Solving Piecewise Polynomial Constraint Systems with Decomposition and a Subdivision-Based Solver

Boris van Sosin[1,*], Gershon Elber[1]

Abstract

Piecewise polynomial constraint systems are common in numerous problems in computational geometry, such as constraint programming, modeling, and kinematics. We propose a framework that is capable of decomposing, and efficiently solving a wide variety of complex piecewise polynomial constraint systems, that include both zero constraints and inequality constraints, with zero-dimensional or univariate solution spaces. Our framework combines a subdivision-based polynomial solver with a decomposition algorithm in order to handle large and complex systems. We demonstrate the capabilities of our framework on several types of problems and show its performance improvement over a state-of-the-art solver.

Keywords: Underconstrained problems, Inequality constraints, B-spline representation, Multivariate composition

1. Introduction

Geometric and algebraic constraint systems are ubiquitous in all computational geometry and CAD systems, and have probably been used in CAD systems as long as CAD software has existed [1]. The power to design models by specifying the relations (or constraints) between geometric entities, rather than manually setting the location and properties of all the geometric entities plays a crucial part in the usability of the computational geometry system. In many applications, geometric constraint systems can get very complicated. Virtually all computational geometry systems have to support non-linear constraints, as even a constraint as simple as a Euclidean distance between two points is quadratic. Further, constraints on geometric entities such as Bézier, B-spline and NURBs curves (and also surfaces and multivariates) can be high degree polynomial, piecewise-polynomial and even piecewise-rational. In order to handle such problems, most applications require numerical solvers for constraint systems, and for such solvers, solving large constraint systems, with many variables and constraints is a very time-consuming task. Therefore, finding ways of speeding up the solution of constraint systems is highly desirable. One way to speed up the solution is by constraint system decomposition: instead of solving the entire system at once, the solver decomposes the system into smaller sub-systems whenever possible, solves the sub-systems, and finally reconstructs the solution of the original system from the solutions of the sub-systems. Since the time it takes most constraint system solvers to run scales non-linearly, and often even exponentially, with the size of the problem, solving a series of sub-problems in sequence can be very effective at speeding up the solution process.

Constraints can be expressed in algebraic form as equations, and, in the general case, have the form: $f(x) = 0$. Such constraints are called *zero constraints*. Additionally, geometric problems can have *inequality constraints*, which can usually be expressed in the form: $f(x) \geq 0$.

Typically, constraint systems have variables, which determine the number of degrees of freedom (DoF) the system has. Here are a few examples: a point which can move without any restrictions in 2D Euclidean space is defined by two independent variables, x and y, and therefore has two DoFs. Similarly, in 3D Euclidean space a point has three DoFs. A rigid body in 3D Euclidean space has six DoFs. A point that can move only along a parametric curve (such as a Bézier or a B-spline curve) has only one DoF. The total degree of freedom of the system is defined as follows:

Definition 1.1. *Let $\mathcal{G}$ be the set of all entities in the constraint system $\mathcal{S}$. Let $\#c$ be the number of independent zero constraints in the system $\mathcal{S}$. Then, the* Total Degree of Freedom *of $\mathcal{S}$ is: $(\sum_{g\in\mathcal{G}} DoF(g)) - \#c$.*

Inequality (semi-algebraic) constraints do not affect the total degree of freedom of the system, while they restrict the domain of the solution search. Constraint systems can be classified into three categories, by the number of degrees of freedom: well-constrained, underconstrained and over-constrained. Well-constrained systems have a total DoF of zero. In the general case, well-constrained systems have a finite number of solutions. Underconstrained systems have a positive total DoF, and typically have an infinite number of solutions. In many types of underconstrained geometric problems (such as kinematic problems, which will be discussed in Section 4) the solutions indicate a range of motion of one or more parts of the system. Finally, over-constrained systems have a negative total DoF, and, in general, have no solution.

Our goal in this work is to construct a framework for efficiently solving non-linear constraint systems, represented

*Corresponding author. E-mail address: sosin@cs.technion.ac.il (B. van Sosin)

[1]Computer Science Department, Technion - Israel Institute of Technology, Haifa, Israel

as Bézier or B-spline multivariate functions, with zero-dimensional or one-dimensional solution space (i.e. total DoF of zero or one), by using a subdivision-based polynomial solver and a constraint system decomposition algorithm. We will demonstrate the capabilities of our framework on a variety of geometric problems, including high-order spline geometry, and show a significant improvement in performance, in solving these problems. Additionally, we aim to support inequality constraints in the decomposition scheme of our framework. While many of the previously proposed decomposition treat underconstrained problems as user errors, our proposed framework aims to solve both well-constrained and underconstrained problems, as long as their solution space is (a set of) univariate(s), and handle both cases in a similar way that is also extendable to higher dimensional solution spaces.

