Can Real-Time Search Algorithms Meet Deadlines?

Babak Hamidzadeh
Shashi Shekhar

Computer Science Dept.
University of Minnesota
Minneapolis, MN 55455

ABSTRACT

A real-time AI problem solver performs a task or a set of tasks in two phases: planning and execution. Under real-time constraints, a real-time AI problem solver must balance the planning and the execution phases of its operation to comply with deadlines. This paper provides a methodology for specification and analysis of real-time AI problems and problem solvers. This methodology is demonstrated via domain analysis of the real-time path planning problem and via algorithm analysis of DYNORAII and RTA*[1]. We provide new results on worst-case complexity of the problem. We also provide experimental evaluation of DYNORAII and RTA* for deadline compliance.

1. Introduction and Survey

A real-time AI problem solver typically needs to address response-time constraints. In lightning chess, for example, the problem solver has to search for a move in the state space, within a time limit. Any amount of time saved within a move can be used in later moves. Thus, the lightning chess problem solver has to produce responses within the deadline put on a move, or in less time than that in order to buy time for more complicated decision problems to come. Response-time constraints relate to the limitation on the total time to plan a solution and to carry out the solution. Response-time constraint problems can further be divided into two classes: deadlines and optimal response times. Deadline situations provide a certain amount of time for the system to plan a solution. The deadlines may be fixed or they may vary from case to case. Finding an optimal solution within arbitrary deadlines is hard and often NP-complete [2]. Anytime algorithms [3, 4] formalize characteristics of a class of algorithms which are capable of handling variable deadlines.

Another class of response-time constraint problems impose optimality constraints on the total response-time of the system. The total response-time is the sum of the time spent on planning a solution and the time it takes to execute the solution. The optimal response-time is not achieved by planning for optimal solutions since it may incur large planning costs. A trade-off between planning time and solution quality may be used to optimize the total response-time [5].

Simple blind search algorithms like depth first search, breadth first search and depth first iterative deepening [6] are useful for problem solving in small search spaces and situations where tight deadlines are non-existent. Most real-world applications, however, face very large search spaces and, often times, constraints on response time. Classical search algorithms, such as A* [7] and IDA* [8] which guarantee optimal solutions in terms of execution times, do not guarantee meeting any constraints on response time.

Anytime algorithms characterize the requirements of decision procedures capable of meeting deadline constraints on planning time[3]. The utility of solutions planned via these algorithms increases over time. The algorithms can be terminated at any time and will return some answer at the time of termination. The answer returned improves if more time is available for planning. Meta-Greedy algorithm[9] is an anytime algorithm. It uses a sequence of evaluation functions to evaluate the promise of a node during search. A greedy approach is used to order the multiple evaluation functions. Negative local benefit from a step of planning terminates the search in that direction. The algorithm may be terminated at any time and it will produce a solution at that time.

NORA[5] uses hierarchical planning to improve the solution at hand via the set of semantic information for database query planning. NORA formalizes the trade-off between planning cost and execution cost to address constraints on the total response time. This algorithm assumes that the set of all solutions are available at planning time so it can pick one among them. It has been formally proven that the stopping criterion of NORA provides near optimal response-times.

A framework to address the more general problem of resource constraints may be built around utility theory[10, 11]. This model calculates utility and disutility of certain meta-level actions. It then uses these values to reason about continuing to plan or proceeding with an action. The utility values and the probability distributions are learned through experience.

Problem solving in dynamic worlds has been addressed by RTA*[1, 12]. The algorithm works in cycles of partial planning followed by execution. The complete plan to reach the goal is not worked out if planning takes a long time. The agent executes a partial plan without exploring all the consequences this commitment. RTA* uses a variation of minmax search [13], called minmin look-ahead search for partial planning. Minmin search looks forward from the current state to a fixed depth horizon and applies the heuristic evaluation function (f=g+h) of A* to the nodes at the depth frontier. The best f value is then sent back to the current node. Korf has proven that RTA* is a complete and correct algorithm, namely it finds a solution if one exists, and executing the solution will, indeed, achieve the desired results. DYnamic Near Optimal Response-time Algorithm (DYNORA)[14] was designed to address both response-time constraints and issues concerning dynamic

environments, simultaneously. As a means of characterizing response-times, DYNORA formalizes the trade-off between solution quality and planning time, which is very unique. Also, by combining partial planning with execution DYNORA adds a reactive behavior to its problem solving nature.

