# Simultaneous System Design and Path Planning: A Sampling-based Algorithm

**Kevin Molloy[1], Laurent Denarie[1], Marc Vaisset[1], Thierry Siméon [1] and Juan Cortés[1]**

**Abstract**

This paper addresses the simultaneous design and path planning problem, in which features associated to the bodies of a mobile system have to be selected to find the best design that optimizes its motion between two given configurations. Solving individual path planning problems for all possible designs and selecting the best result would be a straightforward approach for very simple cases. We propose a more efficient approach that combines discrete (design) and continuous (path) optimization in a single stage. It builds on an extension of a sampling-based algorithm, which simultaneously explores the configuration-space costmap of all possible designs aiming to find the best path-design pair. The algorithm filters out unsuitable designs during the path search, which breaks down the combinatorial explosion. Illustrative results are presented for relatively simple (academic) robotic examples, showing that even in these simple cases, the computational cost can be reduced by two orders of magnitude with respect to the naïve approach. A preliminary application to challenging problems in computational biology related to protein design is also discussed at the end of the paper.

**Keywords**

robot motion planning, sampling-based algorithms, computational biology, protein design

## 1 Introduction

System design and path-planning problems are usually treated independently. In robotics, criteria such as workspace volume, workload, accuracy, robustness, stiffness, and other performance indexes are treated as part of the system design (Vijaykumar et al. 1986; Gosselin and Angeles 1991; Merlet 2005). Path planning algorithms are typically applied to systems with completely fixed geometric and kinematic features. In this work, we propose an extension of the path planning problem, in which some features of the mobile system are not fixed *a priori*, and can be selected among a finite number of combinations. The goal is to find the best design (i.e. values for the variable features) to optimize the motion between given configurations.

A brute-force approach to solve this problem would involve individually solving motion planning problems for all possible designs, and then selecting the design providing the best result for the (path-dependent) objective function. However, because of the combinatorial explosion, only simple problems involving a small number of variable design features can be treated using this naïve approach. We propose a more sophisticated approach that simultaneously considers system design and path planning. A related problem is the optimization of geometric and kinematic parameters of a robot to achieve a given end-effector trajectory, usually referred to as kinematic synthesis (McCarthy and Joskowitz 2001), or the reachability of desirable goal regions in cluttered workspaces (Baykal et al. 2015; Baykal and Alterovitz 2017). Nonetheless, the problem we address in this work (see Section 2.1 for details) is significantly different, since we assume that all kinematic parameters and part of the geometry of the mobile system are provided as input. The design concerns a discrete set of features that can be associated to the bodies of the mobile system, such as shape or electrostatic charge, aiming to find the best possible path between two given configurations provided a path cost function. Very few works have considered such a hybrid design and path planning problem. One of the rare examples is a recently proposed method for aerial vehicle path planning (Rudnick-Cohen et al. 2015) where the optimal path planning algorithm considers several possible flying speeds and wing reference areas to minimize path risk and time. Since the considered configuration space is two-dimensional, the proposed solution is based on an extension

[1]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

**Corresponding author:**
Juan Cortés, LAAS-CNRS, Toulouse, France,31400.
Email: juan.cortes@laas.fr

of Dijkstra's algorithm working on a discrete representation of the search-space. This type of approach cannot be applied in practice to higher-dimensional problems, as the ones we address.

Sampling-based algorithms have been developed since the late 90s for path planning in high-dimensional spaces (Kavraki et al. 1996; LaValle 2006), which are out of reach for deterministic, complete algorithms. Our work builds on this family of algorithms, which we extend to treat a combinatorial component in the search-space, associated to the systems design, while searching for the solution path.

Our approach presents some similarities with methods that extend sampling-based path planning algorithms to solve more complex problems such as manipulation planning (Siméon et al. 2004), multi-modal motion planning (Hauser and Latombe 2010) or minimum constraint removal planning (Hauser 2014), which also involve search spaces with hybrid structures. As in these other works, the proposed algorithm simultaneously explores multiple sub-spaces, aiming to find solutions more efficiently. Nevertheless, the hybrid design problem addressed here is different.

This paper presents the Simultaneous Design and Path-planning algorithm (SDAP), which is based on the Transition-based Rapidly-exploring Random Tree (T-RRT) algorithm (Jaillet et al. 2010). As explained in Section 2.3, the choice of T-RRT as a baseline is guided by the type of cost function we apply for the evaluation of path quality. Nevertheless, other sampling-based algorithms can be extended following a similar approach. In particular, algorithms with optimality properties such as other variants of Rapidly-exploring Random Tree (RRT*) (Karaman and Frazzoli 2011), Stable Sparse-RRT* (SST*) (Li et al. 2016) or Fast Marching Tree (FMT*) (Janson et al. 2015) could be possible alternatives for some classes of problem.

We demonstrate the good performance of the method on relatively simple, academic examples (Section 4). These simple examples allow us to apply the naïve exhaustive method, whose results can be used as a reference to evaluate the performance and the quality of the solutions produced by the SDAP algorithm. Results show that SDAP is able to find the best path-design pairs, requiring much less computing time than the naïve method. This advantage increases with the complexity of the problem.

Although the application of the proposed approach to problems of practical interest is out of the scope of this paper, we note that our motivation comes from problems in computational biology. For more than a decade, robotics-inspired algorithms have been applied to this area (Moll et al. 2007; Al-Bluwi et al. 2012; Gipson et al. 2012; Shehu 2013). The method presented in this paper aspires to solve problems related to computational protein design (Keating 2013; Donald 2011), which aims to create or modify proteins to exhibit some desired properties. Progress in this field promises great advances in pharmacology, biotechnology, and nanotechnology. Protein design is extremely challenging, and although there have been some considerable strides in the last years (Privett et al. 2012; Tinberg et al. 2013), the problem remains largely open. Current approaches focus on a static picture (i.e. search the amino-acid sequence that stabilizes a given structure), whereas dynamic aspects related to protein function are rarely considered. Our goal behind this work is to develop new methods to optimize functional protein motions. Flexible regions of a protein known as loops play instrumental roles in protein motion, and therefore, protein function. Section 5 presents a preliminary investigation on the application of the proposed method to designing a loop region in order to facilitate the transition of the protein between two stable states. In addition to computational protein design, applications of the proposed approach in robotics can be envisioned and are briefly mentioned in the conclusion (Section 6).

## 2 Problem Formulation and Approach

This section defines the problem addressed in this work, along with some notation, and presents an overview of the proposed approach.

### 2.1 Problem Definition

Let us consider an articulated linkage $\mathcal{A}$ consisting of $n$ rigid bodies, $A_1..A_n$. The kinematic parameters of $\mathcal{A}$ are static and are supplied as input. The geometry of the rigid bodies $A_i$ can admit some variability, as well as other physical properties (mass, electrostatic charge, ...). More precisely, a discrete set of $m$ design features, $f_1..f_m$, is defined and each body $A_i \in \mathcal{A}$ is assigned a design feature $f_j \in \mathcal{F}$. We denote $d$ as a vector of length $n$ that represents the design features assigned to all the rigid bodies in $\mathcal{A}$, i.e. $d$ defines a particular design. $\mathcal{D}$ denotes the set of possible combinations of assignments of features for $\mathcal{A}$, i.e. $\mathcal{D}$ defines all possible designs. $\mathcal{D}$ is referred to as the design space, which is a discrete space containing $m^n$ elements. A given configuration of $\mathcal{A}$ is denoted by $q$. Let $\mathcal{C}$ denote the configuration space. Note that for each $q \in \mathcal{C}$, only a subset of the possible designs $\mathcal{D}$ can be assigned, since some designs are not compatible with some configurations due to self-constraints or environment constraints.

The workspace of $\mathcal{A}$ is constrained by a set of obstacles $O_i \in \mathcal{O}$. Note that the problem formulation could be extended to the design of the workspace, by associating a discrete set of design features to each obstacle. Nevertheless, for the sake of clarity, we consider in this paper that the obstacles have fixed features.

$\mathcal{C}^d_{free}$ denotes all valid collision-free configurations of $\mathcal{A}$ for a given vector $d$ of design features. A path $P$ connecting two configurations $q_{\text{init}}$ and $q_{\text{goal}}$ of $\mathcal{A}$ with design $d$ is defined as a continuous function $P : [0,1] \to \mathcal{C}$, such that $P(0) = q_{\text{init}}$ and $P(1) = q_{\text{goal}}$. The path is said to be collision-free if $\forall t \in [0,1], P(t) \in \mathcal{C}^d_{free}$. $\mathcal{C}_{free}$ is the union of all individual $\mathcal{C}^d_{free}$:

$$\mathcal{C}_{free} = \bigcup_{d \in \mathcal{D}} \mathcal{C}^d_{free} \ .$$

$\mathcal{P}_{free}$ denotes the set of all feasible, collision-free paths connecting $q_{\text{init}}$ to $q_{\text{goal}}$, considering all possible designs ($\forall d \in \mathcal{D}$).

