Computational Tradeoffs of Search Methods for Minimum Constraint Removal Paths

Athanasios Krontiris and **Kostas E. Bekris**
Department of Computer Science, Rutgers University
New Brunswick, New Jersey 08901

Abstract

The typical objective of path planning is to find the shortest feasible path. Many times, however, there may be no solution given the existence of constraints, such as obstacles. In these cases, the minimum constraint removal problem asks for the minimum set of constraints that need to be removed from the state space to find a solution. Unfortunately, minimum constraint removal paths do not exhibit dynamic programming properties, i.e., subsets of optimum solutions are not necessarily optimal. Thus, searching for such solutions is computationally expensive. This leads to approximate methods, which balance the cost of computing a solution and its quality. This work investigates alternatives in this context and evaluates their performance in terms of such tradeoffs. Solutions that follow a bounded-length approach, i.e., searching for paths up to a certain length, seem to provide a good balance between minimizing constraints, computational cost and path length.

Introduction

In path planning problems, there are frequently constraints that cause no solution path to exist but it may possible to apply changes in the state space so that a solution is possible. In particular, consider robot manipulation (Cohen, Chitta, and Likhachev 2014; Krontiris and Bekris 2015) as in Fig. 1, where a robot needs to transfer an object from the left side of the shelf to the right one. This problem instance has no collision-free solution given the placement of objects. The manipulator, however, can change the workspace so as to solve its task. Thus, a subproblem arises: what is the minimum number of obstacles that must be evacuated to make the problem feasible. This challenge is referred to as the *minimum constraint removal* (MCR) problem (Hauser 2013a; Erickson and LaValle 2013) and is related to navigation among movable obstacles (NAMO), (Stilman and Kuffner 2004; 2006; Wilfong 1988) and manipulation under clutter (Dogar and Srinivasa 2011). MCR (Hauser 2013a) does not capture negative interactions between obstacles, which needs to be dealt with in the context of rearranging multiple objects (Krontiris et al. 2014; Krontiris and Bekris 2015).

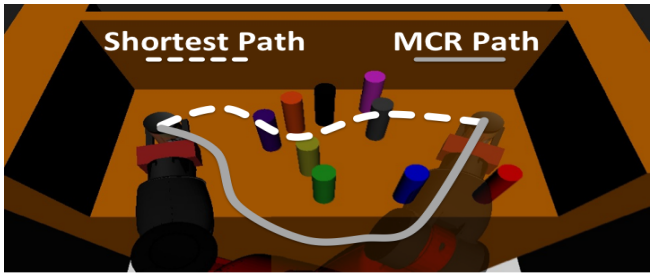

Figure 1: A minimum constraint removal challenge. Dashed line: The shortest path that ignores obstacles is computed quickly but returns many constraints. Solid line: It is more expensive to compute the minimum constraint path.

NAMO challenges relate to Sokoban, for which search methods and abstractions have been developed (Botea, Muller, and Schaffer 2002; Pereira, Ritt, and Buriol 2013).

There are also other domains that can benefit from MCR solutions. It can provide humanly understood explanations why a problem is not solvable (Gobelbecker et al. 2010). In multi-agent path finding (Luna and Bekris 2011; Krontiris, Luna, and Bekris 2013; Sharon et al. 2015), the MCR path can provide the minimum set of agents that must evacuate a path. Block sliding puzzles can involve MCR subproblems (Hearn and Demaine 2005). But, MCR is harder than the shortest path problem, since a subset of an optimal solution is not necessarily optimal, i.e., it does not satisfy dynamic programming properties. Furthermore, MCR is generalizing the task of determining path non-existence, which is known to be hard (Zhang, Kim, and Manocha 2008; McCarthy, Bretl, and Hutchinson 2012).

This paper first reviews existing solutions for the MCR problem (Hauser 2013b): (i) a computationally infeasible approach that searches for all paths, (ii) a faster greedy strategy, which is incomplete, (iii) an exact approach, which takes advantage of the problem structure to properly prune the state-space but is still slow. Then this paper proposes bounded path length search, which bounds the length of paths considered relatively to the shortest path in a constraint-free workspace. The experiments show that with this method there is significant computational benefit, while the constraints found are close to those returned by the exact approach. This also suggests an anytime solution, where the bound on path length incrementally increases.

