The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Navigation Among Movable Obstacles in Unknown Environments

Hai-Ning Wu Martin Levihn Mike Stilman

***Abstract*— This paper explores the Navigation Among Movable Obstacles (NAMO) problem in an unknown environment. We consider the realistic scenario in which the robot has to navigate to a goal position in an unknown environment consisting of static and movable objects. The robot may move objects if the goal can not be reached otherwise or if moving the object may significantly shorten the path to the goal. We consider real situations in which the robot only has limited sensing information and where the action selection can therefore only be based on partial knowledge learned from the environment at that point. This paper introduces an algorithm that significantly reduces the necessary calculations to accomplish this task compared to a direct approach. We present an efficient implementation for the case of planar, axis-aligned environments and report experimental results on challenging scenarios with more than 50 objects.**

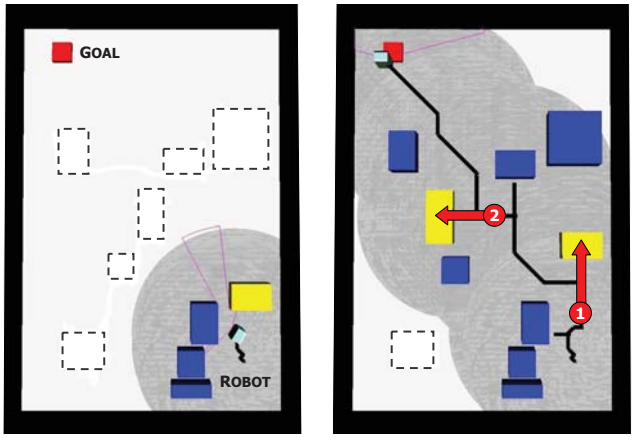
I. Introduction

Robots would be much more useful if they could move obstacles out of the way. Navigation Among Movable Obstacles (NAMO) is an important problem in motion planning because it gives mobile robots the ability to reason about the environment and choose to manipulate obstacles [14]. Robots that solve NAMO will accomplish tasks that are otherwise difficult or impossible. They will operate in cluttered human environments and strive towards human-level navigation. In order to accomplish this goal, motion planning must overcome a number of theoretical and practical challenges.

In this paper, we explore the NAMO problem in practical scenarios where the robot attempts to reach a fixed goal position in a *reconfigurable but unknown environment*. Starting with no knowledge about the environment, the robot uses limited sensor information to locally detect objects and incrementally build and manipulate a world model. The robot may move objects if the goal cannot be reached or if moving the object may significantly shorten the path to the goal. An illustrative example is shown in Fig. 1(a) where the robot is forced to move objects to navigate towards an otherwise unreachable goal. With only local and incomplete information (such as the movability of objects), the robot must make a decision based on partial knowledge acquired so far and gradually improve its world model as it navigates towards the goal (e.g., in Fig. 1(b)).

NAMO in an unknown world poses a fundamental challenge in planning. Potentially, all possible actions have to be reevaluated whenever new information is perceived. However, recomputing the cost of all possible actions for each environment change is infeasible for realistic domains. We investigate a computationally feasible strategy that accounts for environmental changes.

The authors are affiliated with the Center for Robotics and Intelligent Machines (RIM) at the Georgia Institute of Technology, Atlanta, Georgia 30332, USA. Emails: hwu43@gatech.edu, levihn@gatech.edu, mstilman@cc.gatech.edu



(a) Initial State (b) Final Execution

Fig. 1. Successful NAMO with partial information. Yellow (light) objects are movable. Blue (dark) ones are not. (a) Dashed lines represent unknown objects. (b) The evolution of the robots internal map.

The main contribution of this paper is the introduction of a novel algorithm that solves the NAMO problem in an unknown world and significantly reduces the necessary calculations. The proposed algorithm identifies cases where new information does not affect previous calculations. Instead of reevaluating all actions when new obstacles are detected, the algorithm only performs additional computation when newly detected information conflicts with the optimality of the existing plan. We verify the algorithm in dynamic simulation where the robot controller is guided by our implementation in a planar, cell-decomposed, and axis-aligned environment. We demonstrate the performance of the algorithm in challenging scenarios with more than 50 randomly placed objects.