The research in real-time search algorithms has not provided adequate validation of the performance of the proposed algorithms. A part of analyzing a real-time algorithm must demonstrate the capability of such algorithms to handle strict time constraints. It is difficult to compare the alternative algorithms in their performance and in difficulty of formulation for specific real-world applications. This paper makes several contributions to real-time problem solving research in AI.

This paper provides a domain analysis of the real-time path planning problem, in section 2. The analyses of the problem will reveal certain constraints on how well we can expect any real-time algorithm to perform. We introduce a new real-time planning algorithm, named DYNORAII, in section 3.1. For the proposed algorithm, we provide formal and empirical performance analyses. In section 3.2 we show that the time distribution of the number of completed jobs can be used to test the capability of an algorithm to handle strict time constraints. Our empirical results, derived from experimentation on DYNORAII and other real-time planning algorithms, show that DYNORAII outperforms RTA* in compliance with deadlines.

2. Domain Analysis: Optimal Path Planning Problem

The analysis methodology can be divided into two parts: Domain analysis and algorithm analysis. In the domain analysis, the specified problem is analyzed independently of the algorithm that is intended to solve it. In the algorithm analysis, the algorithm that is intended to solve the specified problem is analyzed. The domain analysis can consist of some analytical results on worst-case time complexity of the problem. Other analysis can consist of solution density and solution quality analysis. A solution density analysis is concerned with the number of solutions that exist in an average-case or a worst-case state space. A solution quality analysis is concerned with how good a solution, in terms of some parameter(s) of interest, one can expect from an algorithm. Both of the latter analyses, namely the solution density and solution quality analysis, can help to determine the worst-case time complexity of the specified problem.

An AI problem solver can be characterized by two main components. One component is the state space in which a solution is to be found. The other component is the search algorithm that is used to find the solution[15]. A state space is represented via a graph G(V, E). A state in G is represented by a node $v_i \in V$. Certain pairs of states are connected to each other via edges $(v_i\ v_j) \in E$. The two nodes representing an edge are the states that are connected by that edge in the state space G. Associated with each edge $(v_i\ v_j)$ is an execution cost $ce(v_i,v_j)$ which is regarded as the cost of transforming v_i to v_j, or the cost of traversing the edge $(v_i\ v_j)$. Also associated with each edge, is a planning cost $cp(v_i,v_j)$ which is the amount of planning required to come to the decision to traverse the edge $(v_i\ v_j)$.

Associated with each graph are two special nodes; the start node s and the goal node g. The problem of planning in a graph is that of searching for a path p_i that connects s and g via existing edges in the graph. A path (plan) p_i is represented as an ordered set of edges $\{(s\ v_l), (v_l\ v_j), \cdots, (v_m\ g)\}$. Associated with a path p_i in the graph is an execution cost C_e such that $C_e = \sum_{(v_l,v_m)\in p_i} ce(v_l,v_m)$, and a planning cost C_p such that $C_p = \sum_{(v_l,v_m)\in p_i} cp(v_l,v_m)$. An optimal path refers to a plan that reaches the goal node with the smallest possible execution cost. In dynamic state spaces, like the execution costs of edges, the total execution cost of a path is a function of time. An optimal path in such spaces refers to a plan that remains optimal over a period of time.

The constraints on the total response time of the planning and execution processes of real-time problem solvers can be specified as follows. Total response times are characterized by the sum of the time it takes to plan a solution and the time it takes to execute that solution. If the execution cost C_e and the planning cost C_p of a solution path in a state space are calculated in terms of time, the total response time of a plan can be calculated as $C_p + C_e$. Strict time constraints on total response times typically pose deadlines on the amount of time available for planning and execution of a solution. In such situations a planning algorithm is required to plan a path from the start node to the goal node and execute that path, namely traverse all edges in the path and end up at the goal state, before a deadline is reached. Deadline situations require guarantees on total response times.