Problem Setup and Notation

In cases where a path is blocked by constraints, the minimum constraint removal problem asks for the minimum set of constraints, which if removed, a feasible solution arises. A graph-based abstraction of the challenge is the following.

Minimum Constraint Removal Path: Consider a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where for each edge $e \in \mathcal{E}$ a set of constraints c_e is defined. Then, given a start-target vertex pair $s, t \in \mathcal{V}$, compute the minimum constraint removal path $\pi^*(s,t) = \{e_1, \ldots, e_n\}$ on $\mathcal{G}$, so that the number of unique constraints on the path $\mathtt{c}(\pi) = \cup_i c_{e_i}$ is minimized over all $\pi(s,t)$.

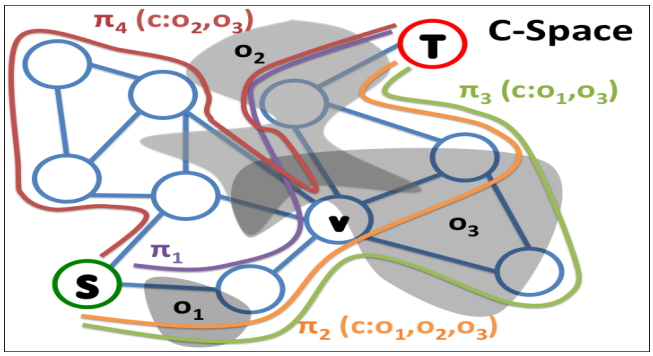

Figure 2: A graph embedded in a space with constraints. Dark regions: regions with constraints, which if removed they lead to a feasible path.

The Case of Manipulation Consider the following 3D setup: (a) A robotic manipulator, that acquires configurations $q \in \mathcal{Q}$; (b) A set of movable rigid-body objects $\mathcal{O}$, where each object $o_i \in \mathcal{O}$ can acquire a pose $p_i \in SE(3)$; (c) A set of grasping configurations $q(p_i) \in \mathcal{Q}_{grasp}$ for grasping an object at pose p_i.

Given this setup define a graph structure $\mathcal{G}(\mathcal{V}, \mathcal{E})$ in the configuration space, using either sampling-based planners (Kavraki et al. 1996; LaValle and Kuffner 2001), or by implicitly considering a discretization of $\mathcal{Q}$ (Cohen, Chitta, and Likhachev 2014). $\mathcal{V}$ are robot configurations, including grasping configurations $\mathcal{Q}_{grasp} \subset \mathcal{Q}$, and an edge $e \in \mathcal{E}$ connects two configurations. Furthermore, each object o_i at pose p_i defines a subset of configurations $\mathcal{Q}_{col}^{o_i} \subset \mathcal{Q}$ for the manipulator that are in collision. Then the constraints c_e for an edge e are the objects o_i colliding with the manipulator while traversing this edge. These constraints need to be avoided in the context of the MCR problem. Fig. 2 shows a graph where the gray regions correspond to constraints. This paper focuses on this manipulation version of the MCR problem. The solution path corresponds to the minimum number of objects $\mathcal{O} \setminus o_i$ that need to be removed for the manipulator to grasp a target object o_i at p_i.

Search for MCR Paths

The basic framework for finding MCR paths corresponds to a best-first search methodology over the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, which makes use of a priority queue Q. The priority queue holds search elements $u = \{\mathtt{v}, \pi, c, f\}$, which correspond to:

- $u.\mathtt{v}$: The graph node $u.\mathtt{v} \in \mathcal{V}$ for the search element.
- $u.\pi$: The path from start $s \in \mathcal{V}$ to $u.\mathtt{v}$, with length $|u.\pi|$.
- $u.c$: The constraints along $u.\pi$.
- $u.f$: An evaluation function, which typically depends on the path length $|u.\pi|$ and a heuristic estimate $h(u.\mathtt{v}, t)$ for reaching the target t. For uniform-cost search, it is $u.f = |u.\pi|$ but for A^*: $u.f = |u.\pi| + h(u.\mathtt{v}, t)$.