The rest of this document is organized as following. In Section 2, we discuss previous work on the subject of constraint system decomposition, and the differences between geometric and algebraic approaches. In Section 3, we describe our proposed solution, and detail its steps. In Section 4, we present several problems that are decomposed with our algorithm and their solutions, and compare the performance of the decomposition process to a similar solver without decomposition. Finally, in Section 5, we conclude and discuss directions for future research.

2. Related work

A variety of schemes for the decomposition of constraint systems have been proposed in the past to fit different types of problems and different solution strategies. Many are described in the survey of [2], and we discuss here some of the works most relevant to the problem we are solving. We describe the two most common approaches to decomposition of constraint systems: the geometric approach (Section 2.1) and the algebraic approach (Section 2.2). We will cite several works which use these approaches, and discuss the differences between them. We also consider works which discuss inequality constraints in Section 2.3.

2.1. Geometric approach to decomposition

The algorithms that take the geometric approach work directly on geometric entities and constraints. One example of the group of geometric approach algorithms is sometimes called Recursive Division [2]. Recursive Division is a top-down approach, which attempts to split a geometric constraint system into rigid subsystems. A geometric constraint system is considered rigid if none of its components (points, lines, curves, etc.) can move independently of the others without violating the constraints, and therefore the solutions to such a constraint system, if any exist, form a rigid body. Rigid geometric constraint systems can be either well-constrained or overconstrained. An implementation of Recursive Division called Owen's Method (e.g. in [2]), that works on rigid systems in 2D geometry is described in [3]: it works by representing the constraint system as a graph, in which geometric entities (such as points, lines, circles, etc.) are represented by vertices, and the relation between them (a line going through a point, the distance between points, the angle between lines, etc.) as edges. Owen's method breaks down the graph of geometric entities into smaller subgraphs, which represent rigid sub-components of the original system, and after obtaining solutions for the sub-components, it re-assembles them to form the complete solution for the system. Additionally, the Recursive Division scheme has also been extended to handle underconstrained systems in [4], and for 3D systems in [5].

A different variant of the geometric approach is called Recursive Assembly, and it works in an opposite (but in a way, similar) way to Recursive Division. Recursive assembly, which is described as a bottom-up approach in [2], works by finding rigid subsystems (preferably small ones) in the constraint graph, and recursively merging them. The order of the merge steps in the recursive assembly process yields the order in which the subsystems need to be solved. Once the partition into subsystems is obtained, the solution is constructed using an explicit geometric construction process (such as ruler and compass steps). A Recursive Assembly algorithm is described in [6]. Another approach, that is related to the geometric approach, uses combinatorial algorithm and is proposed in [7]. The combinatorial algorithm is capable of efficiently solving systems with a total DoF of one (i.e. univariate solution space) in which all the constraints are Euclidean distances in 2D space, but is not suited for more general problems.

2.2. Algebraic approach to decomposition

The algebraic approach to constraint system decomposition works on the algebraic representation of the constraint system (i.e. the equations), rather than on geometric entities and relations. The constraint system can be represented as a bipartite graph in which one side of the vertices represents the variables, the other side represents the constraints, and an edge between a variable and a constraint indicates that the variable appears in the constraint. The goal of the algebraic decomposition algorithms is to find a subsystem in the graph that can be solved independently of the rest of the problem, then to remove it from the graph, and continue recursively until the remaining graph can no longer be decomposed. This process yields a decomposition of the problem into smaller sub-problems. A notable work in the algebraic approach category is [8], and the decomposition phase of our framework is based on it. A similar work to [8], used for debugging electrical circuits is described in [9].

A very different, but noteworthy, algorithm in the algebraic category is called Quick Plan (described in [10]) and is aimed towards user interfaces. The interesting feature in Quick Plan is that it allows the user to attach 'weight' values to constraints, indicating how important it is to satisfy them, if the algorithm can't satisfy all the constraints set by the user. This is very different from our goal, which is to satisfy all the constraints or determine that there is no solution to the system.

The main advantage of the algebraic approach is that it is not dependent on the geometry (2D, 3D, or even higher dimensions), and can therefore be applied to a wide variety of general algebraic equation systems, including geometric constraint problems. On the other hand, the geometric approach is a more tailored solution to the geometric problems they are designed to solve. Geometric algorithms can

be more aware of the semantics of the geometric problem, and they use construction rules that can often be made more efficient than the general solvers used by algebraic algorithms (Section 7 in [2]). This applies, for example, to solving kinematic problems and simulating mechanisms, where, in [11], a subdivision based constraint solver (but without a decomposition step), has been used toward that end. Contrast this with works such as [12, 13], which are aimed specifically at theoretical analysis of mechanisms.

