Minimum Constraint Displacement Motion Planning

Kris Hauser
School of Informatics and Computing, Indiana University at Bloomington
Email: hauserk@indiana.edu

***Abstract*—This paper formulates a new minimum constraint displacement (MCD) motion planning problem in which the goal is to minimize the amount by which constraints must be displaced in order to yield a feasible path. It presents a sampling-based planner that asymptotically approaches the globally optimal solution as more time is spent planning. Key developments are efficient data structures that allow the planner to select small subsets of obstacles and their displacements that are candidates for improving the current best solution, and local optimization methods to improve convergence rate. The resulting planner is demonstrated to successfully solve MCD problems with dozens of degrees of freedom and up to one hundred obstacles.**

I. Introduction

The classical motion planning problem asks to find a path that satisfies obstacle avoidance and other constraints. If no feasible path exists, most planners either run forever or take a very long time to terminate. In several applications it would be advantageous for a motion planner to reason about constraints that *must be violated* in order to yield a feasible path:

- *Mobile manipulators* might need to move obstacles aside in order to complete a task.
- *Crowd-navigating* robots might plan paths through other agents, knowing that they might clear a path to let the robot through.
- *Prioritized task specifications* may allow a robot to occasionally violate low-priority rules (e.g., courtesy) to obey high-priority ones (e.g., safety).
- *Task-based robot design* could recommend to an engineer how to choose design parameters to minimize cost while maintaining the capability of fulfilling some task (e.g., configuring a robot on an assembly line or designing a medical procedure in the human body). A similar problem is faced in *design for assembly*, with the human assembler taking the place of the robot.

Such problems require reasoning both about the robot's motion as well as changes to motion constraints.

This paper asks the question, "how can a robot clear the way for a feasible path while *minimizing displacement magnitudes*?" I formulate a new minimum constraint displacement (MCD) planning problem in which the robot minimizes a weighted sum of displacement costs and path length over the space of paths and displacements.

I present a configuration-space MCD planner that builds a probabilistic roadmap [10] of configurations and randomly-sampled displacements, and then solves the discrete version of the MCD problem on the graph. The discretization is grown in an incremental manner such that the discrete MCD solution

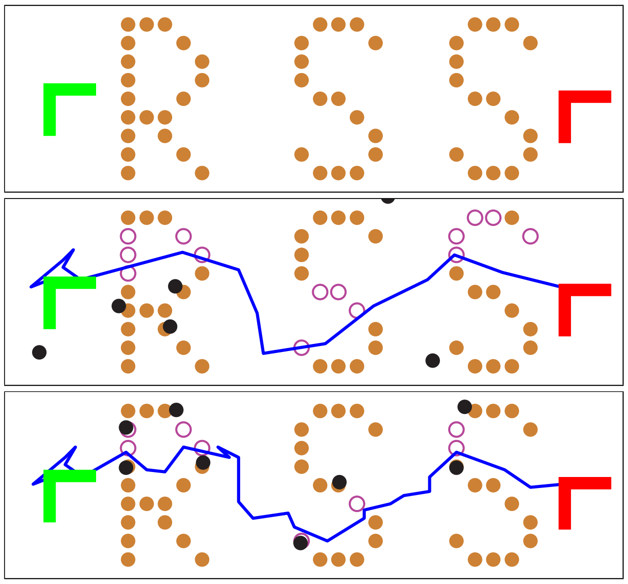


Fig. 1. Top: a translating and rotating L-shaped robot is asked to move from left to right among movable circular obstacles. Middle: after 3,000 iterations, a candidate 14-obstacle solution is found. Bottom: the planner continues, and after 10,000 iterations finds a 10-obstacle solution with small displacements.

asymptotically approaches the true optimum as more samples are drawn (Fig. 1). This paper also studies the discrete MCD problem; it is proven to be NP-hard, but I present search methods that tend to work well in practice.

I also present enhancements that improve scalability, primarily by drastically reducing the number of constraint checks in problems with many obstacles. These enhancements are based on the principle that infeasibility is usually caused by only a few obstacles and the planner should focus its effort on identifying and circumventing them. Local optimization methods also dramatically speed up the rate of convergence to minima for high-dimensional configuration spaces and in the presence of many obstacles.

The resulting planner is demonstrated to be practical for hundreds of obstacles and configuration spaces with dozens of degrees of freedom. Examples are shown on navigation among movable obstacles problems and a real-world robot design problem for a ladder-climbing humanoid.

II. Minimum Constraint Displacement Problems

A (continuous) MCD problem is specified as follows:

Input. connected d-dimensional configuration space $\mathcal{C} \subseteq \mathbb{R}^d$, terminal configurations q_s, $q_g \in \mathcal{C}$, and n *obstacle regions*

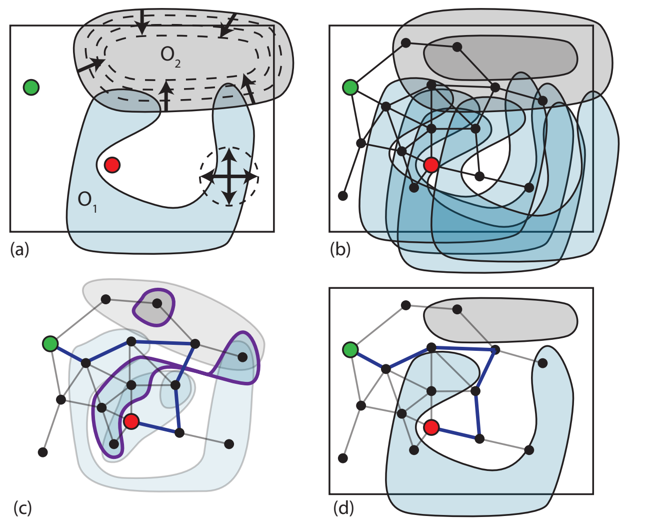


Fig. 2. (a) Converting a continuous MCD problem into a discrete one. The original problem contains obstacles O_1 and O_2, with d_1 a translation and d_2 a shrinking operation. (b) A roadmap is built in the configuration space, and 4 displacements of O_1 and 2 displacements of O_2 are sampled. (c) This is converted to a discrete MCD problem on a graph, and an optimal solution that shrinks O_2 is found (path and displacements highlighted). (d) The solution to the original continuous problem.