Best-First Search for Finding the Shortest Path

Consider the shortest path on $\mathcal{G}$ that ignores the objects. The priority queue is using only the evaluation function $u.f$ to order the search elements. During each iteration the top element u_{top} is removed from the queue and its neighbors $\mathtt{v}_{neigh} \in Adj(\mathcal{G}, u_{top}.\mathtt{v})$ are considered. If they have not been visited before, a new search element u_{neigh} is added to the queue: $u_{neigh} = \{\mathtt{v}_{neigh}, \pi_{neigh}, u_{top}.c \cup c_e, |u_{neigh}.\pi| + h(\mathtt{v}_{neigh}, t))\}$. If the node has been visited, the queue is not affected, because a search node with shorter path has already visited $\mathtt{v}_{neigh}$. This results in an efficient algorithm due to the dynamic programming structure of shortest length paths.

Exhaustive and Greedy Search for MCR

It may seem that the same process can be used to compute MCR paths by replacing the ordering of the priority queue, i.e., use first the number of constraints $|u_{top}.c|$ and then the evaluation function $u_{top}.f$. Furthermore, this "greedy" approach (Hauser 2013b) will prune a newly path π' to vertex $\mathtt{v}$ if $|\mathtt{c}(\pi'(s,\mathtt{v}))| \geq |\mathtt{c}(\pi(s,\mathtt{v}))|$, where π is an already discovered path to $\mathtt{v}$.

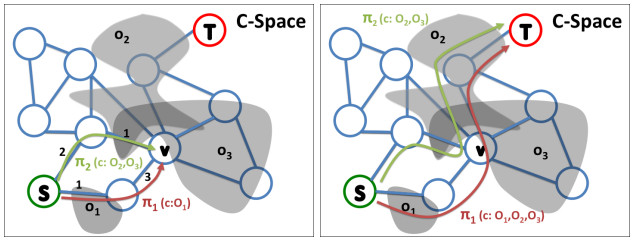

Figure 3: *Left:* Two paths π_1 and π_2, from s to $\mathtt{v}$. π_2 has 2 constraints: o_2, o_3, while π_1 has only 1: o_1. At node $\mathtt{v}$, the optimum path is π_1. *Right:* Two paths that extend π_1 and π_2 to the target. The optimum MCR solution is the extension of π_2, with only 2 constraints versus 3 for π_1's extension.

The problem with this approach is depicted in Fig. 3. Consider the paths from s to node $\mathtt{v}$: π_1 goes through constraint o_1 while π_2 goes through o_2 and o_3. Evidently, the optimal MCR path among the two is π_1. This means, that once π_2 is considered it will be discarded by the greedy approach. All paths, however, from node $\mathtt{v}$ to node t go through constraints o_2 and o_3. As a result, the extension of π_1 to t will have 3 constraints, while the extension of π_2 will have only 2 constraints.

One naïve solution that addresses this challenge is to exhaustively consider all simple paths in the space, i.e., those without loops. This means that every time a neighbor $\mathtt{v}_{neigh}$ is discovered by the best-first procedure, a corresponding search element is always added to the queue Q, unless $\mathtt{v}_{neigh}$ is already included in the path to $u_{top}.\mathtt{v}$. Obviously, this is a very expensive process that does not scale well.

Exact Search

There is an alternative to the naïve solution, which prunes some paths and can find the minimum constraint path, referred as "exact" approach (Hauser 2013a). The observation is that if the set of constraints of a path is dominating the set of constraints of another path to the same node, then the first one cannot possibly be a subset to an optimal solution. This means that a new path π' approaching $\mathtt{v}$ will be pruned only if $\exists\ \pi$ so that $\mathtt{c}(\pi(s,\mathtt{v})) \subset \mathtt{c}(\pi'(s,\mathtt{v}))$ and $|\pi'| > |\pi|$. If the constraints are exactly the same, then π' will be pruned only if $|\pi'| > |\pi|$. This algorithm will also expand multiple times the same node, because paths with different combinations of constraints can reach the same node.

Although the exact approach is slower than the "greedy", it is guaranteed to return the MCR solution and is faster than the exhaustive search because it prunes sets of paths. For example in Fig. 3 the algorithm will not prune away any of the π_1 and π_2 paths. In most manipulation applications, the full search space is significantly larger than the pruned space considered by this method. The previously described algorithms can waste a lot of time searching away from the solution before they decide to move towards the target.

Bounded-Length Exact Search

This paper proposes an approximate method that it is faster than both the exact and greedy algorithms, with some trade-offs on the number of constraints computed. A Bounded-length version of the EXACT_MCR approach is shown in Algorithm 1. This version also allows to incrementally increase the length of the paths considered by the search until there is convergence to the true optimal.