A cost function $c : \mathcal{C}_{free} \times \mathcal{D} \to \mathbb{R}$ associates to each pair $\{q, d\}$ a cost value, $\forall q \in \mathcal{C}_{free}$ and $\forall d \in \mathcal{D}$. We define a *costmap* as the representation of the cost function on the configuration space of $\mathcal{A}$ for each design. Another cost function $c_P : \mathcal{P}_{free} \times \mathcal{D} \to \mathbb{R}$ is defined to evaluate the quality of paths. In this work, the path cost function $c_P$ is itself a function of the configuration cost function $c$, i.e. $c_P$ is a functional. More precisely, we consider the *mechanical work* criterion as defined in (Jaillet et al. 2010; Devaurs et al. 2016) to evaluate paths, which aims to minimize the variation of the configuration cost $c$ along the path. This criterion is a suitable choice to evaluate path quality in many situations (Jaillet et al. 2010), and is particularly relevant in the context of molecular modeling. Nevertheless, other cost functions can be considered, such as the integral of $c$ along the path. A discrete approximation, with constant step size $\delta = 1/l$, of the mechanical work (MW) cost of a path $P$ for a system design $d$ can be defined as:

$$c_P(P,d) = \sum_{k=1}^{l} \max\left\{0\,,\ c\left(P\left(\frac{k}{l}\right),d\right) - c\left(P\left(\frac{k-1}{l}\right),d\right)\right\}. \quad (1)$$

The goal of our method is to find the best pair $\{P^*, d^*\}$ such that:

$$c_P(P^*,d^*) = \min\{c_P(P,d) \mid P \in \mathcal{P}_{free}, d \in \mathcal{D}\}. \quad (2)$$

## 2.2 Illustrative examples

Figure 1 presents a simple example to help understand the problem formulation. In this case, the articulated mechanism $\mathcal{A}$ involves three bodies, in addition to the base and the end effector. The are three possible choices (design features $f_1, f_2, f_3$) for the shape of each body $A_i$. Thus, the number of possible designs is $3^3 = 27$. A cost function $c$ is defined as the inverse of the minimum distance between the robot bodies and the obstacles. With the aim of favoring motions of the robot far from obstacles, $c_P$ can be defined as the integral of $c$ along the path between $q_{\text{init}}$ and $q_{\text{goal}}$. In this simple example, it is easy to figure out that the optimal design is $d^* = [f_1, f_2, f_1]$.

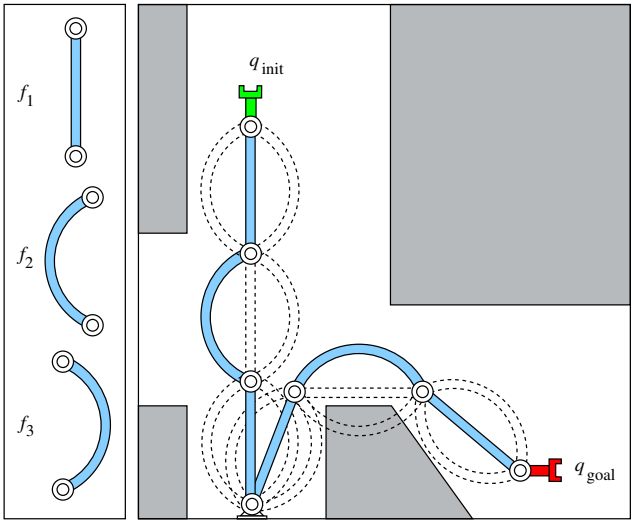


**Figure 1.** Simple illustrative example of a simultaneous system design and path planning problem. There are three possible choices (design features) for each body of the robot. The best design, which optimizes a clearance-based cost function, is represented in blue.

Another simple example, presented in Section 4.1, will be used for a first proof of concept. In this other example, the design features do not involve the shape of the mobile system, but the electrostatic charge of each body. The objective is to minimize the cumulative variation of a potential energy function along the solution path.

## 2.3 Approach

A naïve approach to solve the problem would be to compute the optimal cost path for each design $d \in \mathcal{D}$, and then choose the optimal design $d^*$ that minimizes $c_P$. Such a brute-force approach can be applied in practice to simple problems involving a small number $n$ of variable bodies and/or a few $m$ design features (recall that the design space is size $m^n$). The method proposed below aims to solve the problem much more efficiently by combining both the discrete (design) and continuous (path) optimization in a single stage.

We assume that, for most problems of interest, the configuration space $\mathcal{C}$ is high-dimensional, so that exact/complete algorithms cannot be applied in practice to solve the path-planning part of the problem. For this, we build on sampling-based algorithms (LaValle and Kuffner 2001; Kavraki et al. 1996), which have been very successful in the robotics community since the late 90s, and which have also been applied in other areas such as computational biology (Moll et al. 2007; Al-Bluwi et al. 2012; Gipson et al. 2012; Shehu 2013). We also assume that the cardinality of the design space $\mathcal{D}$ is moderately high,

such that a relatively simple combinatorial approach can be applied to treat the design part of the problem.

The idea is to explore $\mathcal{C}_{free}$ to find paths between $q_{\text{init}}$ and $q_{\text{goal}}$ simultaneously considering all possible deigns $d \in \mathcal{D}$. To reduce the number of configuration-design pairs $\{q, d\}$ to be evaluated during the exploration, it is important to apply an effective filtering strategy. The choice of the particular sampling-based path planning algorithm and filtering strategy mainly depend on the type of objective function $c_P$ being considered. The approach described below has been developed to find good-quality solutions with respect to the MW path evaluation criterion (1). In this work, we extend the T-RRT algorithm (Jaillet et al. 2010), which finds paths that tend to minimize cost variation by filtering during the exploration tree nodes that would produce a steep cost increase. Following a similar approach, alternative algorithms and the associated filtering strategies could be developed to optimize other path cost functions. For instance, variants of RRT* (Karaman and Frazzoli 2011) or FMT* (Janson et al. 2015) could be considered for optimizing other types of monotonically increasing cost functions.

## 3 Algorithm

This section presents the SDAP algorithm, building on a single-tree version of T-RRT. However, the approach is directly applicable to multi-tree variants (Devaurs et al. 2014). First, the basic algorithm is introduced, followed by additional explanation of the tree extension strategy and a brief theoretical analysis.

### 3.1 SDAP Algorithm

The SDAP pseudo-code is shown in Algorithm 1. A search tree, $\mathcal{T}$, is created with $q_{\text{init}}$ as the root node. The tree is grown in configuration space through a series of expansion operations. Each node $s$ in $\mathcal{T}$ encodes a configuration $q$ and a set of designs $D \subseteq \mathcal{D}$ for which the configuration is valid, and which have not been filtered out in previous iterations for the construction of corresponding branches of the tree. In other words, each node's set of designs $D$ is a subset of its parent's designs, i.e. $Designs(s) \subseteq Designs(Parent(s))$. This is essential to reduce the number of designs evaluated during the exploration. The algorithm keeps iterating until the goal configuration $q_{\text{goal}}$ is reached for at least one of the designs, or a maximum number of iterations is performed.

At each iteration, a random configuration $q_{\text{rand}}$ is generated (line 3). In T-RRT, a new node $q_{\text{new}}$ is created by expanding the nearest node in $\mathcal{T}$ $q_{\text{near}}$ in the direction of $q_{\text{rand}}$ for a distance $\delta$. $q_{\text{new}}$ is then conditionally added to $\mathcal{T}$ based on a transition test. A common heuristic for this transition test is the Metropolis criterion, traditionally

**Algorithm 1:** SDAP Algorithm

**input :** the configuration space $\mathcal{C}$; the design space $\mathcal{D}$; the cost function $c$; the start state $q_{\text{init}}$; the goal state $q_{\text{goal}}$; maximum number of iterations *MaxIter*
**output:** the tree $\mathcal{T}$

```
1  T ← InitTree(q_init, D)
2  while not StoppingCriterion (T, q_goal, MaxIter) do
3      q_rand ← Sample (C)
4      Neighbors ← NearestNeighbors(T, q_rand, D)
5      TransitionTest.Init()
6      for s_near ∈ Neighbors do
7          q_new ← Extend(q_rand, s_near)
8          D ← TransitionTest(T, s_near, q_new, c)
9          if NotEmpty (D) then
10             AddNode(T, s_near, q_new, D)
```

applied in Monte Carlo methods (Metropolis et al. 1953). Transitions to lower cost nodes are always accepted and moves to higher costs nodes are probabilistically accepted. The probability to transition to a higher cost node is controlled by a temperature variable $T$. T-RRT dynamically controls $T$, as explained in the next section.