This paper is organized as follows: Section II gives a review of related work both in NAMO and planning under uncertain environments. Section III presents the proposed algorithm and compares it to a naive baseline algorithm. Section IV details the experimental results in a dynamic simulation environment and characterizes the performance of the algorithms. Section V discusses the limitations of the algorithm as well as the challenges introduced by the partial knowledge itself. Finally, Section VI gives concluding remarks and presents directions for future work.

II. Related Work

Wilfong [17] first proved that motion planning among movable obstacles is NP-hard. Demaine [2] further proved that even the simplified version of this problem, in which only unit square obstacles are considered, is also NP-hard.

Chen [1] designed the first planner that handled multiple movable objects and a navigation goal. The heuristic planner first generated a series of subgoals and solved the subgoals separately by a local planner. Chen's planner failed to solve problems in which the order of object manipulations decides the solution. Stilman [14] presented a planner that solved a subclass of NAMO problems named LP_1 in which disconnected components of free-space can be connected independently by moving a single obstacle, reducing the search space of NAMO by considering the difficulty of the navigation task rather than the dimensionality of the space. By formulating a problem in LP_1 into a graph structure, a resolution complete solution can be generated using a heuristic planner. The planner was able to solve the difficult problems presented in [1] and was successfully implemented on the humanoid robot HRP-2 [16]. Beyond LP_1, Stilman further solved the domain where maximally k objects must be moved to connect two disjoint components and each object needs to only be moved once [15]. Li [7] constructed an autonomous system which combined moving objects and leaping over obstacles with other high-level behaviors using a unified planning strategy. However, all these methods solved NAMO given complete knowledge about the environment. Furthermore, instead of aiming for optimality, heuristics were used in order to find a resolution complete solution.

LaValle [6] presented a game-theoretic framework for robot motion planning in uncertain environments. Pirjanian [10] introduced many approaches to formulate the motion planning problem as an action selection problem and also presented an implementation of Multiple Objective Action Selection for robot navigation [11]. By defining objective functions for different subgoals, the Pareto optimality was calculated to find a "good enough" action for the current state. Although they introduced promising ways on decision making under uncertainty, it remains difficult to model the problem within variant configurations and further guarantee the optimality of each action.

The D* algorithm [12] [13] incrementally searched paths in partially known environments by propagating the cost evaluated from the previous state to the new state. Thus, repeated replanning can be avoided without losing optimality. Koenig [5] introduced a rather less complicated algorithm, D* Lite, which only recomputes costs relevant to new information. However, environments with movable obstacles would require a significant reformulation of D* algorithms. In contrast, we propose a set of clear and simple improvements from the base case of search.

Koenig [4] also established a series of techniques for goal-directed acting in the presence of incomplete information. This work suggested applying agent-centered search methods to minimize the cost of planning as well as plan execution. Koenig also used partially observable Markov decision process (POMDP) to enhance the reliability of planning with incomplete information. POMDPs maintain and update a probabilistic model to minimize the cost of plan execution. Yet, such work was restricted to planning solutions that do not change the environment.

The Bug algorithm presented by Lumelski [8] approached the path planning problem for sensor-based robots in unknown environments. The Bug algorithm provided reasonable paths to the global goal based on local information and the lower bound of the path can be guaranteed. Variants of the Bug algorithm [9] utilized different optimizations strategies such as reducing the length of the path or the information needed. However, the Bug family did not handle reconfiguration of environments and sacrificed optimality for completeness and planning efficiency. Solution is inevitably much higher.

III. Algorithm

We consider the navigation scenario with a nonholonomic robot R in a two-dimensional workspace containing movable and static objects. The robot has to find, with respect to his current world knowledge, a collision-free path from the given starting position R_{init} to the given goal position R_{goal}. The robot is given its starting and goal position in world coordinates, but no prior knowledge about the position, size, or movability of the objects is provided. The robot gains information about the position and size of objects through the use of a laser range finder, but movability can only be determined by interacting with the object. We restricted the possible interactions with objects to axis-aligned pushes. The environment is discretized in a $N \times N$ grid and the objects as well as the robot are modeled as rectangles.