2.3. Inequality constraints

Inequality constraints have rarely been addressed in solvers of constraint systems. An interactive system for constructing 3D hierarchical structures from 3D primitives, which is presented in [14], allows the user to specify relations between structures through constraints, including inequality constraints. However, the system does not use an automatic decomposition algorithm, and instead it relies entirely on the user to define the hierarchy of the structures. In [15], inequality constraints are mentioned as a useful tool for allowing the user to select a solution to a problem out of a set of multiple solutions. It is also suggested in [15] to convert inequality constraints into zero constraints by using the identity: $f(x) \geq 0 \Leftrightarrow f(x) - a^2 = 0$. This, however, introduces an extra variable a, which complicates the problem, so finding a way to explicitly handle inequality constraints is highly advantageous.

In [16], a subdivision-based polynomial solver is presented, that is capable of solving systems of multivariate rational spline constraints, including inequality constraints. The solver uses the inequality constraints both as a termination criterion during the subdivision process of the domain, and as a filter on the results. It is capable of solving a wide variety of problems, but does not include a decomposition phase, and solves the entire constraint system simultaneously. Consequently, the time required for this solver to solve a system scales exponentially with the number of zero constraints.

3. Solution process

We start by formally describing the problem we are solving and the subdivision-based polynomial solver we are using to solve it, in Section 3.1. In Section 3.2, we describe the process of symbolic composition of Bézier and B-spline multivariates, which is also required for our framework. Then, the general outline of the algorithm is presented in Section 3.3, the graph decomposition phase is described in Section 3.4, and the solution phase is described in Section 3.5.

3.1. Problem statement

Any polynomial function of degree n can be represented as a Bézier function: $M(t) = \sum_{i=0}^{n} P_i B_i^n(t)$, where $B_i^n(t)$ are the Bézier basis functions. Similarly, B-spline functions can represent piecewise polynomial functions, and NURBs functions can represent piecewise rational functions. Since Bézier functions can be considered a special case of B-spline functions, from now on, we will assume that all the constraints we are using are represented as B-spline functions, unless stated otherwise. All these representations can be extended to multivariates in a variety of ways, most commonly as tensor product multivariates: a polynomial (or piecewise polynomial) function in k variables can be expressed as:

$$M(t_0, t_1, ..., t_{k-1}) = \tag{1}$$
$$= \sum_{i_0=0}^{n_0} \sum_{i_1=0}^{n_1} ... \sum_{i_{k-1}=0}^{n_{k-1}} \left(P_{i_0,i_1,...,i_{k-1}} \prod_{j=0}^{k-1} B_{i_j}^{q_j}(t_j) \right),$$

where $B_{i_j}^{q_j}(t)$ are the B-spline basis functions of order q_j, (or degree $q_j - 1$), and $n_j \geq q_j$. As in the tensor product extension of the B-spline formula to B-spline surfaces (bi-variates) and tri-variates, a B-spline multivariate of k variables requires k knot vectors and has $\prod_{i=0}^{k-1} n_i$ coefficients (or control points). This representation of multivariate piecewise polynomial functions can express a wide range of geometric and algebraic constraint systems:

Definition 3.1. *Let* $\{M_i^{zero}(t)\}_{i=0}^{m-1}$, $\{M_j^{ineq}(\boldsymbol{t})\}_{j=0}^{p-1}$ *be two sets of* m *and* p *scalar B-spline multivariates, respectively, each having* k *variables* $\boldsymbol{t} = (t_0, t_1, ..., t_{k-1})$. *Further, assume that* $t_j \in [t_{j_{min}}, t_{j_{max}}]$ *for all* $j \in [0, k-1]$, *in all the multivariates* $M_i^{zero}(\boldsymbol{t})$, $M_i^{ineq}(\boldsymbol{t})$. *Then:*

$$\mathcal{M}^{zero} = \begin{cases} M_0^{zero}(\boldsymbol{t}) = 0, \\ M_1^{zero}(\boldsymbol{t}) = 0, \\ ... \\ M_{m-1}^{zero}(\boldsymbol{t}) = 0, \end{cases} \quad \mathcal{M}^{ineq} = \begin{cases} M_0^{ineq}(\boldsymbol{t}) \geq 0, \\ M_1^{ineq}(\boldsymbol{t}) \geq 0, \\ ... \\ M_{p-1}^{ineq}(\boldsymbol{t}) \geq 0, \end{cases}$$

$$\mathcal{M} = \mathcal{M}^{zero} \cup \mathcal{M}^{ineq},$$

is a constraint system with $m > 0$ zero constraints, $p \geq 0$ inequality constraints, and k variables.