Algorithm 1: BL_MCR$(\mathcal{G}(\mathcal{V},\mathcal{E}), s, t, threshold)$

```
1  Q ← NEW_ELEMENT(s, ∅, ∅, 0);
2  while Q not empty do
3      u_top ← Q.pop();
4      if u_top.v == t then
5          return u_top.π;
6      for each v_neigh ∈ Adj(G, u_top.v) do
7          π ← u_top.π | e(u_top.v, v_neigh);
8          if |π| < threshold then
9              c ← u_top.c ∪ e(u_top.v, v_neigh).c;
10             if IS_NEW_SET(v_neigh.C, c) then
11                 f ← |π| + h(v_neigh, t);
12                 Q ← NEW_ELEMENT(v_neigh, π, c, f);
13     return ∅;
```

Alg.1 works similar to the "exact" approach. The method uses a priority queue, which prioritizes search elements that have a low number of constraints $|c|$. Ties are broken in favor of elements with a small evaluation function f. While there are nodes in the queue (line 2), the algorithm will pop the highest priority search element u_{top} (line 3) and check if the corresponding graph node is equal to the target node t (line 4). If the nodes are equal then the algorithm will return the path stored in the search element (line 5). Otherwise, all the adjacent graph nodes $\mathtt{v}_{neigh}$ of $u_{top}.\mathtt{v}$ are considered (line 6). The main difference from the "exact" approach is that the BL_MCR approach first checks if the path has length greater than the threshold (lines 7-8). If the path is within the threshold, then the function IS_NEW_SET (line 10) will check if the new set of constraints $\mathtt{c}$ is a subset of any of the constraints of paths that have already been generated for $\mathtt{v}_{neigh}$. For speed purposes, each graph node can keep track of the different sets of constraints, $\mathbf{C}$, for each path that has reached the node, and the corresponding evaluation functions f. The operation of the approach is the following:

- If there is a constraint set $\mathtt{c}' \in \mathbf{C}$, where $\mathtt{c}' \subset \mathtt{c}$ then the algorithm returns false and the node is not added in Q.
- If there is a constraint set $\mathtt{c}' \in \mathbf{C}$, where $\mathtt{c} \equiv \mathtt{c}'$, then the algorithm will compare the f values. If the new f' value is smaller than the existing one, IS_NEW_SET returns true.
- If there is a constraint set $\mathtt{c}' \in \mathbf{C}$, where $\mathtt{c} \subset \mathtt{c}'$ then the set $\mathtt{c}'$ will be replaced by $\mathtt{c}$ and the method returns true.
- If none of the above is true, then the algorithm will return true and $\mathtt{c}$ will be added in the $\mathbf{C}$ list of the graph node.

If IS_NEW_SET detects that the constraints set includes new constraints, then the algorithm will create a new search element and update the corresponding node, using the NEW_ELEMENT function (lines 11-12).

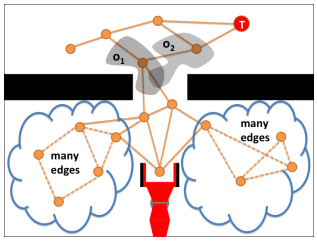

Figure 4: A case where the Bounded-length algorithm will not waste time checking the collision free areas, before moving to the target.

The proposed algorithm with a small threshold can search through short length paths that violate constraints fast without concentrating time searching through constraint-free nodes far from the shortest paths. Te algorithm cannot guarantee the optimum solution but in practice it returns good ones, close to those of the exact approach for MCR and with paths that are of known degradation relatively to the shortest.

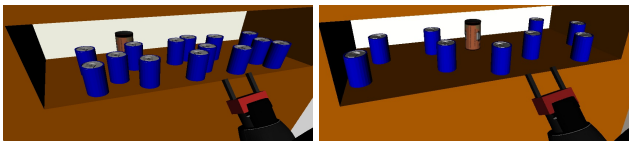

Figure 5: Shelf challenge: (left) 15 objects and (right) 10 objects blocking the way for the arm to reach the beer can.