$O_1(d_1), \ldots, O_n(d_n) \subseteq \mathcal{C}$ which are open sets that depend on *displacement parameters* $d_1 \in \mathcal{D}_1, \ldots, d_n \in \mathcal{D}_n$.

The *displacement spaces* $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are sets of arbitrary dimension and are assumed to contain a zero element. Static obstacles O_i are encoded by setting $\mathcal{D}_i = \{0\}$. Otherwise, the mapping $O_i(d_i)$ determines how C-space obstacle regions are displaced, with $O_i(0)$ giving the original obstacle. The planner makes no restriction on how displacements change C-obstacle shape, and they may encode, for example, translation, rotation, shrinking, or deformation (Fig. 2).

Definition. A *feasible* solution consists of a path $y(u) : [0,1] \rightarrow \mathcal{C}$ together with displacements $d_1, \ldots, d_n$ such that y satisfies endpoint constraints $y(0) = q_s$ and $y(1) = q_g$ and does not overlap the displaced obstacles: $y(u) \notin \bigcup_{i=1}^{n} O_i(d_i)$ for all $u \in [0,1]$.

Output. A *minimum constraint displacement* (MCD) is a feasible solution $(y, d_1, \ldots, d_n)$ that minimizes the cost

$$J(y, d_1, \ldots, d_n) = w_L L(y) + \sum_{i=1}^{n} C_i(d_i) \tag{1}$$

where L gives the length of the path, and C_i are nonnegative *displacement costs*. This paper uses

$$C_i(d_i) = ||d_i|| + w_0 I[d_i \neq 0], \tag{2}$$

in which I is an indicator function. The first term penalizes displacement magnitude and the latter term penalizes any nonzero displacement, which is useful to encode the extra effort involved in manipulating obstacles. The w_L term specifies the relative penalty for taking longer paths vs. larger displacements (see Fig. 3). The examples in this paper use $w_L = 0.1$ and $w_0 = 0.1$ unless otherwise noted.

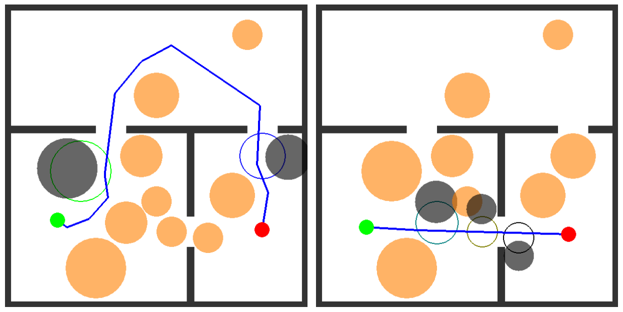


Fig. 3. Changing the path length weight from 0.1 (left) to 1 (right) causes the robot to choose a more direct path at the expense of a larger displacement.

MCD generalizes several existing motion planning problems. An instance of the classical Piano Mover's problem [12] has a feasible solution iff there exists a solution to an MCD with all displacements equal to zero. Moreover if w_0/w_L is at least the length of the shortest feasible path, then MCD solves the shortest path problem. Finally, MCD generalizes the minimum constraint removal problem [7] by encoding obstacles to vanish with nonzero displacements, i.e. $O_i(d_i) = \emptyset$ if $d_i \neq 0$.

A key subroutine in the planner is to solve the Discrete MCD problem, which is specified as follows:

Input. Graph $G = (V, E)$, start and terminal vertices $s, t \in V$, obstacle function $O[i, d]$ with $i = 1, \ldots, n$ and $d = 1, \ldots, m_i$ denoting a subset of V covered by obstacle i at a candidate displacement d. (Note that each obstacle may have differing numbers of candidate displacements). $C[i, d] \geq 0$ is the displacement cost function.

Output. A path $s \rightsquigarrow t$ and a set of displacements $d_1, \ldots, d_n$ such that $(s \rightsquigarrow t) \cap \bigcup_{i=1}^{n} O[i, d_i] = \emptyset$ and such that the cost

$$J(s \rightsquigarrow t, d_1, \ldots, d_n) = w_L |s \rightsquigarrow t| + \sum_{i=1}^{n} C[i, d_i] \tag{3}$$

is minimized. I will say that a node v of the graph is c-reachable for an arbitrary constant c if there exists a path $s \rightsquigarrow v$ and a compatible assignment $d_1, \ldots, d_n$ such that $J(s \rightsquigarrow v, d_1, \ldots, d_n) \leq c$.

III. Continuous MCD Planner

This section presents an MCD motion planner that uses random sampling to build a roadmap of configurations in $\mathcal{C}$ and a set of displacement values, and runs discrete MCD queries on this discretization. As more samples are drawn, the optimal solution in the discretization approaches the optimal solution in the continuous space. It also describes sampling strategies and local optimization techniques that improve the convergence rate beyond the basic algorithm.