The robot has an internal map of the environment which is updated upon the detection of new obstacles and new information about previously known objects, such as updated size and movability. Every unknown cell in the map is assumed to be free-space and every object is considered movable unless a failed push action has been performed. In addition, we are not considering the possibility of moving multiple objects in order to create a new path. The immediate plan is limited to moving at most one object.

For clarity, we will not explicitly mention static objects. All cells that correspond to a detected static object are marked as blocking robot and object motion in future plans.

A. Baseline

The direct solution to the NAMO problem in unknown environments is to calculate plans for all possible actions on all known objects once any change in the environment is detected. This approach is outlined in Algorithms 1 and 2.

The algorithm is initialized by calculating a plan with an A* search from R_{Init} to R_{Goal} using the Euclidean distance as an admissible heuristic. If no new observations are made, this plan remains unchanged until the goal is reached. However, if a new observation is made, then for all known objects (line 8) all possible push actions (line 9 and 10) are evaluated. This is done in Algorithm 2 where a plan is constructed that consists of moving to the object position, pushing the object, and moving from the final object position to the goal. The procedure is visualized in Fig. 2 where the steps are labelled c_1, c_2 and c_3, respectively.

Algorithm 1 BASELINE(R_{Init}, R_{Goal})

1: $R \Leftarrow R_{Init}$;
2: $\mathcal{O} \Leftarrow \emptyset$; {set of objects}
3: $p_{opt} \Leftarrow A^*(R_{Init}, R_{Goal})$;
4: **while** $R \neq R_{Goal}$ **do**
5: $\quad \mathcal{O}_{new} \Leftarrow$GET-NEW-INFORMATION();
6: $\quad$ **if** $\mathcal{O}_{new} \neq \emptyset$ **then**
7: $\quad\quad \mathcal{O} = \mathcal{O} \cup \mathcal{O}_{new}$;
8: $\quad\quad$ **for** each $o \in \mathcal{O}$ **do**
9: $\quad\quad\quad$ **for** each possible push direction d on o **do**
10: $\quad\quad\quad\quad p \Leftarrow$ EVALUATE-ACTION(o,d);
11: $\quad\quad\quad\quad$ **if** $p.cost < p_{opt}.cost$ **then**
12: $\quad\quad\quad\quad\quad p_{opt} = p$;
13: $\quad\quad\quad\quad$ **end if**
14: $\quad\quad\quad$ **end for**
15: $\quad\quad$ **end for**
16: $\quad$ **end if**
17: $\quad R \Leftarrow$ Next step in p_{opt};
18: **end while**

Algorithm 2 EVALUATE-ACTION(o, d)

1: $p_{o,d} \Leftarrow \emptyset$
2: $c_1 = |A^*(R, o.init)|$;
3: $o.position = o.init$;
4: **while** push on o in d possible **do**
5: $\quad o.postion = o.postion + one\_push\_in\_d$;
6: $\quad c_2 = (o.postion - o.init)$;
7: $\quad c_3 = |A^*(o.position, R_{Goal})|$;
8: $\quad p = c_1 + c_2 + c_3$;
9: $\quad p.cost = c_1 * moveCost + c_2 * pushCost + c_3 * moveCost$;
10: $\quad P_{o,d} \Leftarrow P_{o,d} \cup \{p\}$;
11: **end while**
12: return $p \in P_{o,d}$ with min $p.cost$;

B. Optimized algorithm

In order to gain better scalability for bigger maps with more objects, we design three techniques that reduce the necessary calculations in the baseline algorithm given above.