To characterize real-time path planning as a hard problem, we provide a worst-case time complexity analysis of optimal real-time planning. We specify the problem of optimal real-time planning as that of finding an optimal solution within a deadline or finding and executing an optimal solution in the minimum total response time. We present a graph-theoretic definition of this problem as follows. We define an optimal path in this problem to be the path connecting s and g with the smallest execution cost C_e. The time, C_p, spent to search for a solution path is calculated by adding up the cost of planning the move from one node to another for all the nodes on the path. The cost of a move from one node to another is calculated by adding up the expansion cost of every node that was expanded during planning for such a move. The total response time for a solution path is calculated by adding up the cost of planning the path and the cost of traversing that path, namely $C_p + C_e$. It can be shown that the problem of finding and traversing the optimal path (i.e. $Min(C_e)$) in the minimum time possible (i.e. $Min(C_p + C_e)$) is NP-complete. An implication of this result is that guaranteeing optimal solutions within arbitrary deadlines is NP-complete. This implication is posed as a corollary at the end of this section. We will provide a proof by using the method of polynomial reduction [16].

We define the optimal real-time search (ORTS) problem as follows. Assume we are given a graph G(V, E), with |V|=n and |E|=m. We measure the size of the problem instance by the sum of the number of nodes and the number of edges, namely m+n. Given two nodes s

and g in V, our problem is to find and traverse the shortest path connecting s and g such that the total response time for such a path is a minimum. Therefore, a solution to this problem is a set of vertices V' *in* V $(V'=\{v'_1,v'_2,...,v'_g\},\ v'_1=s,\ v'_g=g)$ such that the path connecting s and g is the shortest path in the graph, namely: $C_e = \sum_{i=1}^{g} ce(v'_i,v'_{i+1})$ is a minimum, and the total response time to search for and i execute this path is also a minimum, namely: $C_p + C_e = \sum_{j=1}^{g} cp(v'_j,v'_{j+1}) + \sum_{i=1}^{g} ce(v'_i,v'_{i+1})$ is a minimum.

Theorem 1: Given an instance I of the ORTS problem and positive integers L & P, the problem of answering whether there is a path of length less than L with a total response time less than P+L is NP-complete.

PROOF: The ORTS problem is in NP, since a nondeterministic polynomial time algorithm can solve the decision version of the ORTS (DORTS) problem. This algorithm nondeterministically guesses a solution path, connecting nodes s and g in a graph, and verifies that the length of the path is less than L and the response time is less than P+L.

Next, we reduce the problem of "Shortest Weight-Constrained path" (SWCP), a known NP-complete problem[16], to the ORTS problem in polynomial time. SWCP can be stated as follows: Given a graph G(V,E), length $l(e)\varepsilon Z^+$ (where Z^+ is the set of all positive integers), weight $w(e)\varepsilon Z^+$ for each $e\varepsilon E$, specified vertices $s,t\varepsilon V$, and positive integers K and W, is there a simple path in G from s to t with total weight W or less and total length K or less? A direct, polynomial transformation of and instance I' of SWCP to and instance I of DORTS follows: Graph G in I' is the same as G in I, with the same set of vertices and edges. Each l(e) in I' has a corresponding $ce(v_i,v_j)$ in I, such that $l=ce$ and e is the edge in E which connects the vertices v_i and v_j $(v_i,v_j\varepsilon V)$. Similarly, each w(e) in I' has a corresponding $cp(v_i,v_j) + ce(v_i,v_j)$ in I, such that $w=cp+ce$ and e is the edge in E which connects the vertices v_i and v_j $(v_i,v_j\varepsilon V)$. It is clear that the transformation function can be carried out in polynomial time. From these we deduce that the optimal real-time search problem is NP-complete. We note that the ORTS and SWCP problems are not NP-complete for the special case where weights W(e) are identical or the lengths l(e) are identical.

One implication of the above results is that we should not seek optimal solutions for the real-time search problem without characterizing special cases. This is reflected in the algorithms that address the problem. Other implications of these results are reflected in the following corollaries.

Corollary 1: The problem of searching for a path of optimal length, within a deadline is NP-hard.

Corollary 2: The problem of searching for a path of optimal length, while optimizing total response times is NP-hard.

The proofs of these corollaries are based on the corresponding decision problem of finding and executing a path of length L within time P+L.

3. Algorithm Analysis: Real-Time Path Planners

In this section we analyze two real-time path planners. One of these path planners uses our new real-time search algorithm, DYNORAII, that is capable of handling time constraints in dynamic environments. We prove the new algorithm to be correct and complete in static worlds. We, then, provide the data from a set of comparison experiments that test the capability of these algorithms in meeting deadlines. Analysis of the data show that DYNORAII is capable of meeting much tighter deadlines with higher degrees of reliability.