A. Summary

The planner is specialized to a problem instance given subroutines *Feasible?*(q, O_i, d_i) that tests whether configuration $q \neq O_i(d_i)$, *Visible?*(a, b, O_i, d_i) that tests whether the line segment between configurations a and b intersects $O_i(d_i)$, and

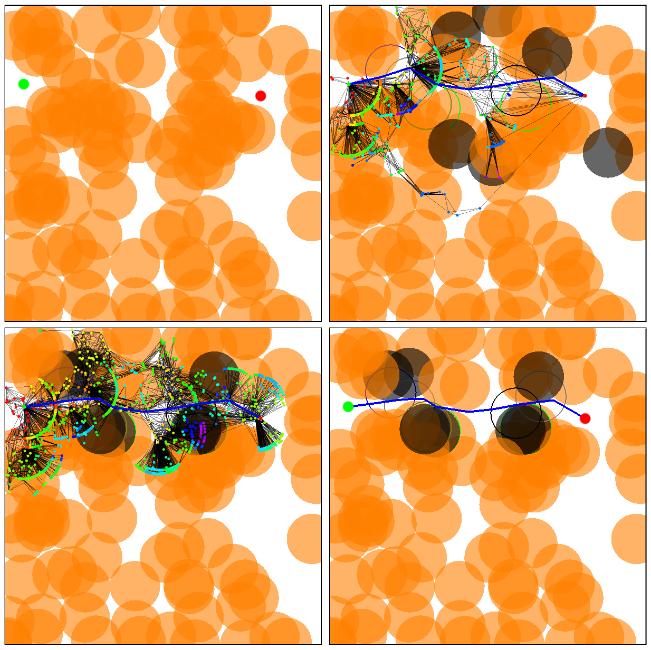


Fig. 4. A point robot scenario with 100 circular obstacles. The roadmap is grown with incrementally larger cost bounds until a path is found. Then, the roadmap is refined until it contains a path with 6 moved obstacles.

Sample-Disp$(\mathcal{D}_i, c_{max})$ that randomly samples a displacement d_i from $\mathcal{D}_i$ with cost $C_i(d_i) \leq c_{max}$.

It builds a probabilistic roadmap $G = (V, E)$ and displacement sets $D_1, \ldots, D_n$, where each D_i, $i = 1, \ldots, n$ is a set of samples drawn from $\mathcal{D}_i$. The roadmap is initialized to contain the start and goal configurations and $D_i = \{0\}$ for all $i = 1, \ldots, n$. The planner than alternates between *roadmap expansion* and *displacement sampling* steps. It also performs *local optimization* steps when a better solution is found. The planner runs as follows:

Continuous-MCD:
```
1. $G \leftarrow (\{q_s, q_g\}, \emptyset)$, $D_1, \ldots, D_n \leftarrow \{0\}$
2. $J_{max} \leftarrow 0$
3. For $N = 1, 2, \ldots$ repeat:
4.     If solution not found, set $J_{max} \leftarrow J_{max} + \Delta J$.
5.     Run Expand-Roadmap $n_e$ times,
6.     Run Sample-Displacement.
7.     If a new optimal solution is found:
8.         Run Local-Optimize for $n_o$ iterations.
9.         Set $J_{max}$ to the cost of the best solution.
```

The planner maintains an exploration cost limit J_{max} that prevents the planner from considering high cost solutions. Before a solution is found, J_{max} is incrementally raised by a step size ΔJ; once one is found, J_{max} is set to the level of the cost of the best solution found so far, and will be progressively reduced as the planner finds better solutions (Fig. 4).

The balance between the amount of time spent in *Expand-Roadmap*, *Sample-Displacement*, and *Local-Optimize* is governed by the parameters n_e and n_o. Currently, these values and ΔJ must be chosen via empirical tuning. Performance does not seem to be particularly sensitive to the choice of ΔJ and n_o because J_{max} will drop as soon as a solution is found, and *Local-Optimize* tends to be only a small fraction of the overall cost. However, n_e does significantly affect performance, and values that are too low or too high should be avoided. All examples use values $\Delta J = 0.002$, $n_e = 20$, and $n_o = 100$.

B. Roadmap Expansion

Expand-Roadmap expands only from nodes that are J_{max}-reachable on G, as given by the following pseudocode. It operates by generating a random sample q_d (Line 1) and extends an edge from the closest J_{max}-reachable milestone toward q_d (Lines 2–3). The resulting leaf node q is then connected to nearby milestones in G (Lines 4-6). Each extension and connection is limited to a maximum step size δ.

Expand-Roadmap
```
1. $q_d \leftarrow$ Sample()
2. Let $q_n \leftarrow$ Closest$(G, J_{max}, q_d)$
3. $q \leftarrow$ Extend-Toward$(q_n, q_d, \delta, J_{max})$
4. Let $\{q_1, \ldots, q_m\} \leftarrow$ Neighbors$(G, J_{max}, q)$
5. For $i = 1, \ldots, m$, do:
6.     If $d(q_i, q) < \delta$, then add $q_i \rightarrow q$ to $E$
7. Call Discrete MCD to update optimal costs on $G$
```

The subroutine *Closest*(G, J_{max}, q) finds the closest J_{max}-reachable vertex in G, where closeness is measured by a distance metric $d(q, q')$. *Neighbors*(G, J_{max}, q) returns a set of milestones $q_1, \ldots, q_m$ that are close to q and such that the path through the milestone to q would be J_{max}-reachable, assuming all constraints are feasible at q. As demonstrated by Karaman and Frazzoli [9], the choice $m = (1 + 1/d)e \log |V|$ ensures that the optimal path in the roadmap asymptotically approaches the true optimum as more samples are drawn. *Extend-Toward*$(q_i, q, \delta, J_{max})$ extends the roadmap with a new edge from q_i to a configuration q' in the direction of q. Like RRT, the step size is limited to δ by taking $q' = q_i + \min(\frac{\delta}{d(q_i,q)}, 1)(q - q_i)$. Line 7 computes a Discrete MCD update as described in Sec. IV.

C. Displacement Sampling

Displacement sampling serves two functions: 1) allowing the roadmap to explore blocked areas of the space, and 2) improving the costs of existing paths. Uniform sampling allocates samples inefficiently, so the planner uses *unblocking* and *refining* strategies that concentrate samples where they are needed.

Unblocking selects only among obstacles that prevent nodes from being reached within cost J_{max}. It computes the set of obstacles that are infeasible at blocked nodes, and picks an obstacle O_i from this set. Then, it calls *Sample-Disp*$(\mathcal{D}_i, c_{max})$ with c_{max} determined by calculating the maximum cost such that if the new sample were to be feasible at some blocked node, the cost of the path to that node would be less than J_{max}.

