y - 1).

3.2. Searching for Winning Strategy of “rook against king”

Chapter 3.1 describes how chess endgames can be brought into Solver and this chapter describes the execution of the plans by the designed algorithm for “rook against king” example. For other plans its work is similar. Let’s see how the algorithm works for a situation.

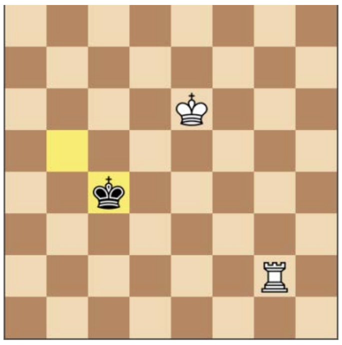

Fig 4. K., R. vs. B.K., An initial position.

1. Algorithm tries to find moves which bring to mate, and returns the empty list.
2. Since the returned list of the 1st goal is empty it takes the initial list of moves and returns the whole list of possible moves since all of them brings to situations where there is no stalemate, so the whole list of moves is passed to the 3rd step
3. "Escape rook from attack" goal is not applicable for this situation, so it just does nothing
4. "Push king to the edge" for all the moves that does not put rook under attack it calculates the first criterion value. Let's see what values it assigns to three of moves.
   a. 1. Rc2… this puts check to the black king, for all king moves it calculates the distance from the vertical edge. King moves can be Kd4, Kd3, Kb4… for Kd4 and Kd3 it assigns will assign the highest value of 4 (the distance from edge is 4). Kb4 will have value 2, so the value assigned to move Rc2 is 4.
   b. 1. Rd2… king can do moves Kc3, Kc5, Kb4… for Kb4 again value as mentioned above is 2, for Kc3 and Kc5 is 3, so the value for Rd2 move is 3.
   c. 1. Rg3… in this case also black king can move to d4 position, so the value will be 4.

   Similarly all moves other than Rd2 will have 4 value, the minimum value is 3, and only Rd2 has that, so after processing the 4th goal the algorithm will return move Rd2

Since only Rd2 move is returned the algorithm is not processed anymore and this move is applied.
Let's assume black does Kc3 move (attacking rook).

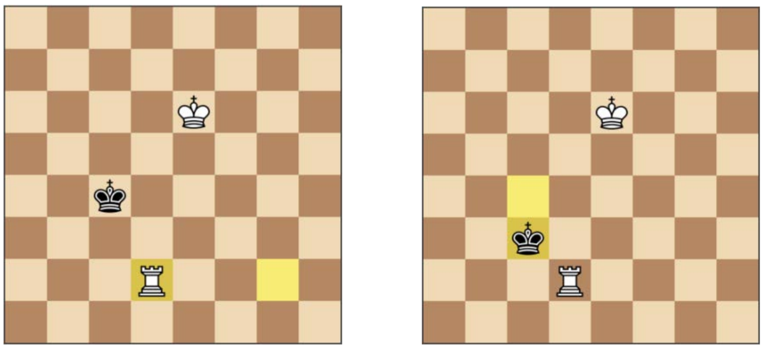

Fig 5. The left: the position after Rd2. The right: the position after Kc3.

After Kc3 move algorithm works again
1. For mate goal again empty list is returned
2. For stalemate all moves list is returned
3. "Escape rook from attack" goal's preCondition is matched to the situation and rook moves are considered to achieve the goal where rook is not under black king's attack since postCondition is "rook not under attack". The criterion to evaluate the move is vertical distance of rook and black king, so Rd8 move is chosen since it has the highest vertical distance from black king. Since the list has only one move in it, the procedure is stopped here and Rd8 move is returned

Rd8 is applied to the situation. Let's assume black makes Kc4 move.

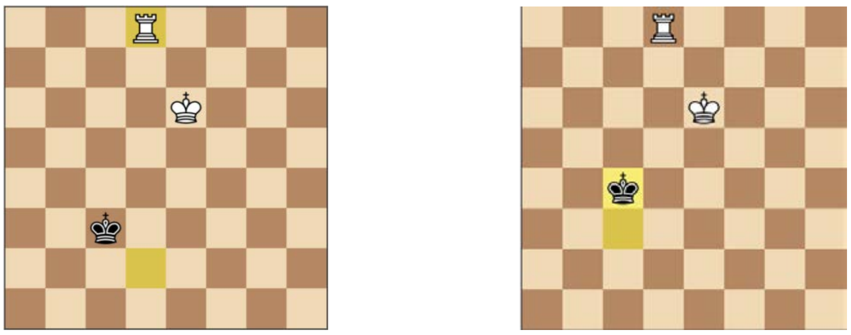

Fig 6. The left: the position after Rd8. The right: the position after Kc4.