A few remarks on the constraint systems we are solving:

1. We require that there is at least one zero constraint, but we allow systems with no inequality constraints.
2. Only the number of zero constraints (m) affects the total DoF of the system, and the decomposition of the system into subsystems depends only on $\mathcal{M}^{zero}$.
3. A constraint does not necessarily depend on all variables. If, in M_i, the polynomial order of the variable t_j is one, then M_i is independent of t_j. This applies for both zero and inequality constraints.
4. Since we are finding the zero set of functions, if our constraint system contains rational constraints of the form: $M_i^{zero}(\mathbf{t}) = \frac{M(\mathbf{t})}{w(\mathbf{t})} = 0$, and if $w(\mathbf{t}) \neq 0$ throughout the domain of $\mathbf{t}$ (i.e. the function has no poles), the rational constraint will have the same zero set as $M(\mathbf{t}) = 0$, and the denominator can be ignored. For rational inequality constraints $M_i^{ineq} = \frac{M(\mathbf{t})}{w(\mathbf{t})} \geq 0$, if $w(\mathbf{t}) > 0$ throughout the domain of $\mathbf{t}$, then $w(\mathbf{t})$ can also be ignored. If $w(\mathbf{t}) < 0$ throughout the domain, then the sign of $M(\mathbf{t})$ is flipped.
5. In this paper, when discussing the representation of constraints as multivariates, we will use the notation of M_i. When discussing constraints in general, we will denote them as c_i.

6. We expect well-constrained systems to satisfy the König-Hall condition: for all subsets $\mathcal{M}' \subseteq \mathcal{M}^{zero}$ of the set of zero constraints, the number of neighboring variable vertices needs to be at least $|\mathcal{M}'|$ [8]. For underconstrained systems, we expect that the system can be made well-constrained by fixing the value of one variable.

The main tool we are using for solving polynomial constraint systems is a subdivision-based polynomial solver [16, 17, 18]. It is capable of finding all the real roots of well-constrained constraint systems (in which $m = k$) and underconstrained systems with a univariate solution space (i.e. in which $m = k-1$), up to a prescribed tolerance. The solutions that the solver produces for well-constrained systems are finite sets of points. For problems with a univariate solution space, the solver produces a piecewise linear approximation of the real solution to the system. If there are multiple disjoint solutions to a problem, the solver is capable of finding all of them, up to a prescribed tolerance. One of the properties of most subdivision-based solvers, is that they have an exponential dependency on the dimension of the problem (k in Definition 3.1 and Equation (1), recalling $\prod_{i=0}^{k-1} n_i$), hence solving large problems is a major challenge.

As stated earlier, our goal, in this work, is to construct a framework for decomposing and solving polynomial constraint systems with zero dimensional or univariate solution space using a subdivision-based solver. Also, since the solver we are using is capable of solving both well-constrained problems and problems with a univariate solution space, we can propose a unified framework for decomposing and solving both types of problems.

3.2. Symbolic Composition of B-spline multivariates

Before we get into the main algorithm, we present an important tool that will be used by our algorithm: multivariate functional composition for Bézier/B-spline functions. Let $M(\mathbf{t})$, $\mathbf{t} = (t_0, t_1, ..., t_{k-1})$ be a Bézier multivariate, and let $\boldsymbol{\tau}(v) = (\tau_0(v), \tau_1(v), ..., \tau_{l-1}(v))$ be an l-dimensional vector function of a parameter v, represented as a B-spline univariate. Additionally, assume that $l \leq k$. Then, the composition $M(\boldsymbol{\tau}(v), t_l, ..., t_{k-1})$, where the variables $\tau_0(v), ..., \tau_{l-1}(v)$ are mapped to $t_0, ..., t_{l-1}$ (without loss of generality, as the indices of the variables of $M(\mathbf{t})$ can be re-ordered) can be expressed as:

$$\begin{aligned}
&= M((\boldsymbol{\tau}(v), t_l, ..., t_{k-1}) \\
&= M(\tau_0(v), \tau_1(v), ..., \tau_{l-1}(v), t_l, ..., t_{k-1}) \\
&= \sum_{i_0=0}^{n_0} \sum_{i_1=0}^{n_1} ... \sum_{i_{k-1}=0}^{n_{k-1}} (P_{i_0,i_1,...,i_{k-1}} \prod_{j=0}^{k-1} B_{i_j}^{n_j}(r_j)) \\
&= M_{comp}(v, t_l, ..., t_{k-1}) \qquad (2)
\end{aligned}$$

where:

$$B_{i_j}^{n_j}(r_j) = \begin{cases} \binom{n_j}{i_j}(1-\tau_j(v))^{i_j}(\tau_j(v))^{n_j-i_j}, & 0 \leq j \leq l-1, \\ \binom{n_j}{i_j}(1-t_j)^{i_j} {t_j}^{n_j-i_j}, & l \leq j \leq k-1. \end{cases} \qquad (3)$$