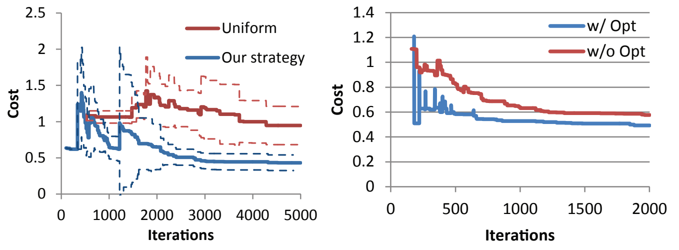


Fig. 5. Left: Comparing the displacement sampling strategy with uniform sampling. Results are gathered over 10 trials on Fig. 4. Solid and dashed curves indicates the mean/standard deviation of the optimal solution cost at a given iteration count, only taken over trials that have found a solution. Right: Local optimization significantly improves convergence rate. Results are gathered over 10 trials on Fig. 3.

Refining selects from obstacles that may help improve the solution itself or the cost of paths to non-terminal nodes. This latter functionality is often needed to help the planner reach new homotopy classes. To refine, the planner considers all optimal assignments to nodes, and counts for each obstacle how many nodes could be improved by an adjusted displacement sample. It then picks an obstacle with probability proportional to this count. Like unblocking, it calls *Sample-Disp* with c_{max} chosen so that the cost of the path to some node can indeed be improved.

Until a first solution is found, unblocking is chosen exclusively because it is useless to consider samples that do not have the potential to unblock unexplored regions of the space. Afterwards, the planner picks between both strategies according to some probability (currently, a 50/50% mix is used). Experiments show a significant improvement upon uniform sampling (Fig. 5, left), finding a first solution 4.2x faster and converging to low-cost solutions much faster.

D. Local optimization

Although random sampling helps explore globally, it converges to an optimum rather slowly. So, the planner uses local optimization to improve solution cost quickly whenever a new candidate for an optimum is found. I extend to the many-parameter case a randomized descent method presented in [8] for optimizing a path given a single displacement parameter. Fig. 5, right, shows that local optimization improves convergence rate beyond pure random sampling, allowing the planner to find better solutions for a given time budget.

Optimizing path only. A simple method for optimizing the path is to perform shortcutting with respect to the current displacements. Intervals along the path are selected at random, and straight-line shortcuts are constructed and checked for collision. If the shortcut is collision-free, then the intervening portion of the path is replaced.

Optimizing displacements. The planner uses a randomized coordinate descent to optimize displacements subject to the feasibility of the current path. When a descent step $d_i \rightarrow d_i'$ of obstacle O_i is found to cause infeasibility, the optimizer adjusts the subpath that obstructed the move to d_i' in an attempt to make it feasible. Specifically, each vertex on the subpath is perturbed until it becomes feasible with respect to $O_i(d_i')$, and such the solution does not exceed the current best cost. Afterward, the remaining obstacles are checked, and if the path adjustment caused infeasibility, the move to d_i' is rejected. Rejections are generally unlikely because most subpaths are short segments and their adjustments are unlikely to be affected by other obstacles.

E. Convergence

Here I prove that the planner is asymptotically optimal given relatively weak assumptions. For simplicity, the proof ignores local optimization and assumes uniformly sampled displacements.

Theorem 1. Let $(y^\star, d_1^\star, \ldots, d_n^\star)$ be an optimal solution with cost $J^\star$. Let $\epsilon > 0$ be an arbitrary constant denoting a limit on allowable suboptimality, and let $\delta > 0$ be a constant denoting minimum path clearance. Assume the volume of $\mathcal{D}_i$ is finite and nonzero. If for each i there exists a subset $B_i \subseteq \mathcal{D}_i$ such that:

1) B_i has nonzero volume whenever $d_i^\star \neq 0$,
2) for all $d_i \in B_i$, there exists a solution path homotopic to $y^\star$ with clearance δ from the obstacles $O(d_1), \ldots, O(d_n)$, and
3) for all $d_i \in B_i$, $C_i(d_i) \leq C_i(d_i^\star) + \epsilon/n$,

then the planner converges to a path with cost less than or equal to $J^\star + \epsilon$ with probability 1.

Proof. First, it holds by assumption that each *Sample-Displacement* call has a nonzero probability of sampling a displacement in B_i whenever condition 1 holds. Hence, over time, the planner's sample sets D_i hit all of the B_i with probability approaching 1 as N grows. Second, the cost of the lowest-cost path in G approaches that of the true optimal path when restricted to the current sets $D_1, \ldots, D_n$, since RRT* converges to the optimal path with probability 1 as N grows given condition 2 [9]. Hence, if all B_i are hit (event A) and G converges (event B), then the sum of the best solution path plus the sum of its displacement costs is no more than $J^\star + \epsilon$ given condition 3. The conjunction of A and B holds with probability approaching 1, giving the desired result. □

All else equal, the convergence *rate* becomes slower with larger n, higher-dimensional $\mathcal{D}_i$, and higher-dimensional $\mathcal{C}$.

Fig. 6 demonstrates an example in which a displacement rotates a block rather than translates it. The centers of rotation are staggered so that the optimal path passes along a zigzag motion. This example has 64 locally optimal solutions, corresponding to paths above/below each block, and each block rotated clockwise and counterclockwise. Over 10 runs, the planner finds a first solution in 2.9 s and 8/10 reached an optimal homotopy class within 60 s. Of those that reached an optimal homotopy class, average time is 26 s, with std. dev. of 19 s. (This and all other experiments in this paper are performed on a 2.67GHz Intel Core i7 CPU on one core.)