Algorithm works again and now with the following result.

1. For goal mate again empty list is returned
2. For stalemate all moves list is returned
3. No rook under attack so this is just omitted
4. “Push king to the edge” for all the moves where rook is not under attack it checks the evaluator, which have two criteria, the 1st is kings distance from the edge is minimum. So for moves Rd1, Rd2, Rd6, Rd7,Ke7, Ke5, Kf7, Kf6, Kf5 the distance of king from the edge will be calculated as it was done for the 1st move, and the value will be 3, which is selected as the minimum value. Then the second criterion (which is the number of moves opponent king can make) is checked for the moves which are best for criterion 1. Number of moves of black king is always 5 for all the mentioned moves. So the whole list is returned from this goal processing procedure.
5. The situation is not a preOpposition, so preCondition is not matched, this goal is just omitted.
6. “Bringing king closer” preCondition is any situations, and postCondition is a situation where no opposition appears. The list of moves is [Rd1, Rd2, Rd6, Rd7, Ke7, Ke5, Kf7, Kf6, Kf5], which does not bring to opposition, so all of them satisfy postCondition. The evaluator criterion is that distance between two kings needs to be minimum. For the moves by rook distance value will be 8 ($(5 -3)^2 + (6-4)^2$). For king moving by f vertical the value will be rising, e.g., after Kf6 criterion returns 13 ($(6 -3)^2 + (6-4)^2$). The best move will be Ke5, which will have evaluation value 5 ($(5 -3)^2 + (5-4)^2$). Ke5 will be returned.

Since only Ke5 is returned this is applied to the situation. To make the example shorter let’s consider Kd5 move for black.

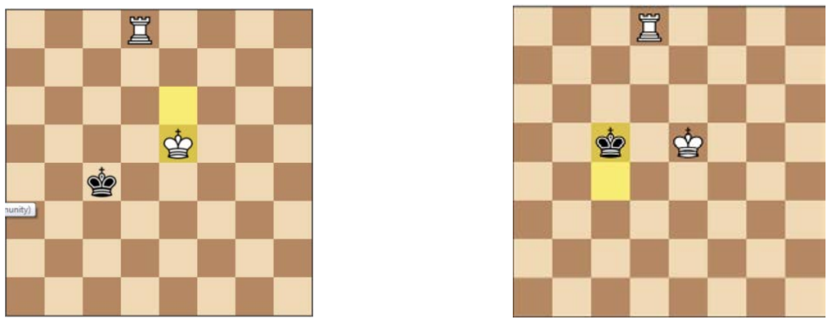

Fig 7. The left: the position after Ke5. The right: the position after Kc5.

After Kc5 move similar to the 1st move for "push king to the edge goal" Rc8 move will be selected. Again we will assign black king moves which finish the game sooner, we will consider the move Kb4. So after the following moves

1. Rd2 Kc3 2. Rd8 Kc4 3. Ke5 Kc5 4. Rc8 Kb4 5. Kd5 Kb5 6. Rb8 Ka4 7. Kc5 Ka3 8. Kc4 Ka2 9. Kc3 Ka1 10. Kc2 Ka2.

After the 10th move (Ka2 by black) the algorithm will work and find that mate is achievable and Ra8 move will be returned. This move will be applied and the plan is achieved.

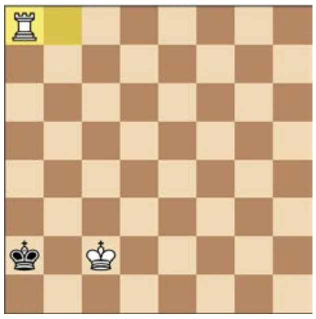

Fig 8. The position of putting mate.

4. Conclusion

1. Structures of plans and goals are defined for the Solver of RGT class allowing user to describe generic plans and goals for any problem of this class in a regular manner. Goals are defined as a composition of preCondition, postCondition situations, depth of game tree to achieve the goal and evaluator to evaluate the utility achieved in a situation while accomplishing the goal. Plans are sets of prioritized goals.
2. An algorithm of searching strategy by a plan was constructed and developed based on PPIT algorithms previously developed by our team for certain problems and with injected knowledge usage. The algorithm works only with defined plans and goals, regardless of the problem it solves. Previously the constructed PPIT consists of three modules RHP, CPMU, GMP.
   a. In the following we developed algorithms for GMP module
   b. Future development of other modules within the scope of Solver to complete PPIT algorithm are in progress now, which is related to constructing algorithms to choosing the best plan from the given list of plans. This corresponds to CPMU module.