In the first case of Equation (3), $\tau_j(v)$, assumed to be a scalar B-spline function, is mapped to the variable t_j, so that t_j becomes a function of v: $t_j = \tau_j(v)$, and can be expressed as a B-spline function through addition, subtractions, and multiplication of B-spline functions. Examples of works which discuss symbolic arithmetic computations on B-spline functions are [19] and [20]. In the second case of Equation (3), $B_{i_j}^{n_j}(t_j)$ is a Bézier basis function, and is trivially expressed as a B-spline function. The rest of the computations of Equation (2) can also be performed through symbolic addition and multiplication of B-spline multivariates. If $M(\mathbf{t})$ is a B-spline multivariate, rather than Bézier, $M(\mathbf{t})$ is first subdivided at all its internal knot values, to form Bézier patches. Further, $\boldsymbol{\tau}(v)$ is also to be subdivided at all the values of v at which $\boldsymbol{\tau}(v)$ intersects a knot value of $M(\mathbf{t})$, only to be merged back, after the composition is computed. Subdividing $\boldsymbol{\tau}(v)$ at the parameter values at which it intersects knot values of $M(\mathbf{t})$ is simple only if $\boldsymbol{\tau}(v)$ is a univariate function, which is the case herein.

3.3. General outline of the decomposition-based algorithm

An outline of the process of solving a constraint system with decomposition is described in Algorithm 1. The algorithm first analyzes the problem, and a solution plan for the problem is produced:

Definition 3.2. *Let $\mathcal{M}$ be a constraint system with m zero constraints and k variables. A* solution plan graph *for $\mathcal{M}$ is a graph $H_{Plan} = (V_{Plan}, E_{Plan})$ with the following properties:*

1. *H_{Plan} is a directed acyclic graph (DAG).*
2. *Each vertex $v_i \in V_{Plan}$ has a set of constraints attached to it (as constraints$(v_i) = \{M_j\}_{v_i}$). Each vertex $v_i \in V_{Plan}$ represents a step in the solution plan, and constraints(v_i) is the set of constraints of the subproblem solved in this step.*
3. *For $v_i \neq v_j$, constraints$(v_i) \cap$ constraints$(v_j) = \emptyset$, and $\bigcup_{v \in V_{Plan}}$ constraints$(v) = \mathcal{M}$.*
4. *If the subproblem v_j is dependent on the results computed in subproblem v_i, then there is a directed edge $v_i \rightarrow v_j \in E_{Plan}$.*

When discussing inequality constraints, we denote a solution plan graph without inequality constraints as $\bar{H}_{Plan}$, and the solution plan graph augmented with inequality constraints as H_{Plan}. In problems without inequality constraints, $\bar{H}_{Plan}$ and H_{Plan} are identical.

Definition 3.3. *Let $G = (V, E)$ be a DAG. A* topological order *of the vertices V is an order such that if there in a directed edge $v \rightarrow u \in E$, then v comes before u in the ordering [21].*
The act of finding a topological order in a graph is called topological sorting*, but these two terms are sometimes used interchangeably.*

The process of decomposing the problem into a solution plan graph is described in Section 3.4. In the second step, the solution plan, H_{Plan}, which is a DAG, is solved in a topological order. After each step (subproblem) of the plan is solved, its results are propagated to the following subproblems, until the entire plan is solved. The solution process is described in Section 3.5.

Algorithm 1 General solution algorithm

Input:
$\mathcal{M}$: a system of (piecewise) polynomial constraints, as in Definition 3.1, and $m = k$ or $m = k - 1$;

Output:
A set of points (zero dimensional solutions) or piecewise-linear functions (univariate solutions), which solve the constraint system;

Algorithm:

1: H_{Plan}:=*GraphDecomposition*($\mathcal{M}$); // See Algorithm 2.
2: Sol:=$\emptyset$;
3: **while** Subproblems remaining in H_{Plan} **do**
4: Extract subproblem from H_{Plan};
5: Solve subproblem;
6: Add solution of subproblem to Sol;
7: Apply Sol to remaining subproblems; // See Algorithm 6.
8: **end while**
9: **return** Sol;

Algorithm 2 Graph decomposition algorithm

Input:
$\mathcal{M}$: A system of (piecewise) polynomial constraints, as in Definition 3.1, and $m = k$ or $m = k - 1$;

Output:
H_{Plan}: A solution plan graph;

Algorithm:

1: $G = (V, E)$:=Construct a variable-constraint graph for $\mathcal{M}^{zero}$, according to Definition 3.4;
2: $M\!M$:=Find a maximum matching in G; // ($M\!M \subset E$).
3: G':=A dependency graph, constructed from G according to Algorithm 3;
4: H:=Condensed dependency graph of G', according to Algorithm 4;
5: $\bar{H}_{Plan}$:=SCC graph of H; // Each strongly connected component (SCC) represents a subproblem, and the edges between SCCs represent dependencies between the subproblems.
6: H_{Plan} :=Add inequality constraints to $\bar{H}_{Plan}$; // See Algorithm 5.
7: **return** H_{Plan};

3.4. Graph decomposition phase

The decomposition phase of our proposed scheme follows the algorithm described in [8], and the key differences in our algorithm are explained in detail. We now present the decomposition algorithm and demonstrate how it is applied to a simple well-constrained problem shown in Figure 1. In Figure 1, and throughout the rest of the paper, blue dots indicate the points for which the problems are solved (i.e. the unknowns of the problems), and black dots represent fixed points. The decomposition phase, outlined in Algorithm 2, is divided into several steps. First, a variable-constraint graph G that represents the zero constraints of problem is constructed (also demonstrated in Figure 1):

Definition 3.4. *Let $\mathcal{M}^{zero}$ be a the set of zero constraints of a constraint system $\mathcal{M}$ (as described in Definition 3.1) with m constraints and k variables. The* variable-constraint graph *for $\mathcal{M}^{zero}$ is a bipartite graph $G = (V_v \cup V_M, E)$ with the following properties:*

1. *$V_v = \{v_{t_i}\}_{i=0}^{k-1}$, a vertex for each variable t_i in the vector $\boldsymbol{t}$ in $\mathcal{M}^{zero}$.*
2. *$V_M = \{v_{M_j}\}_{j=0}^{m-1}$, a vertex for each constraint in $\mathcal{M}^{zero}$.*
3. *There is an edge $(v_{t_i}, v_{M_j}) \in E$ if and only if constraint M_j is dependent on variable t_i.*

After the variable-constraint graph G is constructed, it is decomposed into subsystems.

Decomposition is done by finding a maximum matching, $M\!M$, in G (see [21]), and then building a new directed graph G' by copying the vertices of G, and converting each of the matched (undirected) edges into a pair of anti-parallel directed edges. All the unmatched edges are also copied to G' as directed edges from the variable vertices to the constraint vertices. The process of building G' is formally described in Algorithm 3, and demonstrated on our simple example in Figure 2.

Algorithm 3 Dependency Graph Construction

Input:
$G = (V, E)$: A variable-constraint graph;
$M\!M \subseteq E$: a maximum matching in G;

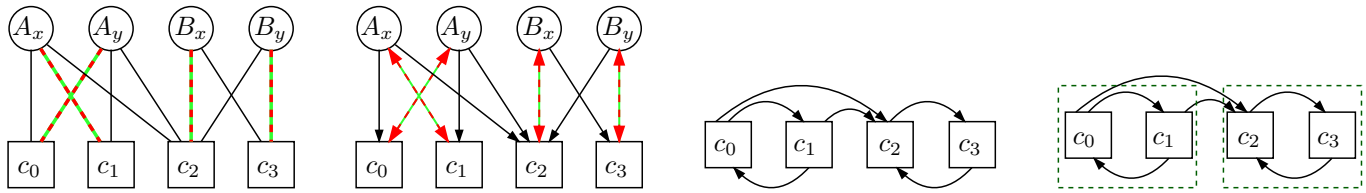
Output:
$G' = (V', E')$: A dependency graph;

Algorithm:

1: V':=V; // G' has the same set of vertices as G.
2: For all matched edges $(v_t, v_M) \in M\!M$, add edges $v_M \rightarrow v_t$, and $v_t \rightarrow v_M$ to E';
3: For all unmatched edges $(v_t, v_M) \in E - M\!M$, add an edge directed from the variable to the constraint, $v_t \rightarrow v_M$, to E';
4: **return** $G' = (V', E')$;

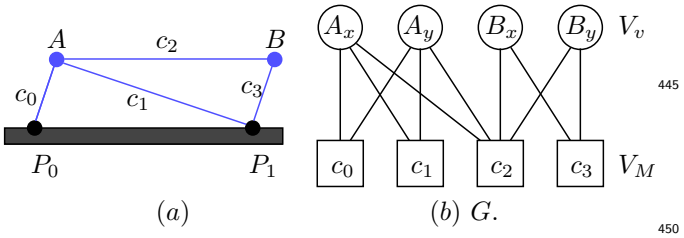
Definition 3.5. *Let $G' = (V', E')$ be a directed graph. The* strongly connected components *[21] (*SCCs*) of G' are a partition of V' into disjoint subsets $\{S_i\}$, such that $\bigcup S_i = V'$, and for $i \neq j$, $S_i \cap S_j = \emptyset$, with the following property: for any two vertices $v_i, v_j \in V'$, v_i and v_j are in the same SCC if and only if there are directed paths from v_i to v_j and from v_j to v_i, in G'.*