1) Recalculation triggering: First, we do not automatically recalculate plans upon the detection of new objects or updated object information. Recalculation is required only if the current plan becomes invalid due to expected collisions with newly detected obstacle data. The calculation can be postponed since each plan is computed with assumed free space in unknown terrain. An obstacle can only increase the cost of traversing the space that it covers. If the current path is not blocked, replanning is unnecessary. This is shown in line 6 of Algorithm 3 and visualized in Fig. 3. Recalculations are not performed prior to the detection of object 3. If the plan is blocked, all newly detected objects are evaluated for possible displacements in line 8-10 of Algorithm 3. Fig. 3 shows a case where recalculation is necessary. If pushing object 2 is not considered after detection then the original path is blocked and the goal is unreachable.

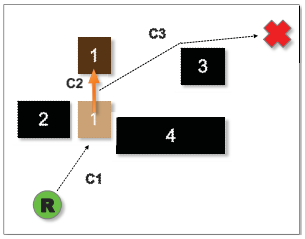
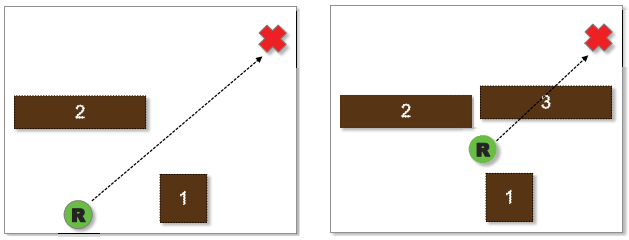


Fig. 2. Visualization of the steps for a plan involving pushing object 1



(a) Time t (b) Time t_x

Fig. 3. Detection of new objects forces recalculation of previously ignored objects. (a) No recalculations are necessary. (b) All objects must be checked.

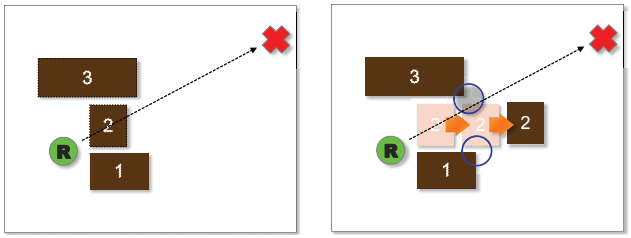
2) Limit A calls:* In order to reduce unnecessary recalculations, we limit the number of push action evaluations for an object. The limit is set by determining an upper bound for the cost of pushes. In line 5 in Algorithm 4 the upper bound is determined by the cost of avoiding objects. If the cost of only pushing an object already exceeds the cost of avoiding the object, further pushes are not considered. This is due to the fact that the plan consists out of the three parts c_1, c_2 and c_3 (see Fig. 2 and Algorithm 4 line 10) which all yield positive cost. Consequently, if c_2 already exceeds the cost of just avoiding the object, any plan involving further pushes than c_2 cannot yield a lower cost. In addition, plans are only calculated for push actions that create a new opening in the map. This can be seen in line 7 Algorithm 4 and is visualized in Fig. 4 where the evaluation of object 2 is reduced to only two plan calculations.

3) Reduce candidate objects: We do not recalculate plans involving all previously considered objects. We consider only those objects where a calculation appears promising. This is done by retaining a sorted list with lower bounds for previously computed plans. The list is sorted according to $minCost$ of a plan, representing partial plan cost. $minCost$ is set at the time of plan calculation in line 12 of Algorithm 4 and represents an underestimate for the true cost associated with the plan. This list is traversed and updated plans (with the current environment information) for the elements in the list are computed. Traversal can be terminated once a plan with lower cost than the under-estimated cost for the next element in the list is found. Notice that objects that were detected by the robot earlier are typically farther from the goal. Hence they have have high c_3 value and are not reevaluated. This method is presented in line 14-19 of Algorithm 3.