Evaluation

The methods have been tested in the setup of Fig. 1 and in a cluttered shelf with limited maneuverability as in Fig. 5. A model of a Baxter arm is used for testing. For the benchmark environment the Baxter arm has to move a target object from the left side to the right side inside a shelf, while 9 objects are blocking the straight transfer path for the arm. For the shelf, 10 to 20 cylinders are placed randomly. The objective is to detect the minimum number of cans that need to be moved to reach a beer can at the back of the shelf (Fig.5).

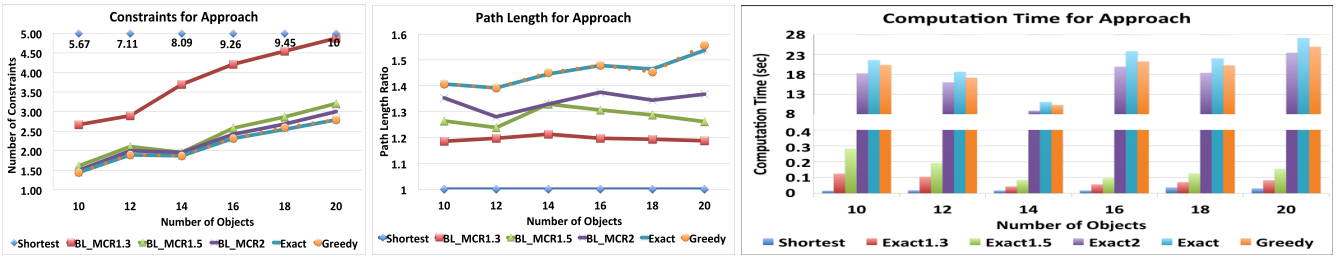

Figure 6: Average number of constraints (left). Length ratio relative to shortest path (middle). Computation time (right).

Four methods are tested: (a) shortest path, (b) the greedy algorithm, (c) the exact search and (d) the bounded-length version of the exact approach with different thresholds (1.3, 1.5 and 2.0). 50 experiments were performed for each combination of method and environment. The shortest path algorithm ignores the movable objects.

A transit roadmap was precomputed that contained on each edge the set of objects that led to collisions. This allowed to speed up the execution of the experiments so that each path is computed in sub-second time. If this preprocessing is not available, then the computation time for each algorithm is approximately two orders of magnitude larger, since collision checking needs to be performed online. This change affects all algorithms uniformly.

Fig. 7 shows the results of a single run for the benchmark of Fig.1. The shortest path goes through the majority of the objects to reach the target. The robot would need to clear on average 8 objects before performing this transfer with the shortest path. The remaining algorithms find a better path with only 2 constraints. The exact and greedy methods are relatively expensive in computing a solution. In this case, the greedy method manages to find the correct number of constraints, however. As the threshold for the bounded-length version becomes smaller, the algorithm becomes faster in finding a solution. The bounded-length variant that used a threshold of 3 found the same path length path and constraints with the exact method but quite faster. With lower thresholds, the algorithm manages to find way faster a path but more constraints.

Variables	*Time*	*Constraints*
SHORTEST_PATH	0.093	7
BL_MCR1.3	0.195	7
BL_MCR1.5	0.201	5
BL_MCR2	0.454	5
BL_MCR3	0.814	3
EXACT_MCR	1.86	3
GREEDY_MCR	1.552	3

Figure 7: Average computation time, length ratio relative to the shortest path and number of constraints for all the algorithms in the benchmark.

As Fig. 6 shows the shortest path returns a significant number of constraints, although it is the shortest in length. The remaining algorithms return fewer constraints and close to those of the exact solution, with the exception of BL_MCR using the smallest threshold. As the threshold is relaxed, the number of constraints approaches that of the exact. The exact and the greedy methods return the longest paths. Fig. 6 (middle) depicts the ratio of the path lengths with respect to the shortest path. As the threshold for the bounded length version decreases, the path returned is closer to the shortest path.

The computation time for the scenario in the shelf is shown in Fig. 6(right). The shortest path does not check for collisions with the movable objects and returns solutions faster than alternatives. For all the other algorithms, the computation time increases. The exact method takes the longest time. Although the greedy algorithm expands fewer nodes, the computation time is almost the same. Nevertheless, the bounded-length variants result in faster computation of the solution. The difference in time between the algorithms relates to the issue described in Fig.4. The greedy and the exact algorithm will waste a lot of time searching over the collision-free edges before moving through the objects, a situation that arises often in manipulation. The proposed method will avoid expanding nodes that are far away from the target. In summary, it appears that a threshold of 1.5 tends to return solutions with a similar number of constraints as the exact algorithm but considerably faster.