SDAP modifies the expansion and transition test functions of the standard T-RRT algorithm in order to address the design and path planning problems simultaneously. At each iteration, SDAP attempts to expand at least one node per design in $\mathcal{D}$. The process is shown in Figure 2, where each design $d \in \mathcal{D}$ is encoded as a color on each node of the tree. During each iteration, SDAP expands a set of nodes that covers all designs $\mathcal{D}$. In other words, the `NearestNeighbors` function (line 4) returns a set, *Neighbors*, containing the closest node to $q_{\text{rand}}$ for each design. In Figure 2, $q_{\text{rand}}$ is shown in black and the 3 nodes in the set *Neighbors* are circled in red. Each node in *Neighbors* is extended towards $q_{\text{rand}}$ (lines 6 - 10) creating new candidate nodes which are labeled $s_1$, $s_2$ , and $s_3$ in Figure 2. All 3 designs in $s_1$ fail the transition test, so the new node is not added to $\mathcal{T}$. For $s_2$ the blue design passes the transition test and the node is added to $\mathcal{T}$. Finally for $s_3$, 1 of the 3 designs (yellow) fails the transition test, resulting in a node with 2 designs being added to $\mathcal{T}$. Such a filtering of designs during the construction of the exploration tree is the key to SDAP's good performance.

### 3.2 Controlling Tree Expansion

The transition test is governed by the temperature parameter $T$. T-RRT automatically adjusts $T$ during the exploration and has been shown highly effective in balancing tree exploration and tree refinement (Jaillet et al. 2010). At each iteration, T-RRT adjusts $T$ by monitoring the acceptance rate of new nodes. SDAP extends this idea by maintaining a separate temperature variable $T(d)$ for each design $d \in \mathcal{D}$.

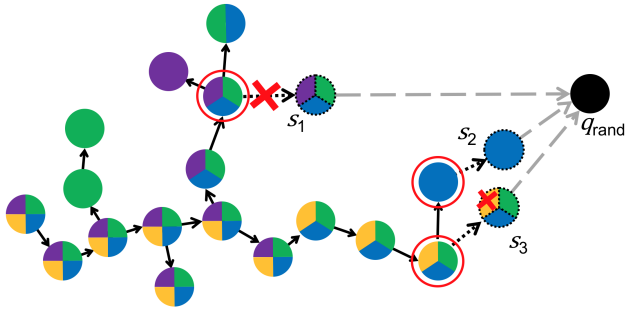


**Figure 2.** An expansion operation for SDAP. Designs are encoded as colors within each node. The nodes being expanded are circled in red. The expansion towards node $s_1$ fails for all 3 designs, The expansion to $s_2$ succeeds, and the expansion to $s_3$ succeeds for 2 of the 3 designs.

A given design $d$ can appear in multiple nodes in *Neighbors*. For each design $d$, the node in *Neighbors* closest to $q_{\text{rand}}$ is identified. The temperature $T(d)$ is adjusted based on the success or failure of the extension operation from this node for design $d$.

The pseudocode for the transition test function is shown in Algorithm 2. The *Neighbors* set is processed in ascending order of the distance of each node from $q_{\text{rand}}$ (line 6 of Algorithm 1). For the node being expanded ($s_{\text{near}}$), each design $d$ has its cost evaluated (line 4). Transitions to lower cost nodes are always accepted (line 8). Transitions to higher cost nodes are subjected to probabilistic acceptance (line 10). The set $V$ (lines 12 and 18) tracks designs which have had their temperature adjusted during this iteration. The function returns the set $D$ of designs that pass the transition test.

**Algorithm 2:** TransitionTest($\mathcal{T}$, $s_{\text{near}}$, $q_{\text{new}}$, $c$)

**input** : the input tree $\mathcal{T}$; vector of temperatures T; parent node $s_{\text{near}}$; new node $q_{\text{new}}$; the cost function $c$; temperature adjustment rate $T_{\text{rate}}$; Boltzmann constant $K$
**internal:** set of designs $V$ with adjusted temperatures
**output** : vector of designs $D$ that pass the transition test

1 S $\leftarrow \phi$
2 **for** $d \in Designs(s_{\text{near}})$ **do**
3  **if** `CollisionTest`$(q_{\text{new}}, d)$ == *False* **then**
4  $c_{\text{near}} = c(\text{Config}(s_{\text{near}}), d)$; $c_{\text{new}} = c(q_{\text{new}}, d)$
5  success $\leftarrow$ false
6  $\Delta c$ = $c_{\text{new}}$ - $c_{\text{near}}$
7  **if** $\Delta c < 0$ **then**
8  success $\leftarrow$ true
9  **else**
10  **if** $exp(-\Delta c \,/\, (K \cdot T(d))) >$ `UniformRand()` **then**
11  success $\leftarrow$ true
12  **if** $d \notin V$ **then**
13  **if** *success* **then**
14  $T(d) \leftarrow T(d) \,/\, 2^{(\Delta c \,/\, \text{energyRange}(\mathcal{T}, d))}$
15  **else**
16  $T(d) \leftarrow T(d) \cdot 2^{T_{\text{rate}}}$
17  **if** *success* **then** $D \leftarrow D \cup d$
18  V $\leftarrow V \cup d$
19 `return` $(D)$

## 3.3 Theoretical Analysis

In this section we provide some theoretical analysis of the SDAP algorithm's completeness and path optimality. A theoretical analysis of the complexity of SDAP with respect to the brute-force approach is difficult, since both are stochastic processes. In this work, we instead provide empirical results in Section 4 that clearly show SDAP's efficiency versus an exhaustive search of paths for all possible designs.

*3.3.1 Probabilistic Completeness:* SDAP's probabilistic completeness directly derives from that of RRT (LaValle and Kuffner 2001), which is inherited by T-RRT under the condition to guarantee a strictly positive probability of passing the transition test as explained in (Jaillet et al. 2010). Since SDAP maintains this property by incorporating temperatures in the transition test for each given design $d \in \mathcal{D}$, it also ensures the positive transition probability and that each $\mathcal{C}^d_{free}$ will be completely sampled, thus maintaining the probabilistic completeness of the algorithm.

*3.3.2 Path Optimality:* The current SDAP implementation is based on T-RRT, which has been empirically shown to compute paths that tend to minimize cost with respect to the MW criterion (Jaillet et al. 2010), but without theoretical guarantee of optimality. Using anytime variants of T-RRT (AT-RRT or T-RRT*) (Devaurs et al. 2016) would provide an asymptotic convergence guarantee. Implementing these within SDAP remains as future work.

# 4 Empirical Analysis and Results

As a proof of concept, we apply SDAP to a set of academic problems. SDAP is implemented as an adaptation of the Multi-T-RRT algorithm (Devaurs et al. 2014), with two trees growing from the initial and goal configurations. The search stops when the algorithm is able to join the two trees. For each problem, SDAP is compared against a naïve approach consisting of multiple independent runs of Multi-T-RRT on each design $d \in \mathcal{D}$.

## 4.1 Test System Description

The test system is a 2D articulated mechanism with a fixed geometry surrounded with fixed obstacles. The bodies

$A_1..A_n$ are circles with radius $R$. The first body $A_1$ is a fixed base. The other bodies $A_2..A_n$ are articulated by a rotational joint centered on the previous rigid body that can move in the interval $[0, 2\pi)$. A configuration $q$ is described by a vector of $n-1$ angles corresponding to the value of each rotational joint. The features $f_1..f_n$ assigned to each body are electrostatic charges in $\mathcal{F} = \{-1, 0, 1\}$ (i.e. $m = 3$). The design vector $d$ contains $n$ charges $f_1..f_n$ associated to each rigid body $A_1..A_n$ of the mechanism. In the following, $d$ will be written as a string, with each charge $\{-1, 0, 1\}$ corresponding to N, U, and P respectively. For example, the design NPUN corresponds to the vector $d = [-1, 1, 0, -1]$. Obstacles $O_1..O_k$ are circles of radius $R$ and have electrostatic charges with predefined values $g_i \in \mathcal{F}$.