Analysis of the algorithm starts with correctness and completeness analyses. An algorithm is correct if the solution that it produces does indeed solve the specified problem. An algorithm is complete if it is guaranteed to find a solution when such a solution exists. The algorithm analysis can also contain verifications of optimality of the solution provided by the algorithm.

A real-time AI problem solver is amenable to a variety of other analyses, including deadline compliance, which refers to the problem solver's ability to meet a given deadline. In graph-theoretic framework, deadline compliance can be analyzed for a problem defined on a fixed graph. For example, the path planner can be analyzed to check the number of (start, goal) pairs, for which a path can be discovered and traversed within a given deadline. A distribution graph, representing the number of jobs that were able to meet a given deadline, provides a representation of deadline compliance ability.

3.1. Algorithm Specification: DYNORAII

DYNORAII performs planning and execution cycles repeatedly until a goal node is reached, assuming that the graph has a solution. A plan-execute cycle consists first of conducting a heuristic search (plan phase) for the next move starting at the current state (node) in the graph. The search continues from the start state to a certain depth in the graph until the following stopping criterion is reached:

$$C_p \geq \alpha C_e \quad (1)$$

This stopping criterion provides a tradeoff between planning costs (C_p) and execution costs (C_e). This tradeoff takes into account the utility of the heuristic solution found in the current plan phase versus the amount of planning that was performed to find that solution. At the execute phase, the algorithm commits to the best action found during the previous plan phase. In the case of path planning for a robot, for example, the execute phase consists of physically moving the robot from its current position to its next position which was chosen among a set of available options. Figure 1. provides pseudo-code of the DYNORAII algorithm. C_p in DYNORAII, is determined by the number of nodes that were evaluated during the search. These are the nodes whose h,g and f values were calculated, where h is the estimated distance from the evaluated node to the goal, g is the the cost of a move from the parent of the evaluated node to the evaluated node, and f is the sum of h and g. C_e is calculated by adding the actual length of the current path to the estimated distance between the current node and the goal node. When the criterion of inequality 1 is satisfied, the smallest f value found so far is returned to the top level of

the algorithm. The successor node with the smallest f value is chosen as the next physical move for the DYNORAII algorithm. This process is repreated until a solution is reached.

1.Put the start node on the queue.
2.Until the first node in the queue is the same as the goal node or the stopping criterion is satisfied:
 2.1.If the first node in queue is the same as the goal node,
  2.1.1.Announce success.
 2.2.If the stopping criterion is satisfied,
  2.2.1.Execute the best action found.
 2.3.Otherwise:
  2.3.1.Remove the first node from the queue.
  2.3.2.Create a list of successor nodes of the removed node.
  2.3.3.For each successor node,
   2.3.3.1.Assign cost of edge between the removed node and the successor node as cost value (i.e. g) of successor node.
   2.3.3.2.Assign the value returned by the heuristic function to the heuristic value (i.e. h) of the successor node.
  2.3.4.Add, to queue, the list of those successor nodes that are not already on the queue.
  2.3.5.Sort nodes in queue with respect to their sum of cost and heuristic values (i.e. f=g+h).

Figure 1: Pseudo Code for DYNORAII

An important parameter involved in the tradeoff between C_p and C_e is α (see inequality 1). The appropriate value of α depends on certain characteristics of the graph and the application at hand. Examples of such characteristics are the graph size, the branching factor and the time available for planning. The general rule of thumb is to choose a large α when the search space is small, the branching factor is small and the time to plan is long. A small α is chosen when the search space is large, the branching factor is large and the time to plan is short. The intuitive rationale behind these heuristics is that a large graph or a high branching factor with a large α can considerably increase the amount of planning. Also, when the available time to plan is short one must obviously reduce the amount of planning in each plan-execute cycle.

DYNORAII guarantees termination if a solution path from start node to goal node exists in the graph. It also is able to get out of local minima and graph cycles. This is done by penalizing cyclic and dead-end paths, and by leaving the h value of the second best path at each decision point [1]. Next, we will present some formal results about the algorithm and its performance. Empirical results based on performance comparison experiments will follow.