Discussion

This paper studies a challenge that should be of interest to the combinatorial search community. This is the problem of computing "minimum constraint removal" (MCR) paths, which is important in many application domains, such as robot manipulation. This is a computationally hard problem in the general case as such paths do not exhibit dynamic programming properties. Various algorithmic alternatives are described here, ranging from approximate to exact algorithms. Approximate solutions that bound the path length of the considered path seem to provide a desirable tradeoff in terms of returning solutions with a low number of constraints, relatively short path lengths and low computation time. It is interesting to consider how the arsenal of methodologies that have been developed by the combinatorial search community can be used to further improve performance in the context of planning MCR paths.

Acknowledgments

The authors are with the Computer Science Dept. at Rutgers University, NJ, USA. Their work is supported by NSF awards IIS-1451737, CCF-1330789. Any opinions, findings and conclusions or recommendations expressed in this paper do not necessarily reflect the views of the sponsors.

References

Botea, A.; Muller, M.; and Schaffer, J. 2002. Using Abstraction for Planning in Sokoban. *Computers and Games* 360–375.

Cohen, J. B.; Chitta, S.; and Likhachev, M. 2014. Single- and dual-arm motion planning with heuristic search. *International Journal of Robotic Research* 33(2):305–320.

Dogar, M., and Srinivasa, S. 2011. A Push-Grasping Framework in Clutter. In *RSS*.

Erickson, L. H., and LaValle, S. M. 2013. A Simple, but NP-Hard Motion Planning Problem. In *AAAI Conference on Artificial Intelligence*.

Gobelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. In *International Conference on Automated Planning and Scheduling*.

Hauser, K. 2013a. Minimum Constraint Displacement Motion Planning. In *RSS*.

Hauser, K. 2013b. The minimum constraint removal problem with three robotics applications. *The International Journal of Robotics Research* 0278364913507795.

Hearn, R., and Demaine, E. 2005. P-Space Completeness of Sliding-block Puzzles and other Problems through the Non-Deterministic Constraint Logic Model of Computation. *Theoretical Computer Science* 343(1):72–96.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic Roadmaps for Path Planning in High-Dim. Configuration Spaces. *IEEE TRA*.

Krontiris, A., and Bekris, K. E. 2015. "dealing with difficult instances of object rearrangement". In *Robotics: Science and Systems (RSS)*.

Krontiris, A.; Shome, R.; Dobson, A.; Kimmel, A.; and Bekris, K. E. 2014. Rearranging similar objects with a manipulator using pebble graphs. In *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*.

Krontiris, A.; Luna, R.; and Bekris, K. E. 2013. From Feasibility tests to Path Planners for Multi-agent Pathfinding. In *Sixth Annual Symposium on Combinatorial Search (SoCS)*.

LaValle, S. M., and Kuffner, J. J. 2001. Randomized Kinodynamic Planning. *IJRR*.

Luna, R., and Bekris, K. E. 2011. Efficient and Complete Centralized Multi-Robot Path Planning. In *IEEE-RAS International Conference on Intelligent Robots and Systems (IROS)*.

McCarthy, Z.; Bretl, T.; and Hutchinson, S. 2012. Proving Path Non-Existence using Sampling and Alpha Shapes. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2563–2569.

Pereira, A. G.; Ritt, M. R. P.; and Buriol, L. S. 2013. Finding Optimal Solutions to Sokoban Using Instance Dependent Pattern Databases. In *Symposium on Combinatorial Search (SoCS)*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. 2015. The Conflict-based Search Algorithm for Multi-Agent Pathfinding. *Artificial Intelligence Journal (AIJ)* 40–66.

Stilman, M., and Kuffner, J. 2004. Navigation among Movable Obstacles: Realtime Reasoning in Complex Environments. In *Humanoid Robotics*, 322–341.

Stilman, M., and Kuffner, J. J. 2006. Planning Among Movable Obstacles with Artificial Constraints. In *WAFR*.

Wilfong, G. 1988. Motion Planning in the Presence of Movable Obstacles. In *Annual Symp. of Computational Geometry*, 279–288.

Zhang, L.; Kim, Y.; and Manocha, D. 2008. A Simple Path Non-Existence Algorithm using C-Obstacle Query. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.