The cost function is inspired by a simple expression of the potential energy of a molecular system. It contains two terms, one corresponding to the Lennard-Jones potential and the other to the electrostatic potential. It is defined as:

$$c(q, d) = LJ(q, d) + ES(q, d) \tag{3}$$

with:

$$LJ(q,d) = \sum_{i=1}^{|\mathcal{A}|-2}\left[\sum_{j=i+2}^{|\mathcal{A}|}\left(\frac{2\cdot R}{\|A_iA_j\|}\right)^{12} - \left(\frac{2\cdot R}{\|A_iA_j\|}\right)^{6}\right] + \sum_{i=1}^{|\mathcal{A}|}\left[\sum_{j=1}^{|\mathcal{O}|}\left(\frac{2\cdot R}{\|A_iO_j\|}\right)^{12} - \left(\frac{2\cdot R}{\|A_iO_j\|}\right)^{6}\right] \tag{4}$$

$$ES(q,d) = \sum_{i=1}^{|\mathcal{A}|-2}\left[\sum_{j=i+2}^{|\mathcal{A}|}\left(\frac{f_i\cdot f_j}{\|A_iA_j\|}\right)\right] + \sum_{i=1}^{|\mathcal{A}|}\left[\sum_{j=1}^{|\mathcal{O}|}\left(\frac{f_i\cdot g_j}{\|A_iO_j\|}\right)\right] \tag{5}$$

where $\|X_iX_j\|$ represents the Euclidean distance between the centers of the bodies/obstacles $X_i$ and $X_j$.

SDAP is empirically tested using several scenarios with the objective to find the optimal path-design pair $(P^*, d^*)$. First, a simple 4-body system is used to illustrate the effect of the design choice and to discuss the optimality of the solution. Then, two more complex systems, one involving a variable number of bodies to be designed and the other involving a variable number of degrees of freedom (DOF), are used to analyze the effect of the complexity of the problem on the performance of the algorithm.

*4.1.1 4-Body System:* The first system consists of four bodies and five obstacles as shown in Figure 3 (top). $q_{\text{init}}$ and $q_{\text{goal}}$ correspond to fully stretched configurations, to the left and to the right, represented with solid and dashed outlines respectively. The design space consists of $3^4 = 81$ possible combinations and the configuration space is 3 dimensional. Owing to the position of the positively and negatively charged obstacles, this scenario favors designs involving negative charges. However, the

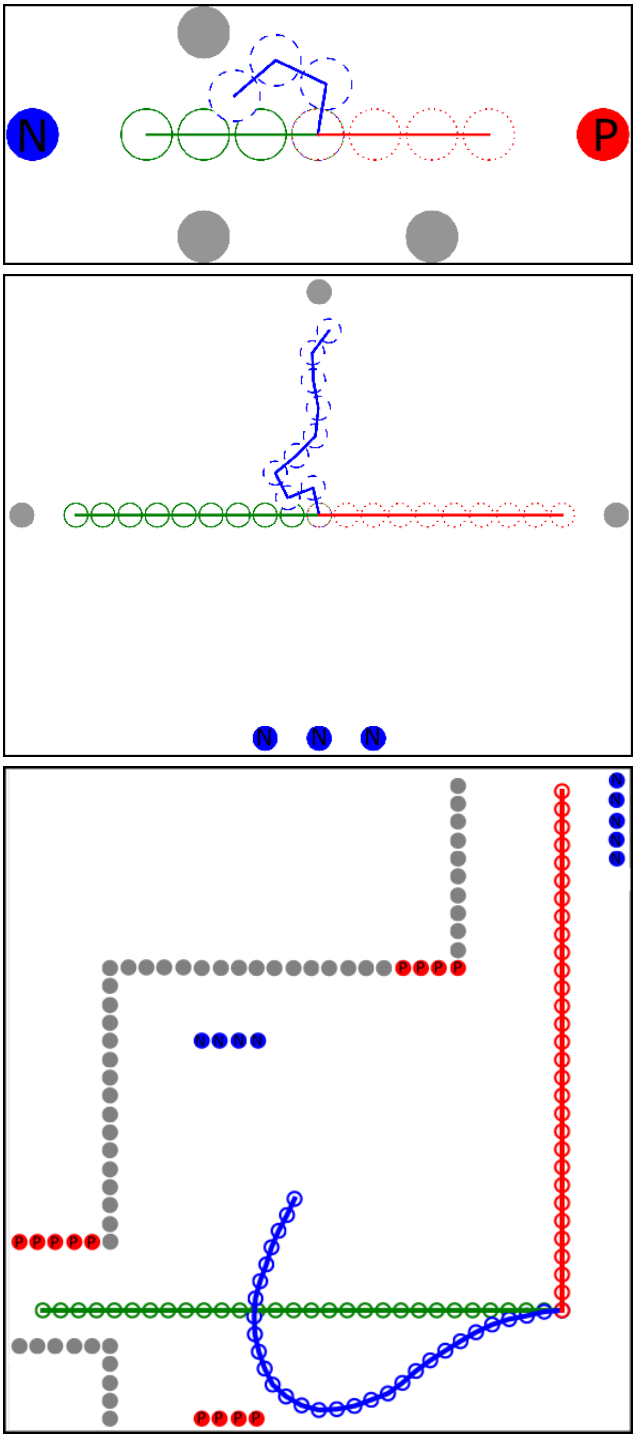


**Figure 3.** A 4-body (top), 10-body (middle) and 30-body (bottom) scenarios. Obstacles with positive charges (labeled P) are shown in solid red, negative (labeled N) in solid blue, neutral in gray (unlabeled). The initial state is shown in green with a solid line, a transition state shown in blue, and the goal state is shown in red with a dashed line.

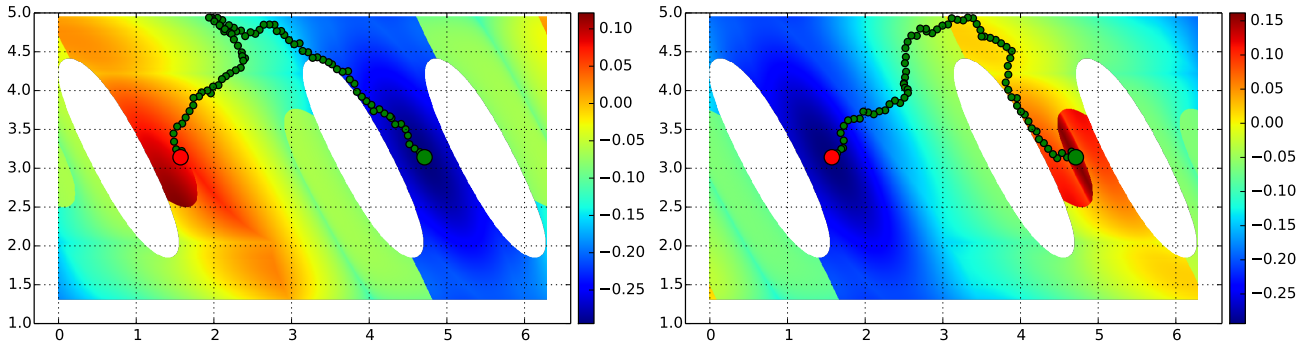

**Figure 4.** Configuration-space costmap of the 4-body system projected onto the first two DOF of the system expressed in radians. The initial configuration is indicated by the red dot on the left, and the goal by the green dot on the right. The plot on the left is for the UUUP design, the one to the right for UUUN. Each cell's cost is computed by finding the value of the third angle that minimizes the cost.

uncharged obstacles at the top and bottom of the workspace create a narrow passage that enforces the articulated chain to fold up, which penalizes designs with negative charges everywhere, owing to their repulsion. Therefore, this is a non-trivial design problem.

*4.1.2 10-Body System:* A larger system with 10 bodies and six obstacles is shown in Figure 3 (middle). The design space $\mathcal{D}$ contains $3^{10} = 59049$ possibilities, which cannot be exhaustively explored within a reasonable computing time, and is also challenging for SDAP because of memory issues (further discussed below). For that reason, three simplified versions of this scenario are constructed. The first one fixes the design for the first seven bodies $A_1..A_7$ as UUUUUUU. The remaining bodies ($A_8$, $A_9$, and $A_{10}$) can be designed, resulting in a design space of $3^3 = 27$ designs. The second version expands the design space to the last 4 bodies ($A_7..A_{10}$) , resulting in a design space of $3^4 = 81$ designs. The last case expands the design space to the last 5 bodies ($A_6..A_{10}$), resulting in a design space of $3^5 = 243$. In all cases, the configuration space is 9 dimensional. All versions of the 10-body system are constrained with the same obstacles. They were chosen so that designs with strongly positively or negatively charged end-effectors will be trapped at local minima resulting from attractive or repulsive forces generated by the bottom obstacles.