   For the current state we assume that expert knowledge for plans is being defined by a user but in the development process we aim to achieve creating algorithms for Solver to generate plans by itself relying on the knowledge set it already has for the game.
3. Demonstration of the structures and the algorithms were carried out for chess endgames, their adequacy is shown. More experiments are in progress now for different chess situations, particularly Reti etude planning is in progress now.

Acknowledgement

Author expresses his deep gratitude to Professor Edward Pogossian for supervising the work and to Boris Karapetyan for support in defining the winning plan of chess endgame “Rook against king”.

References

[1] Ch. Brutyan, I. Zaslavski and L. Mkrtchyan, “On methods of automated synthesis of positional strategies in games”, *Problemi Kibernetiki,* Moscow, Russia, vol. 19, pp. 141-175, 1967.

[2] E. Pogossian, V. Vahradyan and A. Grigoryan, “On competing agents consistent with expert knowledge”, *Lecture Notes in Computer Science*, *AIS-ADM-07: The International Workshop on Autonomous Intelligent Systems - Agents and Data Mining*, pp. 229-241, St. Petersburg, Russia, June 6-7, 2007.

[3] E. Pogossian, A. Javadyan and E. Ivanyan, “Effective discovery of intrusion protection strategies”, *The International Workshop on Agents and Data Mining, Lecture Notes in Computer Science,* St. Petersburg, Russia, vol. 3505, pp. 263-274, 2005.

[4] M. M. Botvinnik, *About Solving Approximate Problems,* (in Russian), S. Radio, Moscow, 1979.

[5] M.M. Botvinnik, “Computers in chess: solving inexact search problems”, *Springer Series in Symbolic Computation, with Appendixes, Springer-Verlag,* New York, 1984.

[6] E. Pogossian, V. Vahradyan and A. Grigoryan, “Experiments in consistency of chess expertise with decision making for etudes of Retie and Nodareishvili”, *Transactions of IIAP NAS RA, Mathematical Problems of Computer Science,* (in Russian), vol. 28, pp. 94—113, 2007.

[7] K. Khachatryan and S. Grigoryan, “Java programs for presentation and acquisition of meanings in SSRGT games”, *Proceedings of SEUA Annual conference,* pp. 127-135, Yerevan, Armenia, 2013.

[8] K. Khachatryan and S. Grigoryan, “Java programs for matching situations to the meanings of SSRGT games”, *Proceedings of SEUA Annual conference,* pp. 135-141 Yerevan, Armenia, 2013.

[9] B. Karapetyan, “High level strategy description language in games”, *Mathematical Problems of Computer Science,* (in Russian), vol. 16, pp. 167-183, 1986.

[10] E. Pogossian, “On modeling cognition”, *International Conference in Computer Sciences and Information Technologies,* Yerevan, Armenia, Sept. 26-30, pp. 194-198, 2011.

Submitted 10.12.2014, accepted 16.02.2015.

Նպատակների և պլանների կառուցում անձնավորված պլանավորման և պլանների ինտեգրացված թեստավորման համար

Ս. Գրիգորյան

Ամփոփում

Մենք ուսումնասիրում ենք մրցակցային խնդիրները՝ սահմանված որպես դաս, որտեղ լուծումների բազմությունը վերարտադրելի ծառ է (RGT)։ Մշակված են Անձնավորված պլանավորման և ինտեգրացված թեստավորման ալգորիթմներ RGT խնդիրներում լավագույն ռազմավարության փնտրման համար։ Աշխատանքում զարգացվում են նպատակների և պլանների կառուցվածքներ, կառուցվում է ռազմավարության փնտրման ալգորիթմ ըստ պլանի և ցուցադրվում է նրանց հիմնավորությունը։

Структурирование целей и планов для персонализированного планирования и интегрированного тестирования

С. Григорян

Аннотация

Разработаны алгоритмы и программы представления планов и целей при решении задач класса RGT. Представлено описание поиска стратегий на основе планов для пакета Solver. Обоснованность алгоритмов показана на примере шахматных эндшпилей.