Theorem 2: DYNORAII is correct and complete. Proofs of the this theorem are provided in[17]

3.2. Deadline Compliance Evaluation

A part of analyzing a real-time algorithm must demonstrate the capability of such algorithms to handle strict time constraints. In this section we present the results of experiments that were designed to compare the performance of DYNORAII and RTA* in meeting deadlines.

This experiment was conducted on a randomly generated graph, G(V, E), with 30 nodes. V in the graph represents the set of nodes and E represents the edges in the graph. The nodes of the graph are represented by their euclidean coordinates within a 100x100 coordinate system. The edges of the graph are represented by the two nodes at each end. The size of the graph is characterized by the number of nodes (n) in the graph. The degree of connectivity (β) of the graph of this experiment was chosen to be 4/n. This avoids extremely small solution spaces (set of possible solution paths), as well as, trivial paths to the goal. For each possible pair of start and goal nodes in the graph (a total of 870 start-goal pairs) eight sets of data were collected. four of the eight sets corresponded to RTA*(n) for n=1,2,3, and 4. The other four sets corresponded to DYNORAII(α) for α = 0.05, 0.1, 0.5, and 1. We collected data on path length and number of nodes expanded to find the path. The former is a measure of execution cost and latter is a measure of planning cost.

In face of strict time constraints an algorithm needs to guarantee results within a given deadline, namely an algorithm needs to specify the minimum amount of time that it requires to guarantee the completion of all the given jobs. Alternatively, an algorithm is required to provide the fraction of the jobs that can be completed within any given deadline. For our problem of planning a path in a state space graph, the average, best and worst total response times of the search between the set of all possible start and goal nodes will provide the distribution of job completions over time. This distribution will allow calculation of the number of jobs that will be guaranteed to be completed within arbitrary deadlines.

Figures 2, 3, 4, 5, and 6 provide the results of analyzing the data produced by searching for a path between all possible start and goal nodes in the specified graph. These figures represent the population distribution of problem instances over deadlines. The population distribution represents the percentage of problem instances, which can be solved with a response time less than the given deadline.

Figure 2 demonstrates the effect of parameter α on DYNORAII's capability to meet deadlines. Figure 3 demonstrates the same capability for RTA* and different values of the look-ahead parameter n. Figures 4, 5, and 6 represent the best case, worst case and average case performances of the two problem solvers. The average case population distribution for a given deadline is derived by computing the average population distribution for the given deadline over possible values of performance parameter α and n.

The effect of performance parameters α and n on the problem solvers can be summarized as follows. DYNORAII's ability to meet shorter deadlines increases as the value of α increases, as shown in Figure 2. However, that RTA*'s ability to meet deadlines decreases as the depth of search increases, as shown in Figure 3.

The comparative performance of the two problem solvers can be judged by examining Figures 4, 5, and 6 which show the best, worst and average case comparisons of deadline compliance. DYNORAII outperforms RTA* in all three cases. This implies that DYNORAII can guarantee a response in a significantly shorter time than RTA*. Table 1 demonstrates the times that it takes both algorithms to complete %100 of the jobs. Table 2 demonstrates the times that it takes both algorithms to complete %90 of the jobs. The data in table 1 represent the

