Real-Time Path Planning for a Robotic Arm

Kavita Krishnaswamy, Jennifer Sleeman, Tim Oates
University of Maryland, Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
{kavi1, jsleem1, oates}@umbc.edu

ABSTRACT

With robotics technology, services can be provided to care for individuals with disabilities. This paper describes an effort to improve path planning performance for a robotic arm, resulting in faster user response in real-time. For a robotic arm, particularly with multiple degrees of freedom, path planning is computationally expensive. We propose that it is possible to achieve rapid response times with an assistive robotic arm by caching frequent arm trajectories and creating a "roadmap" of arm movements. By calculating trajectories to possible target goals in advance, we anticipate an improvement in user response times.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics—*Workcell organization and planning*

General Terms

Algorithms, Measurement, Performance, Design, Experimentation, Verification

Keywords

Robotic arm, cache, assistance

1. INTRODUCTION

Time is a crucial factor in providing immediate assistance. For people with disabilities it is of great importance since they often depend on medical care from others. When immediate physical assistance is required, robotics technology can provide services and care in times of urgent need for medical attention.

Path planning can be computationally data intensive and time-consuming in robotics. A robot needs to access and process information from sensors to learn about an environment. The challenges of path planning involve finding an optimal path and handling precise manipulations. While processing a search through all the possible trajectories for a path planning algorithm, a user has to wait until the procedure is completed before receiving a response from the robot.

Our goal is to reduce the computation time for path planning of movements for a robotic arm. To provide this solution, we propose to use the techniques of caching frequent arm trajectories and creating a "roadmap" of robotic arm movements. We aim to increase productivity in processing user requests with a reduced expense of time and resources. We describe preliminary work that will support a way to perform caching of precomputed path plans and enable prefetching of path plans.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PETRA'11 May 25 - 27, 2011, Crete, Greece.
Copyright 2011 ACM 978-1-4503-0772-7/11/05 ...$10.00.

2. BACKGROUND

A number of researchers have devised variations of the classic A* algorithm for path planning. The State-Abstracted Hierarchical Task Network (SAHTN) algorithm recursively decomposes a high-level action task into lower-level actions until reaching primitive tasks that cannot be broken down any further[5]. The path plan of the primitive tasks are cached by the algorithm. By caching the sub–action tasks for a given task, computation times are reduced so that once a plan for a sub–action is computed it should not have to be re–computed the next time it is used.

Other work uses the framework of hierarchical heuristic search to solve shortest paths problems through a hierarchy of levels of abstractions [2]. While heuristic values are computed on-demand, Hierarchical IDA* (HIDA*) employs abstraction of the search space of the original problem so that it can be decomposed into smaller problems for faster computation. Once an optimal path is calculated, this algorithm also uses caching strategies for efficiency.

The concept of abstraction is also used in the Hierarchical Pathfinding Algorithm (HPA*) for path-finding in environments found in computer games [1]. Given a grid-based map with start and goal locations, the map is made abstract by dividing it into disjoint rectangular areas called clusters. The distances between nodes in a cluster and the optimal distances between clusters are precomputed and cached to minimize time.

In particular, we believe applying a method based on these ideas can improve path planning performance for robotics that are used specifically to assist persons with disabilities. The path plans involved are based on everyday tasks, such as fetching a glass cup, or lifting a spoon. As outlined in [4], these everyday tasks can be complicated due to the complexities that may occur in motion capture. Variations

in task performance is affected by different grasps, arm postures, orientations, and more [4].

By applying a technique that involves partial path pre-computations and caching, we believe this can reduce the complexity and increase the overall performance of path planning in this specific domain.

3. APPROACH

By analogy with a roadmap with major highways eventually leading to side streets, we also plan to break down the path to reach a target object by first translating the robotic arm towards the goal, rotating and translating to get as close as possible to the goal, and then finally grasping or manipulating the goal. We decompose tasks into subtasks. With this decomposition, as in previous work[5, 2, 1], we can apply the A* path planning algorithm a priori and cache optimal subpaths.

We consider a problem space where a potentially large number of variations for a particular task exist. When an individual performs an action such as 'Reaching', the tracked motion can vary based on a number of attributes, such as, the position of the individual[4]. In Figure 1 below, two different instances of arm movements can be seen to pick up an object from points unknown a priori. This task is achieved by the subactions of reaching and taking something. Thus, the planned path will contain subpaths in a region common to both instances which we define as a subpath district.