The directed graph G' represents the flow of information in the proposed solution sequence of the problem, and the decomposition will be decided based on G'. For example, a directed edge from a variable vertex to a constraint vertex



(a) $\mathcal{MM}$ in G (b) G' (c) H (d) SCCs of H

Figure 2: The decomposition process of the graph G (Figure 1). (a) A maximum matching, $\mathcal{MM}$, in G, in red-green dashed lines. (b) The directed dependency graph G' that results from $\mathcal{MM}$. (c) The condensed dependency graph H of G'. (d) The SCCs of H, showing the two subproblems and the flow of information from the first subproblem to the second. Note that even though the maximum matching in this graph is non-unique, all maximum matchings result in the same graph decomposition. Proof of this can be found in [8].



(a) (b) G.

Figure 1: (a) A simple, well-constrained 2D point-and-bar problem. The black dots indicate fixed points (therefore they are not assigned variables). The blue dots indicate the unknowns of the problem, and each is assign variables for its x and y coordinates (hence, A_x, A_y, B_x, B_y). The bars c_0, c_1, c_2, c_3 are assumed to be rigid, and are therefore represented as distance constraints. (b) The variable-constraint graph, G, for the system in (a), as described in Definition 3.4.

($v_t \rightarrow v_M$) indicates that constraint M depends on the variable t. A directed edge $v_M \rightarrow v_t$ indicates that variable t depends on constraint M. Furthermore, a cycle, or a SCC in G' indicates a set of variables and constraints that all depend on each other, and therefore must be solved as a subproblem.

Since in any maximum matching $\mathcal{MM}$, a constraint vertex v_M can be matched to at most one variable vertex v_t, in G', v_M can have at most one outgoing edge $v_M \rightarrow v_t$. Additionally, due to the properties of G', there must also be an edge in reverse direction: $v_t \rightarrow v_M$. Therefore, if v_M is matched with v_t in $\mathcal{MM}$, they must both be in the same SCC. Unmatched vertices must be in an SCC of their own, because they either have no incoming edges (if they are variables), or no outgoing edges (if they are constraints). Therefore, any SCC in G' that has at least one variable vertex and one constraint vertex contains an equal number of constraint vertices and variable vertices.

Consider an SCC, $S \subseteq V'$ in G'. If we have a solution for all the variable vertices outside S that have edges leading into S, then we can plug the values of these variables into the constraints of S, and solve the subsystem S. This process describes solving a single subsystem in the decomposed constraint system $\mathcal{M}$. If $\mathcal{M}$ is well-constrained, then $\mathcal{MM}$ is a perfect matching (see proof in [8]), and there will be no unmatched variable vertices. Therefore, there will be at least one SCC with no incoming edges. SCCs with no incoming edges represent subsystems that can be solved first, and their solutions will be propagated to the subsystems that follow.

For underconstrained problems, there will be at least one unmatched variable vertex (such variables are called input variables in [10]). An unmatched variable vertex will be in an SCC of its own, as it can't have any incoming directed edge in G'. In [8], input variables are treated as parameters that can be set externally to make the system well-constrained. However, our goal is to build a unified framework for solving problems with zero-dimensional and univariate solution spaces, while ensuring the topology. Since our polynomial solver is capable of producing solutions for problems with univariate solution spaces, we merge the input variables into the first subsystem that uses them by condensing the graph G' into a graph that contains only the constraint vertices. Condensing the dependency graph was proposed in [22], but with a different goal in mind.

A condensed graph H is constructed by taking the set of all constraint vertices of G' and connecting a pair of vertices v_{M_i}, v_{M_j} by a directed edge $v_{M_i} \rightarrow v_{M_j}$ if and only if in G' there is a directed path from v_{M_i} to v_{M_j} going through a single variable vertex. The construction of H is formally described in Algorithm 4.

Algorithm 4 Condensing The Dependency Graph

Input:
$G' = (V', E')$: A dependency graph;

Output:
$H = (V_H, E_H)$: A condensed dependency graph;

Algorithm:

1: V_H:=$\{v_M | v_M \in V'\}$; // The vertices of H are the constraint vertices of G'.
2: For all pairs of vertices $v_{M_i} \neq v_{M_j}$ in G', if there is a variable vertex v_t such that $v_{M_i} \rightarrow v_t \in E'$ and $v_t \rightarrow v_{M_j} \in E'$, add an edge $v_{M_i} \rightarrow v_{M_j}$ to E_H;
3: **return** $H = (V_H, E_H)$