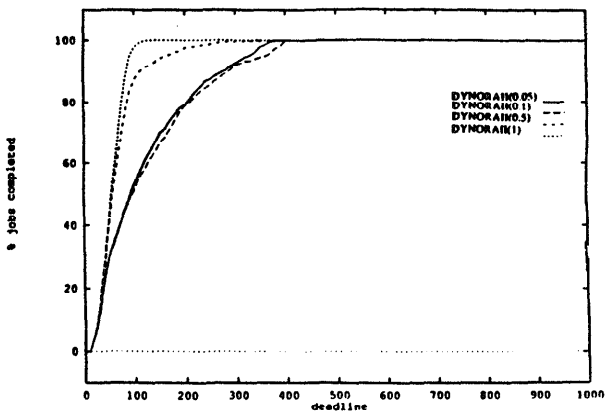

Figure 2: Effect of a on DYNORAII(a)'s Deadline Compliance

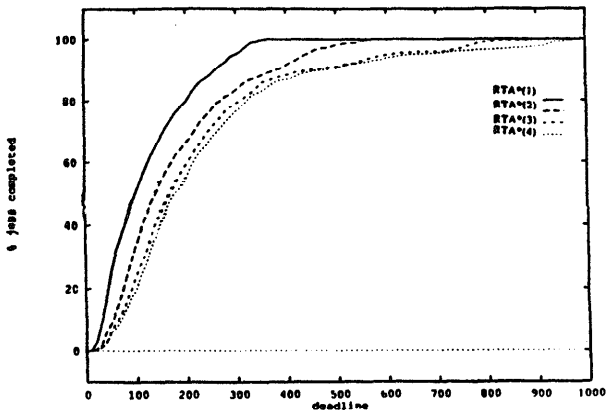

Figure 3: Effect of n on RTA*(n)'s Deadline Compliance

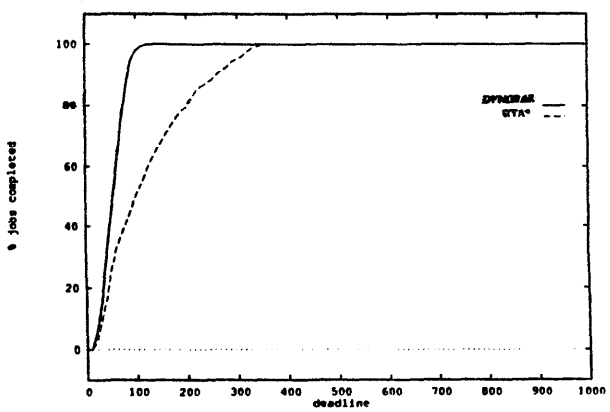

Figure 4: Best-case performance of DYNORAII vs. RTA*

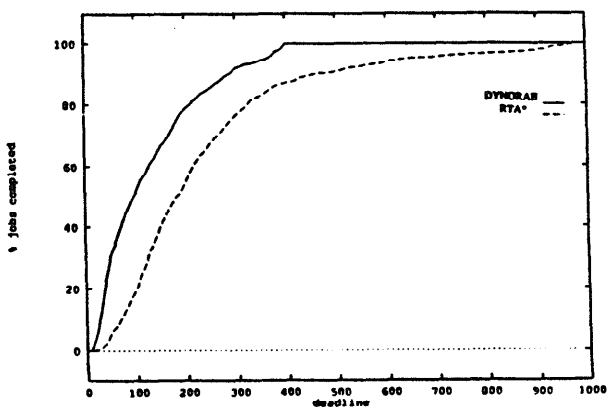

Figure 5: Worst-case Performance of DYNORAII vs. RTA*

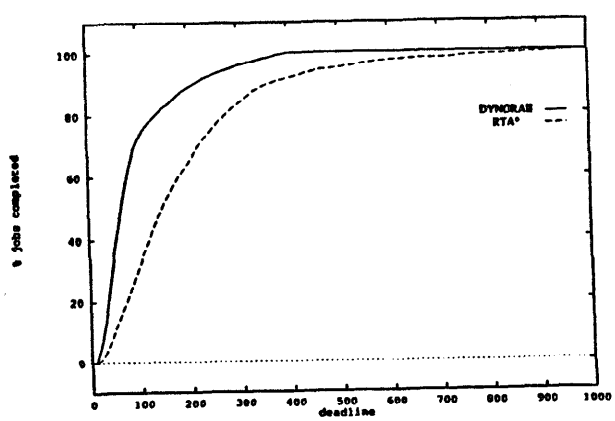

Figure 6: Average-case Performance of DYNORAII vs. RTA*

deadlines within which complete response can be expected from each algorithm. The rows of the table present best, worst and average case response times to complete %100 of the jobs at hand, respectively. The first two columns of the tables list best, worst and average times for DYNORAII and RTA* algorithms. The third column shows the percentage of improvement of DYNORAII in meeting deadlines, over RTA*.

	RTA*	DYNORAII	$\frac{RTA^* - DYNORAII}{RTA^*} \times 100$
Best	350	120	65.7
Worst	970	400	63.3
Average	970	400	63.3

Table 1: Best, Worst & Average Response Times for %100 job completions

The results of the previous section demonstrate that DYNORAII outperforms RTA* in meeting deadlines, in the average, best and worst cases. Collection and analysis of all possible searches of a specific graph allows us to produce a distribution of all jobs across total response times. The best and worst case analysis of algorithm performance requires a complete sample of all problem instances. It is this distribution that allows the average, best and worst case analysis of algorithm performance.