*4.1.3 30-Body System:* A third scenario, presented in Figure 3 (bottom), involves a mobile system with up to 30 bodies. This example is used to analyze the influence of the dimension of the configuration space on the performance of SDAP. For this, a variable number of articulated joints is considered. More precisely, rotational joints are considered for all the bodies, or only for one every 2, 3 and 6 bodies. Thus, the dimension of $\mathcal{C}$ is 30, 15, 10 and 5, respectively. As for the 10-body system, the design of the first bodies in the chain is fixed to U. Only the last 4 bodies ($A_{27}..A_{30}$) are designed.

## 4.2 Benchmark Results

We compare SDAP to a naïve approach (solving individual problems for each design $d \in \mathcal{D}$) using the same Multi-T-RRT implementation. In other words, we compare one run of the SDAP algorithm against $|\mathcal{D}|$ runs of a single-design path search. Multiple runs are performed (100 for the 4-body scenario, 50 for the 10-body scenario with 3 designed bodies, 20 for the 10-body scenario with 4 and 5 designed bodies, and 10 for the variants of the 30-body scenario) to reduce statistical variance inherent with stochastic methods. The single-design explorations can spend time trying to escape local minima associated with the costmaps of unfavorable designs, causing very long execution times and high-cost paths. As we are not interested in finding a solution path for every possible design but only for the designs with low-cost paths, a timeout is enforced for the single-design explorations of 5 minutes for the 4-body scenario, and 15 minutes for the 10-body scenarios. For the 30-body problem, the differences in the dimensionality of the configuration space mandated different timeouts of 15, 30, 30, and 35 minutes for the 5, 10, 15, and 30 DOF scenarios respectively. No timeout was considered for SDAP (i.e. the algorithm always provided a solution). In all the cases, the T-RRT parameter $T_{\text{rate}}$ was set to 0.1, being the initial temperature $T = 1$ and the Boltzmann constant $K = 0.00198721$ (note that $T$ and $K$ do not have much physical meaning in the present context). During the exploration, the maximal values reached by $T$ were around 100, being slightly lower for SDAP compared to the naïve approach. All the runs were performed in a single threaded process on a Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz processor with 32GB of memory.

*4.2.1 Results for the 4-Body Scenario:* We did a simple experiment aiming to show how strongly the choice of the design may affect configuration-space costmap. Figure 4 shows a projection of the configuration-space costmap

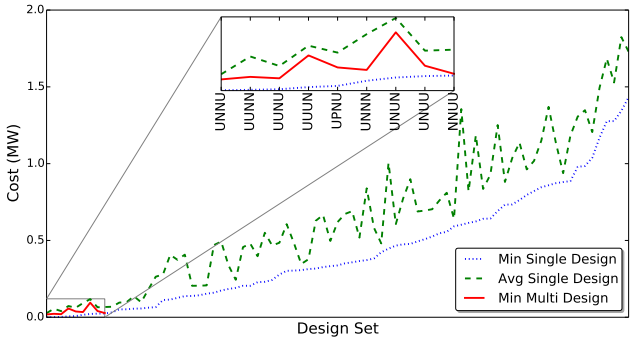


**Figure 5.** Cost of the paths for the best designs found with the naïve approach (dotted blue and dashed green lines) and SDAP (red line) for the 4-body scenario. SDAP solutions are shown only for discovered paths (does not exhaustively search). SDAP discovers the same low-cost designs as the exhaustive single-design searches.

for two designs along with a solution path. One design has a negatively charged end-effector (UUUN) and one has a positively charged end-effector (UUUP). Angles 1 and 2 are projected onto the $x$ and $y$ axis respectively, with angle 3 being set to minimize the cost function. For the UUUN design, the costmap is highly favorable to the desired motion, starting at a high-cost area and proceeding downhill to a low cost area. The costmap associated with the UUUP design shows a non-favorable motion between the two states.

Figure 5 presents results concerning the optimal design obtained by the naïve approach and by SDAP. Both algorithms provide very similar results (see the enlarged region in the figure). The best designs mainly involve negative and uncharged bodies, the best one being UNNU. As expected, owing to repulsive forces between the bodies of the mobile system, designs involving more than two nonconsecutive negative bodies are worst than those containing only one or two negative charges. In terms of computational performance, the naïve approach required $2 \pm 0.1$ minutes to compute the paths for the 81 designs individually, whereas SDAP required $1 \pm 0.6$ minutes. This represents a speed-up of 2 in favor of SDAP. As shown for the more complex examples below, the performance gain significantly increases when the dimensions of the design space and the configuration space increase. Regarding the number of nodes in the T-RRT trees, the 81 runs for the single-design searches generated approximately 77,000 nodes in total (i.e. around 1,000 nodes per design on average). SDAP generated approximately 39,000 nodes for the simultaneous exploration of all the designs, which is directly reflected with the increase in computational efficiency. Note that the high density of nodes created by the SDAP algorithm could be well exploited to improve the path cost by incrementally adding cycles in an any-time fashion, as proposed by Devaurs et al. (2016). Such an extension of the algorithm, which remains for future work (see Section 6), would guarantee asymptotic convergence to the global optimum.

*4.2.2 Influence of the cardinality of the design space:* We used the 3 variants of the 10-body scenario to analyze the influence of the cardinality of the design space on the performance of the SDAP algorithm. Results are summarized in Table 1 and Figure 6. The speedup reported in the right column of the table clearly show that the difference in computing time between SDAP and the naïve approach increases significantly compared to the 4-body scenario. In the 3-designed-body problem, SDAP is 26 times faster than the single-design search on average. In the 4 and 5-designed-body cases, SDAP is 47 and 61 times faster respectively. While the cardinality of design space $\mathcal{D}$ is multiplied by 3 between each of the versions, the average execution time of SDAP only grows by a factor of 2.5 and 2.2 respectively. SDAP achieves this result by limiting the growth of the T-RRT search tree with respect to the size of $\mathcal{D}$. The numbers in Table 1 show that the naïve approach yields a search tree scaling closely with the size of $\mathcal{D}$, which directly correlates with higher computational requirements.

| $\|D\|$ | $\|\mathcal{D}\|$ | Design GF | Naïve | | | | SDAP | | | | SDAP Speedup |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Nodes (x $10^3$) | Node GF | Time (min) | Time GF | Nodes (x $10^3$) | Node GF | Time (min) | Time GF | |
| 3 | 27 | – | $359 \pm 34$ | – | $147 \pm 24$ | – | $79 \pm 30$ | – | $6 \pm 4$ | – | 24.5 |
| 4 | 81 | 3 | $1{,}204 \pm 63$ | 3.4 | $658 \pm 62$ | 4.5 | $205 \pm 52$ | 2.6 | $14 \pm 8$ | 2.3 | 47.0 |
| 5 | 243 | 9 | $3{,}798 \pm 119$ | 10.5 | $2{,}157 \pm 112$ | 14.7 | $432 \pm 103$ | 5.5 | $35 \pm 12$ | 5.8 | 61.6 |

**Table 1.** For each variant of the 10-body scenario, the table shows number of design bodies (D), and the size of the design space ($\mathcal{D}$) for experiments allowing 3, 4, and 5 design positions. The average size of the search tree (expressed in thousands) and the wall clock execution time (minutes) are shown along with their standard deviations. The design space growth factor (GF) is shown along with the GF for each average tree sizes and run times (growth from the 3 design position problem). The tree size for the naïve approach follows the growth in the design space, where the SDAP tree grows at a much slower rate. The final column shows the speed up factor between the naïve approach and SDAP.

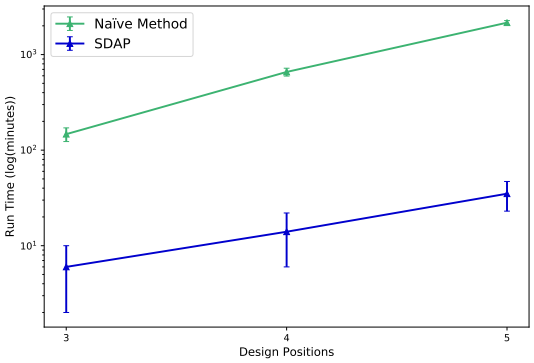


**Figure 6.** Evolution of the computing time (in logarithmic scale) with respect of the cardinality of the design space for the naïve approach and for SDAP.

Importantly, as in the case of the 4-body problem, SDAP successfully identifies the designs yielding the best path costs. Figure 7 compares the solutions found by SDAP to those obtained by the naïve approach.