IV. Algorithms for Discrete MCD

Now I return to the discussion of Discrete MCD. To my knowledge, this is an entirely new class of problems that

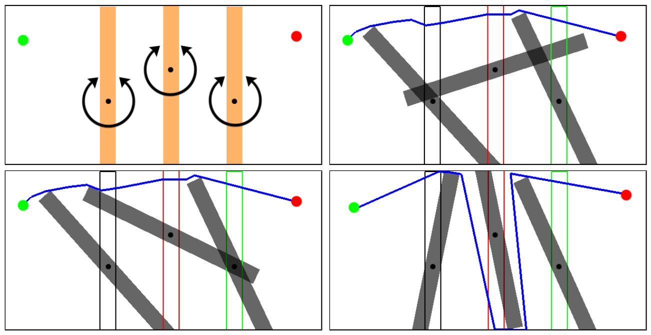


Fig. 6. A point robot scenario with blocks that can be rotationally displaced. The first solution, found after 1,500 iterations, is in a suboptimal homotopy class, and local optimization reaches a local minimum. After 3,000 iterations the planner finds a solution in the globally optimal homotopy class that zig-zags between the rotated obstacles.

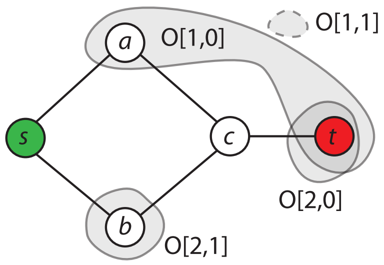


Fig. 7. An example on which greedy search fails. Each of the two obstacles has two candidate displacements $d_i \in \{0, 1\}$, with costs $C[i, 0] = 0$ and $C[i, 1] = 1$ for $i = 1, 2$. $O[1, 1]$ causes the obstacle to vanish, while $O[2, 1]$ causes a shift. At c, greedy search prunes the upper path because its best displacement is $(d_1, d_2) = (1, 0)$ while the lower path has best displacement $(0, 0)$. Ultimately, the lower path fails to yield a feasible solution.

combines graph search and combinatorial optimization.

To construct a Discrete MCD instance on the graph, membership testing in $O[i, d]$ is computed on demand by calling the *Feasible?* subroutine. Results are memoized for faster lookup in subsequent calls. (For brevity, I do not describe edge checking via the *Visible?* subroutine, which requires a modest but straightforward modification of the algorithms below.)

It is easy to show that Discrete MCD is NP-hard by reduction from discrete minimum constraint removal (MCR). A discrete MCR problem can be converted into a discrete MCD problem by setting $m_i = 2$ for all i and making the zero displacement the "obstacle present" case and $d_i = 1$ the "obstacle absent" case. With a cost of 1 for removing obstacles and setting $w_L = 0$, solutions to MCD are in one-to-one correspondence the solutions to MCR and have cost equal to the number of $d_1, \ldots, d_n$ equal to one. Since MCR is NP-hard [7] and can be solved by MCD, MCD is NP-hard as well. So, one must either settle for an approximation or hope that typical problem instances can be solved efficiently. I present three practical algorithms, each with different tradeoffs:

1) Greedy search, which is worst case polynomial time, but is approximate.
2) Exact search, which is exact but has worst-case exponential time.
3) Back-checking search, which is designed to handle large numbers of displacements efficiently. It can be configured to behave like greedy or exact search.

In most practical examples, greedy search has very low error, and exact search runs only marginally slower than greedy search. However, in pathological cases greedy search can produce arbitrarily bad solutions, and exact search can visit a graph vertex exponentially many times.

With proper implementation, back-checking is strictly preferable to greedy search, and in certain useful problem subclasses it can be exact. All examples are generated using it. Furthermore, the planner reduces Discrete-MCD overhead during *Expand-Roadmap* using dynamic updates in the manner of shortest paths algorithms [5], which saves considerable time over recomputing Discrete-MCD from scratch because only a few optimal paths will typically change. However, I will begin by presenting greedy search to build intuition for the problem.

A. Greedy Search among Path Covers

The greedy search performs a best-first search among *path covers* at graph vertices. The cover of a path $s \rightsquigarrow v$ is defined as the subset of compatible displacements $S_1, \ldots, S_n$ such that

$$S_i = \{d \in \{1, \ldots, m_i\} \mid (s \rightsquigarrow v) \cap O[i, d] = \emptyset\}. \quad (4)$$

Starting from the root $N_0 = (s, S_1^0, \ldots, S_n^0)$ with each $S_i^0 = \{d \in \{1, \ldots, m_i\} \mid s \notin O[i, d]\}$, the algorithm searches among states $N = (v, S_1, \ldots, S_n)$ with search order according to the minimum cost of the partial path leading to v, among all assignments $d_1, \ldots, d_n$ with $d_i \in S_i$. In other words, the priority function is

$$P(v, S_1, \ldots, S_n) = w_L |s \rightsquigarrow v| + \sum_{i=1}^{n} \min_{d_i \in S_i} C[i, d_i] \quad (5)$$

where $|s \rightsquigarrow v|$ indicates the depth of v in the search tree, and nodes are expanded in order of increasing P. At each step the unexpanded state $N = (v, S_1, \ldots, S_n)$ with minimum P is expanded along every graph edge $(v, w) \in E$ to obtain a state $N' = (w, S'_1, \ldots, S'_n)$. Here, the successor cover S'_i, $i = 1, \ldots, n$ is computed by subtracting the displacements incompatible with w from S_i:

$$S'_i = \{d \mid d \in S_i \text{ and } w \notin O[i, d]\} \quad (6)$$

If any S_i is empty, the state is a dead end and can be pruned. Also, if a vertex is revisited during search, the state with lower cost is retained and the other is pruned. Search proceeds until the goal t is reached, at which point the search outputs the path to t and the minimum-cost displacement in its cover.

The search generates at most $|V|$ states, each of which maintains $O(M)$ storage, where $M = \sum_{i=1}^{n} m_i$ is the total number of candidate displacements. Running time is $O(M(|E| + |V| \log |V|))$ with the priority queue implemented as a Fibonacci heap. Although this method produces high-quality solutions in practice, the approximation error is unbounded; in some pathological cases, it fails to find a solution entirely (Fig. 7).