	RTA*	DYNORAII	$\frac{RTA^* - DYNORAII}{RTA^*} \times 100$
Best	260	85	67.3
Worst	460	280	39.1
Average	360	210	41.6

Table 2: Best, Worst & Average Response Times for %90 job completions

Even though table 1 provides equal values for %100 job completions in the worst case and the average case, these values are not equal for less than %100 job completions as shown in table 2. The average performance of the algorithms are better than their worst-case performance for less than %100 job completions. This is evident in the graphs of figures 5 and 6. In these graphs the average-case results are shown to reach %100 job completions at steeper slopes than the worst-case results.

4. Conclusion

Real-time problem solvers in AI can meet several but not all deadlines. Each real-time AI problem solver and algorithm should be analyzed. Domain analysis of real-time AI problem solver can reveal the constraints on the best possible performance of any algorithm, for the given detail of problem specification. Algorithm analysis can reveal the deadline compliance ability of specific algorithms in three ways. It characterizes the percentage of problem instances that can be solved by a given deadline. It can also discover the range of deadlines within which the algorithm guarantees a solution to a given problem instance. Finally, it can indicate the expected success of a given algorithm in meeting a given deadline for different problem instances.

We have presented a specification of real-time path planning problem and the real-time problem solvers for path planning. We have shown that real-time path planning is a hard problem in order to justify the heuristic approach taken by real-time AI researchers to this problem. We have provided a new heuristic real-time path planning algorithm, DYNORAII. We experimentally show that the proposed algorithm outperforms traditional real-time AI algorithms in deadline compliance. We plan to extend the analysis model to larger problems such as real-time planning and to larger applications such as robot path planning.

5. References

1. R.E. Korf, RealTime Heuristic Search First Results, *Proc. AAAI Conference*, (1987).
2. M. R. Garey and D. S. Johnson, *Computers and Intractability*, W.H.Freeman and Company, New York (1979).
3. T. Dean and M. Boddy, An Analysis of Time Dependent Planning, *Proc. AAAI*, pp. 49-54 (1988).
4. T. Dean and G. Siege, An approach to reasoning about continuous change for applications in planning, *Proc. AAAI*, pp. 132-137 (1990).
5. S. Shekhar and S. Dutta, Minimizing Response Times In Real Time Planning And Search, *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pp. 238-242 IJCAI, (1989).
6. R. E. Korf, Search: a survey of recent results, *Exploring artificial intelligence (Ed. H. Shrobe)*, Morgan Kauffman, (1988).
7. P. E. Hart, N. J. Nilsson, and B. Raphael, A Formal Basis For the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2) pp. 100-107 (1968).
8. R. E. Korf, Depth-First Iterative Deepening : An Optimal Admissible Tree Search, *Artificial Intelligence* 27 pp. 97-109 North-Holland, (1985).
9. S. Russell and E. H. Wefald, Decision Theoretic Control of Reasoning: General Theory and an Algorithm to Game Playing, *Report No. UCB/CSD 88/435*, p. Computer Science Division, U.C.Berkeley (1988).
10. E. J. Horvitz, G. F. Cooper, and D. E. Heckerman, Reflection and Action Under Scarce Resources: Theoretical Principles and Empirical Study, *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pp. 1121-1127 IJCAI, (1989).
11. M. Boddy, Anytime Problem Solving Using Dynamic Programming, *Proc. Ninth National Conference on Artificial Intelligence*, AAAI, (1991).
12. R.E. Korf, RealTime Heuristic Search New Results, *Proc. AAAI Conference*, (1988).
13. C. E. Shannon, Programming a Computer For Playing Chess, *Philosophical Magazine* 41 pp. 256-275 (1950).
14. B. Hamidzadeh and S. Shekhar, DYNORA: A Real-Time Planning Algorithm to Meet Response Time Constraints in Dynamic Environments, *Proc. Tools for Artificial Intelligence Conference*, TAI, (1991).
15. N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA (1980).
16. M. R. Garey and D. S. Johnson, Complexity Results for Multiprocessor Scheduling Under Resource Constraints, *SIAM Journal of Computing*, pp. 397-411 (1975).
17. B. Hamidzadeh and S. Shekhar, Specification and Analysis of Real-Time AI Problem Solvers, *Submitted to the IEEE Transactions on Software Engineering*, ().