Note that our initial implementation of SDAP employs a brute force nearest neighbor search with $\mathcal{O}(n)$ complexity. The number of average nodes shown in Table 1 for the naïve approach represents the total of a single set of runs to cover each design in $\mathcal{D}$. However, each individual execution produces a search tree containing 16,000 nodes on average for the 5-designed-body problem. The SDAP search tree for the same problem contains 432,000 nodes on average. Thus, a more efficient nearest neighbor search method would certainly further improve SDAP's relative performance.

The memory requirement for SDAP is directly correlated to the size of the search tree. Each node in the search tree takes on average 37 kB of memory for the 10-body scenarios. For the 5-designed-body problem, SDAP required approximately 16 GB of memory (this also involves base data structures used for fast collision detection and distance computation, in addition to the system representation). Extrapolating from the data in Table 1, each additional body in the design space doubles the size of the search tree, and thus, doubles the memory requirements. We further discuss this in Section 6, and suggest some ideas to circumvent this issue.

*4.2.3 Influence of the dimension of the configuration space:* The 4 variants of the 30-body scenario, involving a variable number of DOF, were used to analyze the influence of the dimensionality of the configuration space on the performance of the SDAP algorithm. Results are summarized in Table 2. The numbers in this table (see also Figure 8) show that performance gain of SDAP with respect to the naïve approach increases with the complexity of the problem. The improvement is very significant at the beginning, when changing from 5 DOF to

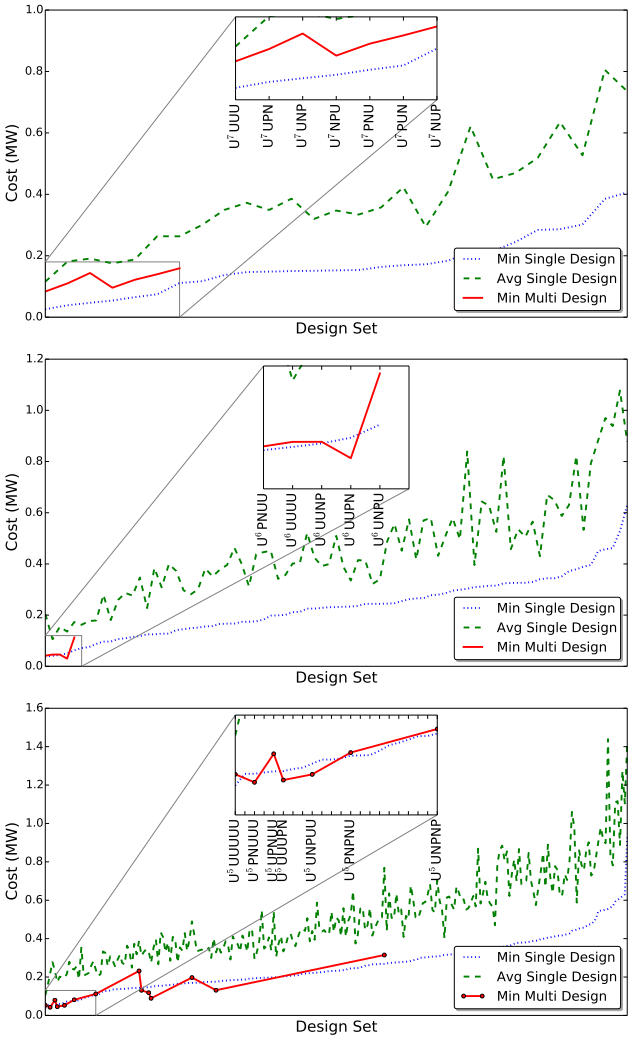


**Figure 7.** Cost of the paths for the best designs found with the naïve approach (dotted blue and dashed green lines) and SDAP (red line) for the three variants of the 10-body problem: 3-designed-body problem (top), 4-designed-body problem (middle), 5-designed-body problem (bottom). SDAP solutions shown only for discovered paths (does not exhaustively search). SDAP discovers the same low-cost designs as the exhaustive single-design searches.

10 DOF, and tends to decrease for larger dimensions. Note however that the speedups reported in higher dimensions are underestimated due to use of a timeout for the single-design runs of the naïve approach. As with the previous experiments, the best designs obtained by SDAP remain very similar to those obtained with the naïve approach.

As illustrated in Figure 9, which summarizes results about the influence of the problem complexity on the SDAP performance versus the naïve approach, the speedup tends to increase with both the cardinality of the design space and the dimension of the configuration space. It is important to

| $\|DOF\|$ | $\|DOF\|$ GF | Naïve Nodes (x $10^3$) | Naïve Node GF | Naïve Time (min) | Naïve Time GF | SDAP Nodes (x $10^3$) | SDAP Node GF | SDAP Time (min) | SDAP Time GF | SDAP Speedup |
|---|---|---|---|---|---|---|---|---|---|---|
| 5 | – | $449 \pm 50$ | – | $104 \pm 23$ | – | $80 \pm 18$ | – | $4 \pm 1$ | – | 26 |
| 10 | 2 | $996 \pm 84$ | 2.2 | $527 \pm 58$ | 5.0 | $98 \pm 19$ | 1.2 | $5 \pm 1$ | 1.2 | 105 |
| 15 | 3 | $1{,}152 \pm 146$ | 2.6 | $614 \pm 100$ | 5.9 | $113 \pm 18$ | 1.4 | $5 \pm 1$ | 1.2 | 123 |
| 30 | 6 | $1{,}912 \pm 99$ | 4.3 | $1{,}511 \pm 100$ | 14.5 | $169 \pm 68$ | 2.1 | $11 \pm 8$ | 2.8 | 137 |

**Table 2.** For the 30-Body scenario, the table shows results for different dimensions of the configuration-space, involving systems with 5, 10, 15 and 30 DOF. The average size of the search tree (expressed in thousands) and the wall clock execution time (minutes) are shown along with their standard deviations. The configuration space growth factor (GF) is shown along with the GF for each average tree sizes and run times (growth from the 5 DOF problem). The run times for SDAP grow at a slower rate than for the naïve approach, as shown in the speed up factors of the last column.

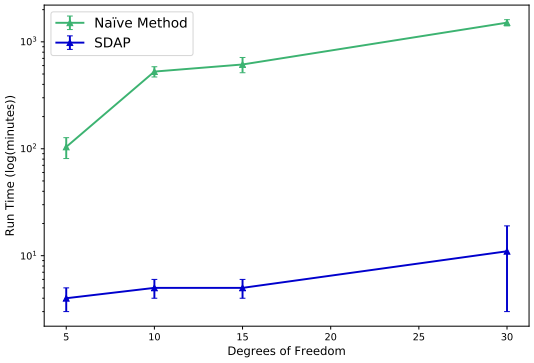


**Figure 8.** Evolution of the computing time (in logarithmic scale) with respect of the dimension of the configuration space for the naïve approach and for SDAP.

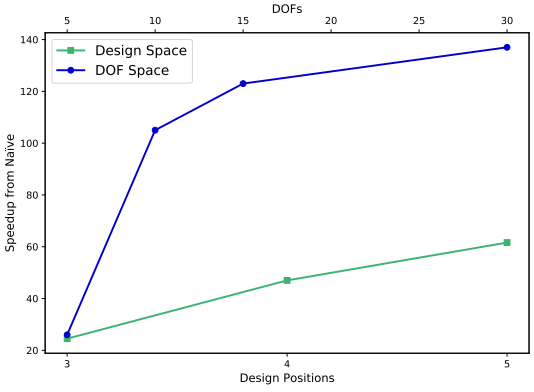


**Figure 9.** The relative performance of SDAP over the naïve approach. The green line shows this with respect to the size of design space using the 10-body problem as benchmark. The blue line showcases the performance varying the number of DOF using the 30-body scenario.

mention that, for the 10-body problem, on average the naïve approach reached the timeout 15%, 23%, 30% of the time for the 3, 4, and 5-designed body problems respectively. For the 30-body problem, the timeout was reached 1%, 31%, 36% and 53% of the time for the 5, 10, 15, and 30 DOF experiments. This large number of failures when computing paths with the naïve approach means that many runs would have needed a longer time to eventually find a solution. Thus, larger timeout values would have certainly led to an even more impressive speedup in favor of SDAP.

## 5 A First Application to Protein Motion Design

Results presented in the previous section show the good performance of SDAP to solve relatively simple problems. This section deals with a more difficult problem related to protein motion design. First, the problem is formulated as a simultaneous design and path planning problem. Some simplifications are required in order to enable the application of the current version of the SDAP algorithm. A first application in this context is attempted for the design of protein region (a loop) aiming to facilitate its transition between two stable states.