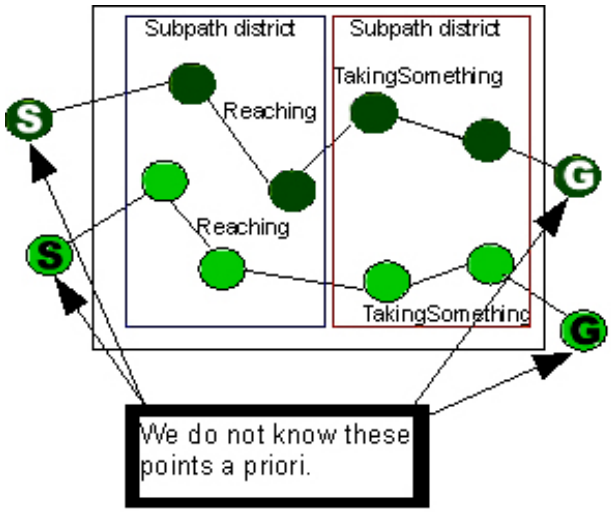

Figure 1: Variations of a Movement

Among these variations and based on the start and goal positions, an optimal path can be calculated in real-time. We propose a supervised method that involves a training phase which uses the A* algorithm. Variations of a particular task include a set of subtasks that could vary in their x,y start and end positions. During the training phase the optimal path is obtained using A* and is then cached. Start nodes and goal nodes are based on bounded areas as defined by the beginning and end positions of the group of subtasks that make up a task. This can be thought of as a neighborhood or district. The bounding neighborhood and optimal path are stored in a cache for real time application. During real time processing, when a path plan is sought for a particular action designated by a start position and a goal position, the cache is called upon. The start position and goal position are compared to the boundaries of the various neighborhoods. If the start and goal positions fall within a bounded neighborhood the optimal path is returned.

Our approach assumes that a start position and goal position are unique to a specified type of task and do not represent two different types of tasks. Our initial work assumes a static environment without obstacles.

4. METHODOLOGY

Our technique is a two-step process. The first procedure involves a training phase that produces the cache of optimized neighborhoods for a given set of points. These points form the boundaries for the neighborhood.

In the training phase, the following steps occur:

1. We create a grid and map our search space on the grid.
2. Given a repetition of a task X, we bound the area. We use these points to create a bounded area on the grid.
3. We run A* for permutations of points at the boundaries (representing start and goal nodes) of the repetitive paths of task X to find the optimal path for each set of boundary points.
4. As seen in Figure 2, the determined boundaries of each specific subtask found in this phase are used to recognize efficient start and goal nodes along the edges of bounded subpath districts with the optimal path between those points that is stored in the cache.

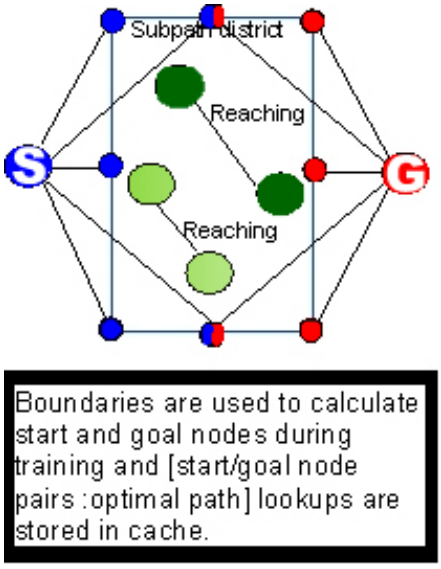

Figure 2: Bounded Task

During run time we perform the following steps:

1. Given the start and goal nodes, we map these points on the grid.
2. We compare our start and goal nodes on the grid with predefined bounded areas represented in the cache.

3. If the start and goal points fall within the dimensions of a cached bounded area, then the cached optimal path is returned.

4. If one or both points do not fall within the dimensions of the bounded area the following has to occur:
   Of all created boundaries, we find the optimal path from the start to the boundary and from the boundary to the goal. We do not need to apply A* to the full set of subtasks or the neighborhood that we defined because we know this a priori. We need simply the optimal path from the start to the fringe of the boundary and the fringe of the boundary to the goal node. A similar method was shown to be successful when implemented by [1].