The only special case in our algorithm occurs if no collision-free path avoiding the object could be found for the optimization step of limited A* calls. In this scenario, the

Algorithm 3 OPTIMIZED(R_{Init}, R_{Goal})

$R \Leftarrow R_{Init}$;
$P_{sort} \Leftarrow \emptyset$; {list of plans, sorted ascending by $minCost$}
$p_{opt} \Leftarrow A^*(R_{Init}, R_{Goal})$;
while $R \neq R_{Goal}$ **do**
  $\mathcal{O}_{new} \Leftarrow \mathcal{O}_{new} \cup$ GET-NEW-INFORMATION();
  if $p_{opt} \cap \mathcal{O}_{new} \neq \emptyset$ **then**
    $p_{opt} \Leftarrow A^*(R, R_{Goal})$;
    for each $o \in \mathcal{O}_{new}$ **do**
      for each possible push direction d on o **do**
        P_{sort}.insert(OPT-EVALUATE-ACTION(o,d, p_{opt}));
      end for
    end for
    $p_{next} = P_{sort}[0]$;
    while $p_{opt}.cost \geq p_{next}.minCost$ **do**
      p=OPT-EVALUATE-ACTION($p_{next}.o$,$p_{next}.d$, P_{opt});
      if $p.cost < p_{opt}.cost$ **then**
        $p_{opt} = p$;
      end if
      $p_{next} = P_{sort}$.getNext();
    end while
    $\mathcal{O}_{new} \Leftarrow \emptyset$;
  end if
  $R \Leftarrow$ Next step in p_{opt};
end while



(a) Internal map (b) Evaluation of Object 2

Fig. 4. Plans are only calculated upon new path openings and when the maximum push distance is limited by the cost of avoiding the object. In (b) only two A* calls, marked with blue circles, are necessary when evaluating pushing object 2.

upper bound for the costs of a push action in the optimization step is infinity. We observe that we can detect which objects are not affecting our ability to find the goal. For example, this can be done by ignoring one object at a time and then checking if a path to the goal without this object can be found. If no path can be found then this object is considered *non-blocking*, however if a path can be found without that object then it is a *blocking* object. We can now iterate over the list of *blocking* objects and in each iteration, we increase the number of push actions evaluated on each object. Upon the first detection of a possible opening, we use this value as an upper bound for evaluating push actions for each object. This special case is handled in our algorithm but not shown in pseudocode.

Algorithm 4 OPT-EVALUATE-ACTION(o, d, p_{opt})

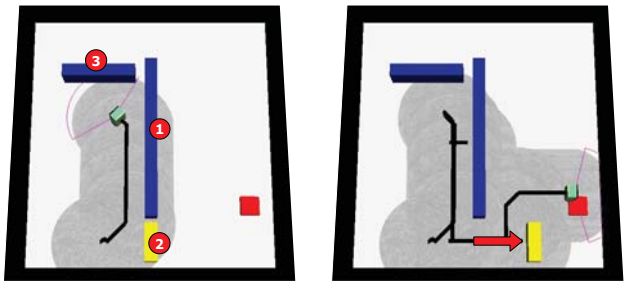
$P_{o,d} \Leftarrow \emptyset$;
$c_1 = |A^*(R, o.init)|$;
$c_2 = 0$;
$o.position = o.init$;
while push on o in d possible AND $c_2 * pushCost < p_{opt}.cost$ **do**
  $o.postion = o.postion + one\_push\_in\_d$;
  if push created new opening **then**
    $c_2 = (o.position - o.init)$;
    $c_3 = |A^*(o.position, R_{Goal})|$;
    $p = c_1 + c_2 + c_3$;
    $p.cost = c_1 * moveCost + c_2 * pushCost + c_3 * moveCost$;
    $p.minCost = c_2 * pushCost + c_3 * moveCost$;
    $p.o = o$;
    $p.d = d$;
    $P_{o,d} \Leftarrow P_{o,d} \cup \{p\}$;
  end if
end while
return $p \in P_{o,d}$ with min $p.cost$;

All the techniques above reduce the necessary calculations for finding a path with low cost to the goal. Our experiments were performed with both the baseline as well as the optimized algorithm. We found no difference in the final plans calculated by the algorithms. In the following section we present examples and statistics for results.