B. *Exact Search among Dominant Covers*

Greedy search prunes too aggressively; a suboptimal path to a vertex v may prove later to yield the optimal path at t. I derive an exact technique by only pruning paths that are provably suboptimal in a global sense. To do so, I introduce the concept of *cover dominance*.

Definition. A cover $(S_1, \ldots, S_n)$ *dominates* another $(S'_1, \ldots, S'_n)$ if $S'_i \subseteq S_i$ for all $i = 1, \ldots, n$.

The principle behind exact search is that among two paths ending at a vertex v, a path with a dominated cover and longer length can safely be pruned. That is, if $N = (v, S_1, \ldots, S_n)$ and $N' = (v, S'_1, \ldots, S'_n)$ are two states encountered during search, then it may prune N' if $(S_1, \ldots, S_n)$ dominates $(S'_1, \ldots, S'_1)$ and $|s \rightsquigarrow N| \leq |s \rightsquigarrow N'|$. Otherwise it can prune N if the converse is true. If the covers are equal, then either one may be pruned.

Theorem 2. If N' is pruned as described above and there exists an optimal solution along the path $N_0 \rightsquigarrow N' \rightsquigarrow t$, then there also exists an optimal solution along $N_0 \rightsquigarrow N \rightsquigarrow t$.

Proof. Let $(d_1, \ldots, d_n)$ be the assignment to an optimal solution $N_0 \rightsquigarrow N' \rightsquigarrow t$. By definition of a cover, $d_i \in S'_i$. Since $S'_i \subseteq S_i$, it holds that $d_i \in S_i$ as well, and hence the assignment $(d_1, \ldots, d_n)$ is an assignment compatible with the path leading to N. Hence the path that follows $N_0 \rightsquigarrow N$ in the search tree and then follows the graph vertices traversed from $N' \rightsquigarrow t$ also contains $(d_1, \ldots, d_n)$ in its cover. Since $|s \rightsquigarrow N \rightsquigarrow t| \leq |s \rightsquigarrow N' \rightsquigarrow t|$, it holds that $J(s \rightsquigarrow N \rightsquigarrow t, d_1, \ldots, d_n) \leq J(s \rightsquigarrow N' \rightsquigarrow t, d_1, \ldots, d_n)$. $\square$

C. *Back-checking search*

Both greedy and exact search must perform and store up to M constraint tests per node. This is wasteful when $M >> n$, many displacements are irrelevant to the final solution, and when constraint tests incur non-negligible expense (e.g., collision detection calls). Back-checking is designed to maintain only those displacements that are *relevant* to the optimal path leading to a particular state. It is often the case that this subset is much smaller than M.

Back-checking procedure. The idea is to search among optimal *assignments* to the path leading to any vertex, and hence states are $N = (v, d_1, \ldots, d_n)$ with each $d_i = \arg\min_{d \in S_i} C[i, d]$ corresponding directly to the compatible cover S_i. Back checking also performs a best first search, but with the added challenge to propagate the optimal assignment from N to a successor state $N' = (w, d'_1, \ldots, d'_n)$.

For each obstacle i the planner must consider two cases. 1) If $w \notin O[i, d_i]$ then $d'_i = d_i$ is the optimal assignment and it is done. 2) Otherwise, it must examine all possible displacements $d \neq d_i$ on the path $N_0 \rightsquigarrow N'$ for compatibility. This process requires *back-checking* along the path to N. To do this efficiently, assume without loss of generality that all displacements are sorted with nondecreasing cost: $C[i, d] \leq C[i, d+1]$ for all d. All displacements $d < d_i$ will have already been found incompatible along the path leading to N (otherwise, this would contradict the assumption that d_i is optimal). So, it start checking displacements $d_{i+1}, d_{i+2}, \ldots, m_i$ and stop when it has found the first one compatible with $N_0 \rightsquigarrow N'$. To avoid extra per-node expense, it caches at each state the tests that are proven to be compatible/incompatible. This prevents it from exceeding M constraint checks per graph vertex, and hence its worst case complexity is no more than that of greedy search.

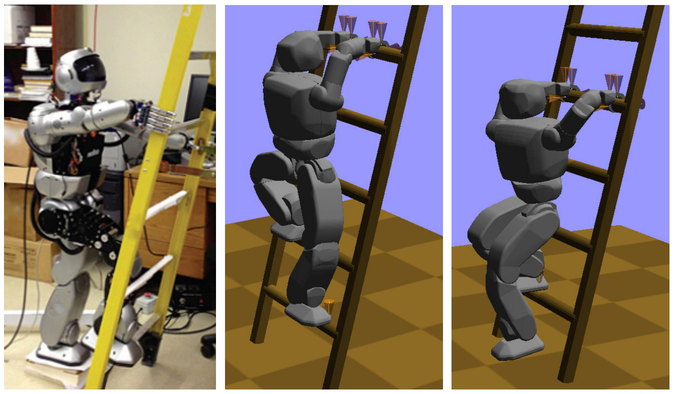


Fig. 8. The Hubo robot posed on a ladder. Due to its short stature it can only achieve this human-like pose by standing on a stepstool. At right, the forward and backward climbing strategies compared in this paper can successfully climb a standard ladder from the point of view of kinematic constraints. However, strong grip forces are required in these two configurations.

Pareto pruning. Back-checking can be configured to prune by cost, and doing so will produce the same output as greedy search. It can also employ *Pareto pruning*, which leads back-checking search to perform largely like exact search. (It can no longer prune via dominance because it cannot be determined without testing *all* constraint displacements.) It does so by storing only those paths that lead to a *Pareto optimal* assignment to *some* vertex in V.

An assignment $(d_1, \ldots, d_n)$ *Pareto dominates* another $(d'_1, \ldots, d'_n)$ if $C[i, d_i] \leq C[i, d'_i]$ for all $i = 1, \ldots, n$. States that are Pareto dominated and do not have a shorter path length are pruned from consideration during forward search. Pareto pruning does produce optimal paths more often than greedy pruning, but there are still cases in which it fails (again, Fig. 7).