We modeled a fully associative cache and the A* algorithm mapped on a grid by extending the aimajava package [3]. The cache is preloaded with frequently used arm movements from the training data. Simulated execution time is computed with and without cache to get the shortest paths from start positions and goal positions with different sets of data for comparison testing.

In the practical sense, when a processor wants to read or write to main memory, it will first check if a copy of that data exists in the cache. If it is found in the cache, it is a cache hit so the processor can access and retrieve the stored data. As a result, time is saved for a path planning computation because it is in the cache. Otherwise, in the event of a cache miss, a processor must retrieve data in a lower level of memory causing latency. By reducing cache misses, we can decrease the time to compute path planning and increase the throughput of requests.

4.1 Experimental Data

We used the Technische Universitat Munchen (TUM) Kitchen Data Set for testing our approach [4]. The TUM Kitchen Data Set is composed of 20 sessions of motion capture data involving tasks related to setting a kitchen table from the TUM's "MeMoMan" motion tracking system. We prepared data by observing captured video to find rates of tasks and represented hand motion data in XML format.

The tasks of interest in our study are the movements of lowering the arm and reaching for an object with and without additional subtasks. In our experiment, we simulated the time spent for path planning of these specified tasks. We test our approach by running a set of tests to compute the execution times for comparison.

5. RESULTS

Our preliminary results were based on running the tasks of lowering, reaching, lowering with additional tasks defined as lowering mix, and reaching with additional tasks defined as reaching mix. The simulated execution time was computed for each of the four types of tasks in three different time instances. All of the tasks in each time instance experienced an improvement of path planning with a cache.

To evaluate performance, we conducted a K-fold cross validation where K = 2 for our experiments. The data set is evenly partitioned with the first half as the training set and the second half as the testing set in the first fold. Similarly, in the second fold the first half becomes the testing set and second half becomes the training set. The results of both the first and second folds are averaged to acquire an estimate of performance improvement.

For time instance 1, the results of performance are shown in Figure 3. The task of lowering improved with an average speedup of 156%. Likewise, the task of reaching had an enhanced average speedup of 86%. In the same degree, the mix task of lowering with additional subtasks gained an average speedup of 302%. Finally, the mix task of reaching with subtasks improved with a average speedup of 144%.

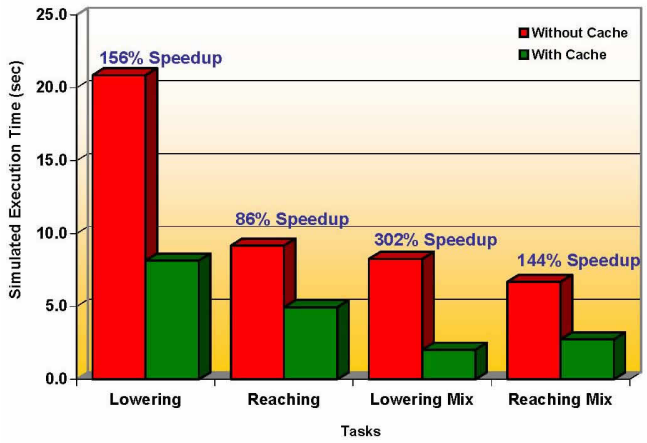

Figure 3: Results at Time Instance 1

Figure 4 displays the outcome of performance for time instance 2. The lowering task enhanced with an average speedup of 183% and the reaching task had an average speedup of 26%. Also, the mix of lowering with additional subtasks improved with an average speedup of 20%. Along with the mix of reaching with an average speedup of 50%.