IV. Experiments and Discussion

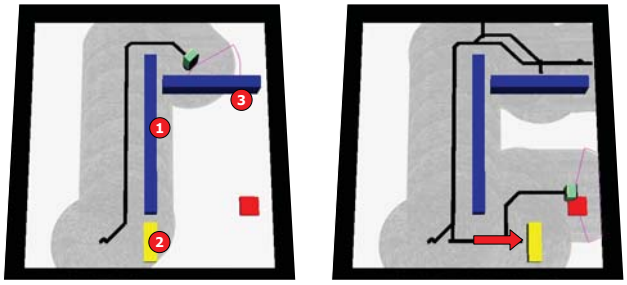
We evaluated our algorithms in dynamic simulation using srLib [3]. First, we give four representative domains with 3 to 50 obstacles and explain algorithm operation. Second, we collect statistics from 10 randomized experiments with 10-20 obstacles and compare performance. While the baseline and optimized algorithms generated identical robot decisions, the computation times were significantly lower for the latter.

Fig. 1 is a typical scenario that demonstrates how the robot replans given new information. Fig. 5 and Fig. 6 are two interesting examples showing that small differences on the map can significantly affect decisions. In Fig. 5(a) the robot initially thinks that there exists a less expensive path to the goal by going around object 1 ($o1$). After detecting object 3 ($o3$) the robot continues to circumnavigate $o3$ until its finds that the cost of returning and pushing $o1$ is lower in comparison with the length of a path around $o3$. Failing to push the unmovable $o1$, the cost of returning and pushing $o2$ are still less than bypassing $o1$. The robot finally reaches the goal by pushing $o2$. In Fig. 6(a) the robot does not know that all the paths to the goal that do not push $o2$ are blocked. It proceeds around $o1$ until it detects $o3$ close to the goal. At this point, the cost of returning to push $o2$ is very high and the robot explores a greater region of space. The robot makes more attempts to push before it is confident that the estimated value of returning to push $o2$ is less than the cost of further exploration.



(a) Initial (b) Goal

Fig. 5. (a) The robot sees object 3 at early stage. (b) The robot goes back to push when the free path becomes expensive due to new obstacles.



(a) Initial (b) Goal

Fig. 6. (a) The robot moves around object 1 initially because it expects free path before detecting object 3. (b) The robot realizes all the paths are blocked so it returns and tries pushing different objects.

In order to evaluate improvements against the baseline algorithm by each of the three optimizations we use three metrics. In the following list, for each optimization, the first bullet is the cost of the optimized algorithm and the later one is the cost of the baseline. We compare these two values to measure the improvement due to each optimization.

1) Optimization 1 - Recalculation triggering
   - Number of steps on which the path is blocked
   - Number of steps during the entire navigation
2) Optimization 2 - Limited A* calls
   - Number of A* calls
   - Number of all possible push steps
3) Optimization 3 - Reduce candidate objects
   - Number of actions after filtering by $minCost$
   - Number of all candidate actions

We conducted experiments on 10 solvable configurations with random placements of random numbers of obstacles, ranging from 10 to 20. The result summarized in Table I shows that most computation can be saved since it has no influence on the robot's decision. The performance of Optimization 1 is highly dependent on the density of obstacles and the robot's trajectory. If the robot's plan is always blocked by new obstacles, recalculation is triggered often. Optimization 2 demonstrates that only a few candidate push steps are potential solutions for a given action. Hence, many A* calls can be avoided. Finally, Optimization 3 shows that the set of candidate actions, representing candidate objects can be reduced by 80% by considering overall plan cost.