### *5.1 Problem Definition*

The protein motion design problem can be formulated as follows: given two protein scaffold conformations* $q_1$ and $q_2$, find a sequence of $n$ amino acids with (meta-)stable states for both structures, and favoring the conformational change (i.e. the motion) between them. Although existing protein design methods may identify sequences for which the two given conformations ($q_1$ and $q_2$) correspond to (meta-)stable states, these methods do not consider the feasibility of the transition path between these two states. In order to design a candidate protein for this motion, it is necessary to find a tuple (path between $q_1$ and $q_2$, sequence

*In this paper, the term conformation is used to define a spatial arrangement of the atoms of the molecule. It is the equivalent to the term configuration for a robot.

of amino acids) that minimizes energy variation (i.e. the mechanical work) along the transition path.

For this preliminary application of the SDAP algorithm to protein motion design, we consider the coarse-grained model proposed by Brown et al. (2003) (usually called BLN model) to limit the combinatorial complexity of the design space. More precisely, the 20 natural amino acids are grouped into 3 classes according to their chemical properties: *B* for the hydrophobic, *L* for the hydrophilic, and *N* for the neutral (see Table 3). This simplification reduces the cardinality of the design space to $3^n$ possible designs (instead of $20^n$). Furthermore, each amino acid (i.e. each body $A_1, \ldots, A_n$) is modeled as a single bead centered on its $C_\alpha$ atom. Consecutive beads are linked by virtual bonds. This simplification allows us to neglect additional degrees of freedom associated with the protein side-chains. The configuration cost function corresponds to the potential energy, defined as:

$$\begin{aligned} c(q,d) = & \sum_{\theta} \frac{1}{2} k_\theta (\theta - \theta_0)^2 \\ & + \sum_{\phi} \Big[ A(1+\cos\phi) + B(1-cos\phi) \\ & + C\,(1+\cos 3\phi) + D\left(1+\cos\left[\phi+\frac{\pi}{4}\right]\right)\Big] \\ & + \sum_{i,j\geq i+3} 4S_1 \left[ \left(\frac{\sigma}{r_{ij}}\right)^{12} - S_2 \left(\frac{\sigma}{r_{ij}}\right)^{6} \right] \end{aligned} \tag{6}$$

In this unitless equation, the first term is a bond angle energy term, where $\theta$ is the angle formed by two consecutive virtual bonds, $k_\theta$ is a force constant of 20 $rad^{-2}$, and $\theta_0$ is a reference angle of 105°. The second term is a dihedral angle energy term, where $\phi$ is the dihedral angle formed by three consecutive virtual bonds, $\sigma$ is the distance unit ($\sigma = 1$Å), and $A$, $B$, $C$, $D$ are parameters varying as a function of the local secondary structure (computed using DSSP (Kabsch and Sander 1983; Touw et al. 2015)):

- $A = 0, B = C = D = 1.2$ for an helical structure,
- $A = 0.9, B = D = 0, C = 1.2$ for a strand,
- and $A = B = D = 0, C = 0.2$ for other structures.

The last term considers pair-wise non bonded interactions, where $r_{ij}$ is the distance between the $i^{th}$ and the $j^{th}$ $C_\alpha$ atoms in the sequence, and $S_1$ and $S_2$ are parameters that depend on the designed features associated to the pair of amino acid residues: $S_1 = S_2 = 1$ for B-B interactions, $S_1 = \frac{1}{3}$ and $S_2 = -1$ for L-L and L-B interactions, and $S_1 = 1$ and $S_2 = 0$ for N-L, N-B, and N-N interactions.

| 20 | 3 | 20 | 3 | 20 | 3 | 20 | 3 |
|---|---|---|---|---|---|---|---|
| Ala | B | Met | B | Gly | N | Asn | L |
| Cys | B | Val | B | Ser | N | His | L |
| Leu | B | Trp | B | Thr | L | Gln | L |
| Ile | B | Tyr | B | Glu | L | Lys | L |
| Phe | B | Pro | N | Asp | L | Arg | L |

**Table 3.** Correspondence of the 20 natural amino acids (columns 20) with the 3 coarse-grained design features (columns 3).

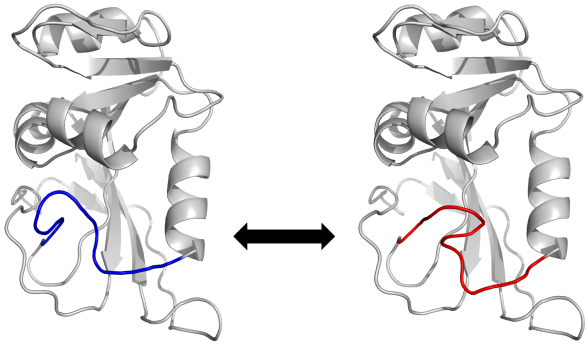

**Figure 10.** Representation of the closed (left) and occluded (left) conformations of the Met20 loop in *ec*DHFR protein.

## 5.2 Experiments

*5.2.1 Test System:* For this preliminary work, we have investigated *Escherichia coli* dihydrofolate reductase (*ec*DHFR). Prior work showed that the loop formed by residues 9–24 (called the Met20 loop) fluctuates between two conformations (called *closed* and *occluded* in related work) at a rate comparable with the speed of the reaction catalyzed by this enzyme (Falzone et al. 1994), indicating that this flexible loop is implicated in this reaction. Other studies also demonstrated that the replacement of central residues of the Met20 loop (Met16, Glu17, Asn18, and Ala19) by glycine residues results in a striking decrease in the enzyme activity (Li et al. 1992; Osborne et al. 2001). Our aim is to study the effect of these mutations on the protein loop motion using SDAP.

*5.2.2 Experiment Setup:* Structures of *ec*DHFR corresponding to the closed and occluded conformational states of the Met20 loop have been solved experimentally and are available in the Protein Data Bank with IDs 1RX2 and 1RX4, respectively. These two structures, which are the starting point for our studies, are illustrated in Figure 10. A comparison of these two structures shows that only the Met20 loop moves significantly, while the rest of the protein backbone is very rigid. Therefore, to reduce the dimension of the configuration space to be explored, all the

protein amino acid residues excepting those of the Met20 loop (residues 9–24) are considered as rigid bodies and are fixed in the space. This simplified model involves 32 DOFs. To keep the two ends of the loop connected to the rest of the protein, kinematic loop closure constraints need to be satisfied during the exploration performed by SDAP. For this, we have applied the approach proposed by Cortés and Siméon (2005) to extend sampling-based planners to closed-chain mechanisms. This approach, based on a decomposition of the degrees of freedom together with the combination of geometrically-constrained random sampling and inverse kinematics, was successfully applied to protein loops in previous work (Cortés et al. 2004).

Concerning the design space, only the four central residues (16–19) are associated to variable design features. The other residues are each constrained to keep the design feature B, L, or N, corresponding to its original amino acid type. This reduces the design space to a size of $3^4 = 81$ possible designs. SDAP was applied to find paths from the occluded conformation to the closed one, considering an initial temperature of 300 K, and a maximum variation of bond rotations of 0.015 radians at each step.

*5.2.3 Results:* For this difficult problem, SDAP was able to find paths connecting the closed and occluded conformations of the Met20 loop. The algorithm required about 45 minutes of CPU time, and generated around 50,000 nodes. However, it failed to identify the best designs for this motion. Indeed, all the nodes in the solution path were accepted by SDAP for all the designs. Moreover, 93% of the nodes generated by the algorithm were labeled as valid for the 81 possible designs. A more accurate analysis of the energy along the solution path revealed that the design-dependent component of the energy is negligible (about 5 orders of magnitude smaller) compared with the total energy variation induced by the loop motion. These results clearly indicate that the chosen coarse-grained energy function is not well adapted to protein motion design. A more detailed energy function, able to differentiate the quality of the sequences for a conformational transition, is required. However, using more accurate (all-atom) energy function is not possible in practice with the current implementation of SDAP, which does not scale to a sequence space of size $20^n$, mainly due to memory issues. As mentioned below, the implementation of an enhanced version of SDAP able to deal with such a complex problem remains a challenging objective for future work.

## 6 Conclusion

In this paper, we have presented an original formulation of a challenging problem combining system design and path planning, and have proposed a new approach to solve it, building on sampling-based algorithms. Our first implementation of SDAP already shows significant gains in efficiency and accuracy compared with a brute-force approach.

While our work was primarily motivated by protein motion design problems, several applications in robotics can also be envisioned. In addition to the design of some robot's features to optimize its motion in a given workspace, it would also be possible to apply the proposed method to optimize the workspace layout for a given robot. One can also imagine applications for helping to the design of modular self-reconfigurable robots. We aim to implement SDAP within robot motion planning software to investigate these potential applications further.