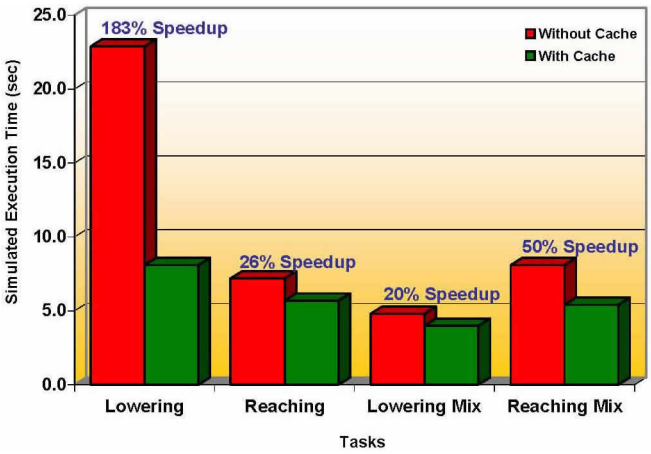

Figure 4: Results at Time Instance 2

As seen in Figure 5 for time instance 3, the lowering task improved with an average speedup of 305% and the reaching task had an average speedup of 37%. In the same manner, the mix of lowering with subtasks improved with an average speedup of 70%. Lastly, the mix of reaching with subtasks improved with an average speedup of 243%.

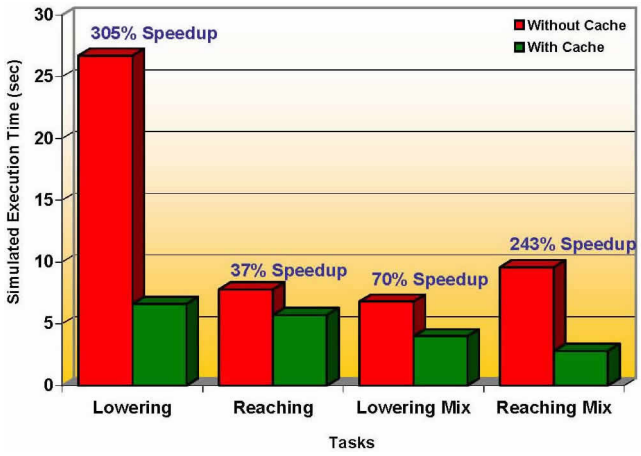

Figure 5: Results at Time Instance 3

We predicted that this approach would be less optimal when compared with using A* in real time to find the most optimal path. Our tests showed that 20% of the time we were less optimal than using A* in real time. However, we did not implement any path smoothing techniques which did increase optimality in previous work [1]. We also worked with a limited set of training data and a minimum amount of boundary points. Future experiments will consider more boundary points and larger training sets.

In the process of running each test, we collected samples of memory usage for the specific type of action, with and without cache. Results showed that A* with caching required on average 80% more memory than running A* in real time without any caching. We sampled throughout each test run and averaged the overall memory usage for tests that used caching and tests that did not use caching.

We wished to compare our approach with [1] since software is readily available to use and since they showed both performance improvements and close to optimal results. We ran two experiments, the task of lowering and the task of reaching, and discovered that on average they were 8% faster than us. We find this encouraging since our work is preliminary.

6. FUTURE WORK

We described preliminary work and intend to further our research and techniques based on past work for computer games. We intend to work with data sets that will produce a larger search space and that are rich with variation for everyday tasks. Our thoughts are that the larger and more complex the search space, the more benefit we will see from our approach. We plan to also consider search spaces that involve obstacles, this complicates the path plan further but is a common occurrence that needs to be considered. We also plan to further our work in relation to prefetching. Prefetching will involve an element of prediction that would prefetch future plans based on past behaviors and the current state of the robot.

7. CONCLUSION

We show how mapping repetitive tasks to a grid and finding an optimal path that can be preloaded for future executions reduces the complexity and time of an A* path planning implementation. The estimation of points during training can impact the optimally of our solution and needs further attention but its performance impact can be significant for large repetitive search spaces. Our caching approach did improve performance using our tests as can be seen by our hit rates. Since we have a speed up with caching for a single task, it implies that latency will be deceased for the path planning computation for a task.

8. REFERENCES

[1] Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1:7–28, 2004.

[2] R. Holte, J. Grajkowski, and B. Tanner. Hierarchical heuristic search revisited. 2005.

[3] R. Mohan. Java implementation of algorithms from norvig and russell's 'artificial intelligence - a modern approach 3rd edition.', 2010.

[4] Moritz Tenorth, Jan Bandouch, and Michael Beetz. The TUM Kitchen Data Set of Everyday Manipulation Activities for Motion Tracking and Action Recognition. In *IEEE Int. Workshop on Tracking Humans for the Evaluation of their Motion in Image Sequences (THEMIS). In conjunction with ICCV2009*, 2009.

[5] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. In *International Conference on Automated Planning and Scheduling*, 2010.