TABLE I
COMPUTATIONAL ADVANTAGE FOR EACH OPTIMIZATION

Optimizations	Triggering	Limit Calls	Reduce #Cand
Avg Optimized Cost	52.4	473.7	395.2
Avg Base Cost	224.1	19588.3	2033.6
Min Improvement	67%	97%	75%
Max Improvement	85%	98%	88%
Avg Improvement	76%	98%	81%

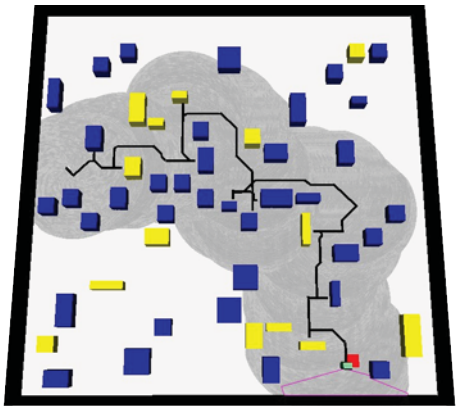


Fig. 7. Example with 50+ objects.

Fig. 7 and Fig. 8 demonstrate that more information does not necessarily imply that equally more actions need to be evaluated. In Fig. 8, the optimized curve grows slowly because as new objects are detected, some old objects can be ignored. Similar to [5], our optimization only recomputes actions relevant to surrounding areas, or actions that will potentially reduce the cost. Thus, as the robot explores more areas, the difference between the two curves in Fig. 8 becomes larger.

Our algorithm models human-like learning behaviors when faced with obstacles with a logical process for machine intelligence. When searching for solutions with partial information, humans intuitively choose the near solutions rather than far ones. Once an action is confirmed useless, there is no need to reevaluate it repeatedly as long as the action will only cost more with more information. Also, in our model, the robot has to interact with objects to learn their movability. After failing to move the unmovable objects, the robot considers other actions. This process resembles natural human behavior.

V. CHALLENGES

The domain of NAMO with incomplete information has unsolvable problems. For example, in Fig. 9(a), the robot first pushes object 2 and intends to move directly to the goal. Unfortunately, in Fig. 9(b), the robot detects object 3 and tries to push object 3. Since object 3 is unmovable, the robot can never reach the goal, even with more information. However, if the robot knows object 3 beforehand, it will push object 1 first to avoid blocking itself. Given only partial information, the robot cannot avoid all the negative effects resulted from reconfigurations. Thus, local solutions will not always solve the global problem.

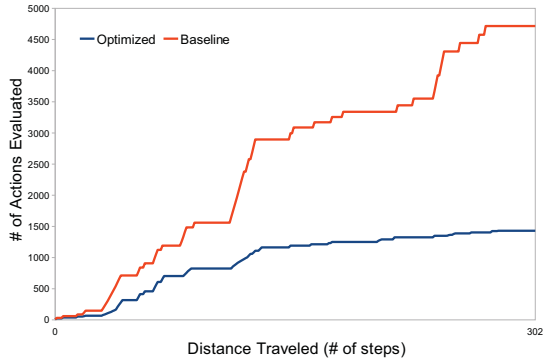
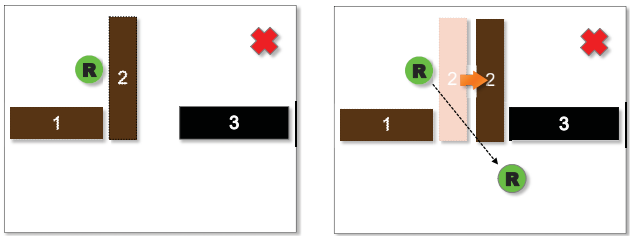


Fig. 8. Growth of action evaluations with more information in Fig. 7.



(a) Initial (b) Failed

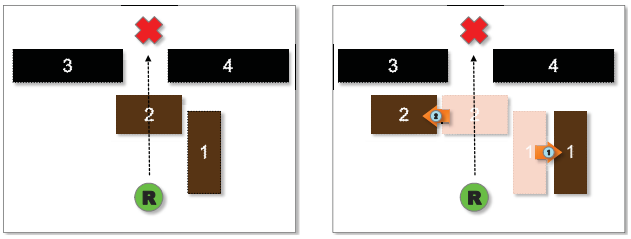
Fig. 9. Objects 1 and 2 are movable. Object 3 is static. (a) The robot pushes object 2 and creates a new path. (b) The new obstacle isolates the robot outside the region that contains the goal.