For future work, in addition to investigating a variant of SDAP with asymptotic optimality guarantees based on the any-time T-RRT algorithm (AT-RRT) (Devaurs et al. 2016), we also aim to introduce further improvements. First, the exploration of very-high-dimensional configuration and design spaces implies computer memory issues (the resulting trees or graphs are very large). A solution to this problem would be to introduce pruning stages during the exploration, as is done in the SST* algorithm (Li et al. 2016). Larger design spaces also require SDAP to employ more sophisticated filters and heuristics. A promising direction is to perform statistical learning of the structure of the space, aiming to explore it more efficiently and to control the size of the search tree. To tackle the extreme combinatorial complexity of real-size problems in protein design, the algorithmic improvements described here clearly need to be combined with efficient parallelization strategies of the SDAP algorithm, similar to those proposed for T-RRT (Estaña et al. n.d.) for running on (possibly large) computer clusters.

## Funding

This work was partially supported by the French National Research Agency (ANR) under project ProtiCAD (grant number ANR-12-MONU-0015), and by the European Union's Horizon 2020 research and innovation program under (grant number 644271 AEROARMS).

### References

Al-Bluwi, I., Siméon, T. and Cortés, J. (2012). Motion planning algorithms for molecular simulations: A survey, *Computer Science Review* **6**(4): 125–43.

Baykal, C. and Alterovitz, R. (2017). Asymptotically optimal design of piecewise cylindrical robots using motion planning, *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts.

Baykal, C., Torres, L. G. and Alterovitz, R. (2015). Optimizing design parameters for sets of concentric tube robots

using sampling-based motion planning, *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4381–4387.

Brown, S., Fawzi, N. J. and Head-Gordon, T. (2003). Coarse-grained sequences for protein folding and design, *Proceedings of the National Academy of Sciences* **100**(19): 10712–10717.

Cortés, J. and Siméon, T. (2005). Sampling-based motion planning under kinematic loop-closure constraints, *in* M. Erdmann, D. Hsu, M. Overmars and F. van der Stappen (eds), *Algorithmic Foundations of Robotics VI*, Springer-Verlag, Berlin, pp. 75–90.

Cortés, J., Siméon, T., Remaud-Siméon, M. and Tran, V. (2004). Geometric algorithms for the conformational analysis of long protein loops, *Journal of Computational Chemistry* **25**(7): 956–967.

Devaurs, D., Siméon, T. and Cortés, J. (2014). A multi-tree extension of the transition-based RRT: Application to ordering-and-pathfinding problems in continuous cost spaces, *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2991–2996.

Devaurs, D., Siméon, T. and Cortés, J. (2016). Optimal path planning in complex cost spaces with sampling-based algorithms, *IEEE Transactions on Automation Science and Engineering* **13**(2): 415–424.

Donald, B. R. (2011). *Algorithms in Structural Molecular Biology*, The MIT Press.

Estaña, A., Molloy, K., Vaisset, M., Sibille, N., Siméon, T., Bernadó, P. and Cortés, J. (n.d.). Hybrid parallelization of a multi-tree path search algorithm: Application to highly-flexible biomolecules, *Parallel Computing* . (in press).

Falzone, C. J., Wright, P. E. and Benkovic, S. J. (1994). Dynamics of a flexible loop in dihydrofolate reductase from Escherichia coli and its implication for catalysis, *Biochemistry* **33**(2): 439–442.

Gipson, B., Hsu, D., Kavraki, L. and Latombe, J.-C. (2012). Computational models of protein kinematics and dynamics: Beyond simulation, *The Annual Review of Analytical Chemistry* **5**: 273–91.

Gosselin, C. and Angeles, J. (1991). A global performance index for the kinematic optimization of robotic manipulators, *Journal of Mechanical Design* **113**(3): 220–226.

Hauser, K. (2014). The minimum constraint removal problem with three robotics applications, *International Journal of Robotics Research* **33**(1): 5–17.

Hauser, K. and Latombe, J.-C. (2010). Multi-modal motion planning in non-expansive spaces, *International Journal of Robotics Research* **29**(7): 897–915.

Jaillet, L., Cortés, J. and Siméon, T. (2010). Sampling-based path planning on configuration-space costmaps, *IEEE Transactions on Robotics* **26**(4): 635–646.

Janson, L., Schmerling, E., Clark, A. and Pavone, M. (2015). Fast marching tree, *International Journal of Robotics Research* **34**(7): 883–921.

Kabsch, W. and Sander, C. (1983). Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features, *Biopolymers* **22**(12): 2577–2637.

Karaman, S. and Frazzoli, E. (2011). Sampling-based Algorithms for Optimal Motion Planning, *Int. J. Rob. Res.* **30**(7): 846–894.

Kavraki, L. E., Svestka, P., Latombe, J.-C. and Overmars, M. (1996). Probabilistic roadmaps for path planning in high dimensional configuration spaces, *IEEE Transactions on Robotics and Automation* **12**(4): 566–580.

Keating, A. E. (2013). *Methods in protein design*, Vol. 523 of *Methods in enzymology*, Academic Press/Elsevier, Amsterdam.

LaValle, S. M. (2006). *Planning Algorithms*, Cambridge University Press, New York.

LaValle, S. M. and Kuffner, J. J. (2001). Rapidly-exploring random trees : Progress and prospects, *in* B. Donald, K. Lynch and D. Rus (eds), *Algorithmic and Computational Robotics: New Directions*, A.K. Peters, Boston, pp. 293–308.

Li, L., Wright, P. E., Benkovic, S. J. and Falzone, C. J. (1992). Functional role of a mobile loop of Escherichia coli dihydrofolate reductase in transition-state stabilization, *Biochemistry* **31**(34): 7826–7833.

Li, Y., Littlefield, Z. and Bekris, K. E. (2016). Asymptotically optimal sampling-based kinodynamic planning, *International Journal of Robotics Research* **35**: 528–564.

McCarthy, J. M. and Joskowitz, L. (2001). Kinematic synthesis, *in* J. Cagan and E. Antonson (eds), *Formal Engineering Design Synthesis*, Cambridge Univ. Press.

Merlet, J.-P. (2005). Optimal design of robots, *Robotics: Science and Systems*, pp. 8–11.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1953). Equation of state calculations by fast computing machines, *Journal of Chemical Physics* **21**: 1087–1092.

Moll, M., Schwarz, D. and Kavraki, L. E. (2007). *Roadmap Methods for Protein Folding*, Humana Press.

Osborne, M. J., Schnell, J., Benkovic, S. J., Dyson, H. J. and Wright, P. E. (2001). Backbone Dynamics in Dihydrofolate Reductase Complexes: Role of Loop Flexibility in the Catalytic Mechanism†, *Biochemistry* **40**(33): 9846–9859.

Privett, H. K., Kiss, G., Lee, T. M., Blomberg, R., Chica, R. A., Thomas, L. M., Hilvert, D., Houk, K. N. and Mayo, S. L. (2012). Iterative approach to computational enzyme design, *Proceedings of the National Academy of Sciences* **109**(10): 3790–3795.

Rudnick-Cohen, E. S., Azarm, S. and Herrmann, J. (2015). Multi-objective design and path planning optimization of unmanned aerial vehicles, *Proc. 16th AIAA/ISSMO Multidisciplinary*

*Analysis and Optimization Conference, AIAA Aviation*.

Shehu, A. (2013). Probabilistic search and optimization for protein energy landscapes, *in* S. Aluru and A. Singh (eds), *Handbook of Computational Molecular Biology*, Chapman & Hall/CRC Computer & Information Science Series.

Siméon, T., Laumond, J.-P., Cortés, J. and Sahbani, A. (2004). Manipulation planning with probabilistic roadmaps, *International Journal of Robotics Research* **23(7)**: 729–746.

Tinberg, C. E., Khare, S. D., Dou, J., Doyle, L., Nelson, J. W., Schena, A., Jankowski, W., Kalodimos, C. G., Johnsson, K., Stoddard, B. L. and Baker, D. (2013). Computational design of ligand-binding proteins with high affinity and selectivity, *Nature* **501**: 212–6.

Touw, W. G., Baakman, C., Black, J., te Beek, T. A. H., Krieger, E., Joosten, R. P. and Vriend, G. (2015). A series of PDB-related databanks for everyday needs, *Nucleic Acids Research* **43**(D1): D364–D368.

Vijaykumar, R., Waldron, K. J. and Tsai, M. J. (1986). Geometric optimization of serial chain manipulator structures for working volume and dexterity, *International Journal of Robotics Research* **5**(2): 91–103.