Exact search in ordered problems. Call a problem *ordered* if its obstacle function satisfies $O[i, d+1] \subseteq O[i, d]$ and its costs satisfy $C[i, d] \leq C[i, d+1]$ for all i and $d \in \{1, \ldots, m_i - 1\}$. In other words, larger displacements shrink obstacles. Then, back-checking with Pareto pruning is exact: it only needs to search for the first displacement compatible with w because it is guaranteed to be compatible with the path leading to N. Moreover, *back-checking is unnecessary*. Hence, it would appear that the ordering property appears to be quite helpful for tackling huge numbers of obstacles.

V. Application to Humanoid Robot Design

Now consider a challenging humanoid robot design problem. Fig. 8 shows the Hubo+ humanoid posed to climb a ladder, but it is unable to physically do so with its current hardware. Relative to a human, Hubo has weaker grip strength, is shorter (130 cm), has shorter arms, has fewer degrees of freedom in each arm (6 instead of 7DOF), and less flexible

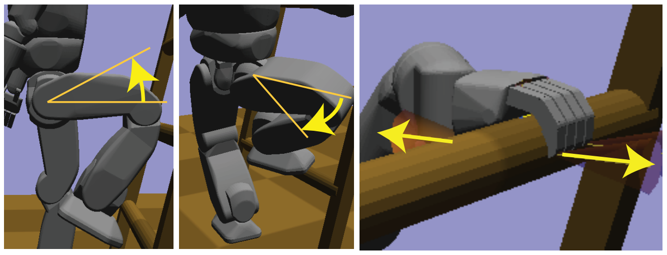


Fig. 9. The three types of constraint displacement considered: hip pitch limit, hip yaw limits, and finger/thumb force limits.

legs (e.g., upper limit of flexion is 90°). Hence, human-like ladder climbing strategies cannot be directly applied. Rather than embarking on a radical redesign, I ask the question, *how little must the robot change so that it can climb a ladder?* Observe that although the parameters of the robot are changed, rather than the environment, these *parameter changes deform the shape of obstacles in C-space*.

To address this problem, I apply the MCD planner to each subsegment of the climbing sequence that maintains a constant set of contacts, hereafter known as a *stance*. These subsegments correspond to either placing a hand/foot or removing a hand/foot. Keeping the robot's fingers fixed, the configuration space $\mathcal{C}$ is 33-D, including 6 "virtual" DOFs for the rigid transform of the robot's base.

Motion constraints. At a stance σ, the robot must satisfy the following motion constraints $\mathcal{F}_\sigma \subset \mathcal{C}$:

- Contact constraints $C_\sigma(q) = 0$ that maintain a constant number of hands and feet in contact against the ladder or ground. These constraints restrict motion to a lower-dimensional submanifold of the configuration space. During sampling, contact constraints are solved using numerical inverse kinematics techniques.
- Joint limits $q_{min} \leq q \leq q_{max}$.
- Collision constraints, both self-collision and between the robot and environment.
- Quasistatic equilibrium of internal and external forces, respecting friction, torque, and grip force limits.

More specifically, quasistatic equilibrium requires gravity to be balanced against contact forces $f_1, \ldots, f_k$ and joint torques τ:

$$G(q) = \tau + \sum_{i=1}^{k} J_{p_i}(q)^T f_i \tag{7}$$

where G is the generalized gravity torque and J_{p_i} is the Jacobian of the i'th contact point. The torque limits must satisfy limits $|\tau| \leq \tau_{max}$, with the inequalities taken element-wise. The 6 components of τ_{max} corresponding to the virtual base DOFs are set to zero. In addition, contact forces must lie in their respective friction cones: $f_i \in FC_i$. These are approximated using polyhedra, and feasible torques and forces $(\tau, f_1, \ldots, f_k)$ are solved via a linear program (LP). The LP is feasible iff the equilibrium constraint is satisfied at q.

Displacements. Consider three types of constraints that would require a tractable amount of mechanical engineering effort to change (Fig. 9):

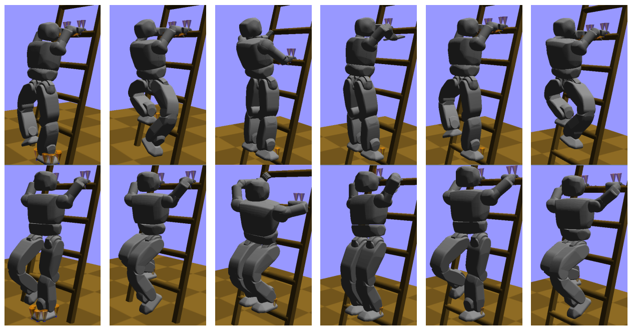
- Increasing hip pitch limits (1 parameter each for L/R). Cost is weighted by 1/5°.
- Increasing hip yaw limits (1 parameter each for L/R). Cost is weighted by 1/5°.
- Increasing finger and thumb strength (2 parameters each for L/R). Cost is weighted by $1\,\mathrm{kg}^{-1}$.

I solve successive MCD problems as follows:

1) Generate stances $\sigma_1, \ldots, \sigma_m$ and *transition configurations* $q_1, \ldots, q_{m-1}$ where q_i satisfies the constraints of both σ_{i-1} and σ_i. These are generated by a simple constrained optimization with randomized restarts. The configurations themselves incur low displacement cost.
2) Call an MCD problem between each pair of configurations q_{i-1}, q_i among the configuration space $\mathcal{F}_{\sigma_i}$ and up to 8 displacement parameters.
3) Take the maxima of each optimized displacement.

On a 75° inclined ladder with cylindrical rungs spaced 30 cm apart, standard forward climbing is unsuccessful because the knees of the robot collide with the rungs. I designed two kinematically-feasible strategies: a forward-facing strategy with splayed feet (FW), and a backward climbing strategy (BW) (Fig 10). Both strategies climb one rung at a time without skipping, and run over 12 stances. MCD is applied to each stance. Running time is between 10 s and 3 min for each stance, and is dominated by collision and equilibrium checking. I also compare a standard sampling-based planner (RRT), with grip force and hip joint displacements set to hypothetical upper limits (10kg and 20° respectively).