Our algorithm can not solve problems like Fig. 10, which requires pushing two obstacles to create new paths. It will first push objects 3 and 4 and realize that they are static. Simply pushing either object 1 or 2 does not create any new openings. However, if the robot considers manipulation of multiple objects within a single action, it is possible to solve the problem as shown in Fig. 10(b).

In summary, our approach faces two major challenges. First, given the premise that partial information is complete, the best solution for the currently known environment does not necessarily solve the global problem. Reconfigurations can even block the solution. Second, our algorithm requires an appropriate metric to evaluate the cost of reconfigurations. To address them, future directions include: 1) Find more solid evaluations for moving and pushing cost; 2) Allow manipulations on multiple objects or multiple actions on a single object to create new paths; 3) Use current knowledge with new information to predict effects on the overall environment instead of always assuming no obstacles in unknown area.

VI. Conclusion

In this paper, we explored NAMO in unknown environments. We proposed an algorithm that only reevaluates actions if the current plan is intersected by a newly detected object or becomes invalid due to wrong assumptions such as the movability of an object. The algorithm also only evaluates objects that have promise to yield a better solution by saving the values of previously calculated plans. The actual calculation of a push action is optimized by setting an upper bound on the number of simulated pushes and only performing A* if a new opening in the map has appeared.



(a) Initial (b) Solution

Fig. 10. Objects 1 and 2 are movable. Objects 3 and 4 are static. (a) An example that can not be solved by our algorithm. (b) Object 1 must be moved first in order to create space to push object 2.

We showed that the algorithm is not only capable of finding a path in an unknown environment with movable and static objects based on partial knowledge but also reduces the number of necessary calculations. Furthermore, we were not able to detect any difference in the final plan to the baseline approach explained in section III-A. However, our approach is also limited by the naive assumption about the environment. Future work includes more precise prediction on environments and better evaluation of actions.

References

[1] P. Chen and Y. Hwang. Practical path planning among movable obstacles. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 444–449, 1991.

[2] E. Demaine, J. O'Rourke, and M. L. Demaine. Pushpush and push-1 are np-hard in 2d. In *In Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 211–219, 2000.

[3] Robotics Lab in Seoul National University. Snu robotics library. `http://r-station.co.kr/forum/`.

[4] S. Koenig. *Goal-Directed Action with Incomplete Information*. PhD thesis, 1997.

[5] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *in Proceedings of the International Conference on Robotics and Automation*, pages 968–975, 2002.

[6] S. LaValle. Robot motion planning: A game-theoretic foundation. *Algorithmica*, 26(3-4):430–465, 2000.

[7] Y. Li and T. Li. A unified approach to planning versatile motions for an autonomous digital actor. *JACIII*, 12(3):277–283, 2008.

[8] V. Lumelski and A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control*, AC-31(11):1057–1063, 1986.

[9] J. Ng and T. Bräunl. Performance comparison of bug navigation algorithms. *J. Intell. Robotics Syst.*, 50(1):73–84, 2007.

[10] P. Pirjanian. Behavior coordination mechanisms – state-of-the-art, 1999.

[11] P. Pirjanian. The notion of optimality in behavior-based robotics, 1999.

[12] A. Stentz. Optimal and efficient path planning for partially-known environments. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3310–3317, 1994.

[13] A. Stentz. The focussed d* algorithm for real-time replanning. In *In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.

[14] M. Stilman and J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Proceedings of the 2004 IEEE International Conference on Humanoid Robotics (Humanoids'04)*, volume 1, pages 322 – 341, December 2004.

[15] M. Stilman and J. Kuffner. Planning among movable obstacles with artificial constraints. In *WAFR*, pages 119–135, 2006.

[16] M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner. Planning and executing navigation among movable obstacles. In *IEEE/RSJ Int. Conf. On Intelligent Robots and Systems (IROS 06)*, pages 820 – 826, October 2006.

[17] G. Wilfong. Motion planning in the presence of movable obstacles. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 279–288, New York, NY, USA, 1988. ACM.