Results indicate that grip strength is the bottleneck in climbing, and no other joint limit displacements are needed. Fig. 10 gives maximum grip forces for each strategy during each three-contact stage (four-contact stages are not listed because they exert lower forces on the hands, and stage 6 is dropped because it is identical to stage 2). The RRT solutions exhibit large grip forces, particularly in stage 2. MCD shows with relative certainty that both the FW and BW strategies are



Strat.	Method	1	2	3	4	5	Max
FW	RRT	0	7.2	2.70	0.98	**0.23**	7.2
FW	MCD	0	**2.22**	2.50	0.99	**0.23**	2.50
BW	RRT	0	3.16	**2.37**	1.48	0.51	3.16
BW	MCD	0	2.37	**2.37**	**0.15**	0.45	**2.37**

Fig. 10. Snapshots along the two climbing strategies and grip forces of each stage of the climb, in kg.

actually limited during the one-handed support stage 3, and approximately 2.5kg grip force is sufficient to carry out the optimized motion.

VI. Related Work

This work is most closely related to disconnection proving, excuse-making, and minimum constraint removal problems. Disconnection proving aims to compute a certificate that no path exists given certain constraints [1, 2, 11, 15]. Doing so, however, is computationally demanding, and practical solutions are typically limited to low-dimensional or geometrically simple configuration spaces. Göbelbecker et al. [6] consider excuse-making in symbolic planning problems, where the "excuse" takes the form of changes to the initial state that yield a feasible solution [6]. The minimum constraint removal [7] problem asks to remove the fewest constraints in order to yield a feasible path. Similar work has addressed violating low-priority tasks for multi-objective tasks specified in terms of LTL formulas [3]. Rather than removing constraints in binary fashion, MCD considers continuous changes.

MCD also bears a resemblance to navigation among movable obstacles [13, 14] and manipulation under clutter [4], which require selecting a small set of obstacles for the robot to move and places to put them. Wilfong [14] demonstrated that the decision version of the navigation among movable obstacles problem with polygonal obstacles is NP-hard if the final locations of the obstacles are unspecified, and PSPACE-hard when specified. The approaches of [4, 13] use backward chaining techniques that are often successful in finding a manipulation sequence, but are suboptimal in that too many objects may be selected for manipulation or they may be moved unnecessarily far. This work makes steps toward addressing *global* displacement optimality as well as path optimality. However, unlike backward chaining, MCD does not consider negative interactions between constraints, and combining these approaches may prove fruitful for future work.

VII. Conclusion

This paper described a minimum constraint displacement problem for simultaneous motion planning and displacement cost optimization. It presented algorithms for solving it on discrete graphs and in continuous spaces that are demonstrated to be general to multiple types of displacements (translations, rotations, and shrinking), large numbers of obstacles (100), and many degrees of freedom (33). In future extensions of this work, I hope to handle interacting obstacles, such as obstacles that must themselves be collision free; more general constraints, like differential constraints; more general costs, such as curvature minimization; and more general terminal conditions, such as the goal regions that arise in manipulation problems.

Acknowledgments

Jeff Johnson provided helpful comments on this manuscript. This work is partially supported by NSF Grant # 1218534.

References

[1] J. Basch, L. Guibas, D. Hsu, and A.T. Nguyen. Disconnection proofs for motion planning. In *IEEE Int. Conf. Rob. Aut.*, pages 1765–1772, Seoul, Korea, 2001.

[2] T. Bretl, S. Lall, J.-C. Latombe, and S. Rock. Multi-step motion planning for free-climbing robots. In *Workshop Algo. Found. Robotics*, Zeist, Netherlands, 2004.

[3] L. I. Reyes Castro, P. Chaudhari, J. Tumova, S. Karaman, E. Frazzoli, and D. Rus. Incremental sampling-based algorithm for minimum-violation motion planning, 2013. URL http://arxiv.org/abs/1305.1102. arXiv:1305.1102v1 [cs.RO].

[4] M. R. Dogar and S. S. Srinivasa. A framework for push-grasping in clutter. In *Robotics: Science and Systems*, 2011.

[5] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *J. Algorithms*, 34(2):251 – 281, 2000. ISSN 0196-6774.

[6] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel. Coming up with good excuses: What to do when no plan can be found. In *Int. Conf. on Automated Planning and Scheduling*, 2010.

[7] K. Hauser. The minimum constraint removal problem with three robotics applications. In *Workshop Algo. Found. Robotics*, Boston, MA, 2012.

[8] D. Hsu, J.-C. Latombe, and S. Sorkin. Placing a robot manipulator amid obstacles for optimized execution. In *IEEE Int. Symp. on Assembly and Task Planning*, pages 280–285, 1999.

[9] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems*, Zaragoza, Spain, 2010.

[10] L. E. Kavraki, P. Svetska, Jean-Claude Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. and Autom.*, 12(4):566–580, 1996.

[11] Z. McCarthy, T. Bretl, and S. Hutchinson. Proving path non-existence using sampling and alpha shapes. In *IEEE Int. Conf. Rob. Aut.*, 2012.

[12] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual IEEE Symp. Foundations of Computer Science*, pages 421–427, 1979.

[13] M. Stilman and J.J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *Int. J. Humanoid Robotics*, 2(4), December 2005.

[14] G. Wilfong. Motion planning in the presence of movable obstacles. In *4th Annual Symp. Comp. Geometry*, pages 279 – 288, Urbana-Champaign, IL, 1988.

[15] L. Zhang, Y. Kim, and D. Manocha. A simple path non-existence algorithm using c-obstacle query. In S. Akella, N. Amato, W. Huang, and B. Mishra, editors, *Algorithmic Foundation of Robotics VII*, volume 47 of *Springer Tracts in Advanced Robotics*, pages 269–284. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-68404-6.