The graph H allows us to find SCCs which contain only constraint vertices, and correspond to SCCs in G' that contain at least one constraint vertex. This allows us to eliminate SCCs that contain a single variable vertex, and consider only constraints in the solution phase. For example, see the graphs G', H, H_{Plan} in Figure 2: in G', there is a directed path $c_0 \rightarrow A_y \rightarrow c_1$, indicating that c_1 depends directly on c_0. Therefore, we see in H an edge $c_0 \rightarrow c_1$. Similarly, there is a path $c_1 \rightarrow A_x \rightarrow c_0$ in G', and therefore there is an edge $c_1 \rightarrow c_0$ in H, and so on for the rest of G'. We use the SCC graph of H, denote $\bar{H}_{Plan}$, as the

DAG for the solution plan, and employ any topological order on $\bar{H}_{Plan}$ as the order in which the subsystems will be solved. The zero constraints in each subsystem are the constraints corresponding to the vertices in the SCCs of H.

Algorithm 5 Adding inequality constraints to the solution plan

Input:
$\mathcal{M}$: A system of (piecewise) polynomial constraints, as in Definition 3.1, and $m = k$ or $m = k - 1$;
$\bar{H}_{Plan} = (\bar{V}_{Plan}, \bar{E}_{Plan})$: The solution plan graph for $\mathcal{M}$ without inequality constraints;

Output:
$H_{Plan} = (V_{Plan}, E_{Plan})$: A solution plan graph augmented with inequality constraints; // See Definition 3.2.

Algorithm:
1: (V_{Plan}, E_{Plan}):=$(\bar{V}_{Plan}, \bar{E}_{Plan})$;
2: **for all** $v \in V_{Plan}$ **do**
3: **for all** $M_i^{ineq} \in \mathcal{M}^{ineq}$ **do**
4: **if** at least one of the variables on which M_i^{ineq} depends is being solved for in subsystem v, and all the other variables which are not being solved for in the subsystem v already have solutions **then**
5: $constraints(v)$:=$constraints(v) \cup \{M_i^{ineq}\}$;
6: **end if**
7: **end for**
8: **end for**
9: **return** $H_{Plan} = (V_{Plan}, E_{Plan})$;

Once $\bar{H}_{Plan}$ is computed, the inequality constraints are added to the subsystems in which they can be solved. Each inequality constraint M_i^{ineq} is added to the subsystem in which at least one of the variable on which M_i^{ineq} depends is being solved for, in the subsystem, and all the variables which are not being solved for in the current subsystem, already have solutions (see Algorithm 5). We assume that the vertices of $\bar{H}_{Plan}$ contain the information of which constraints are solved, and the solutions for which variables are computed in each subsystem. Algorithm 5 only adds inequality constraints to the existing subsystems in $\bar{H}_{Plan}$, and does not change the set of vertices, or the connectivity of $\bar{H}_{Plan}$.

The decomposition for well constrained problems is unique, even for different perfect matchings in G (again, see proof in [8], and in more detail in [23]), but for under-constrained problems, different maximum matchings can result in different decompositions. For example, in Figure 3, three different decompositions of the same under-constrained problem are shown, and the sizes of the subproblems in the different decompositions are different. The time it takes to solve a subsystem in the subdivision-based solver [17, 18] scales exponentially with the number of constraints in the subsystem. Hence, we aim to minimize the size of the largest subsystem in the decomposition. For the purpose of solving systems with a univariate solution space, we propose an exhaustive search on the maximum matchings by iterating over the variable vertices, and for each vertex finding a maximum matching that does not include it. This is done in $O(k) \cdot O(M\!M)$, where k is the number of variables and $O(M\!M)$ is the complexity of the maximum matching algorithm, for example, the Hopcroft-Karp algorithm has a time complexity of $O(E\sqrt{V})$ [21]. The exhaustive search adds negligible running time compared to the numerical solution time of the subsystems.

3.5. The solution phase

With the solution plan graph, H_{Plan}, the subsystems in H_{Plan} are solved in a topological order (see Algorithm 6). Solving the subsystems in a topological order assuring that when a subsystem needs to be solved, there are already values assigned to all the variables which are required to solve it. When the first subsystem is solved, we assign values to the variables computed in the subsystem. Zero dimensional solutions (as a finite set of points) as well as univariate solutions (as a finite set of piecewise-linear solutions) can be assigned as a solution to a variable. Univariate solutions need to be parameterized in order to have the same representation as the constraints for further processing. The parametrization of univariate solutions is done either by parameterizing the piecewise-linear solutions directly, or by fitting a parametric curve (represented as a B-spline curve, in our case) which approximates the piecewise-linear solution (using least squares approximation). One should note that the exact parametrization of the solution is not important. For example, the zero set of a scalar surface is, in the general case, a (set of) univariate curves, but different regular parametrizations of the solution curves are all valid solutions to the problem. When

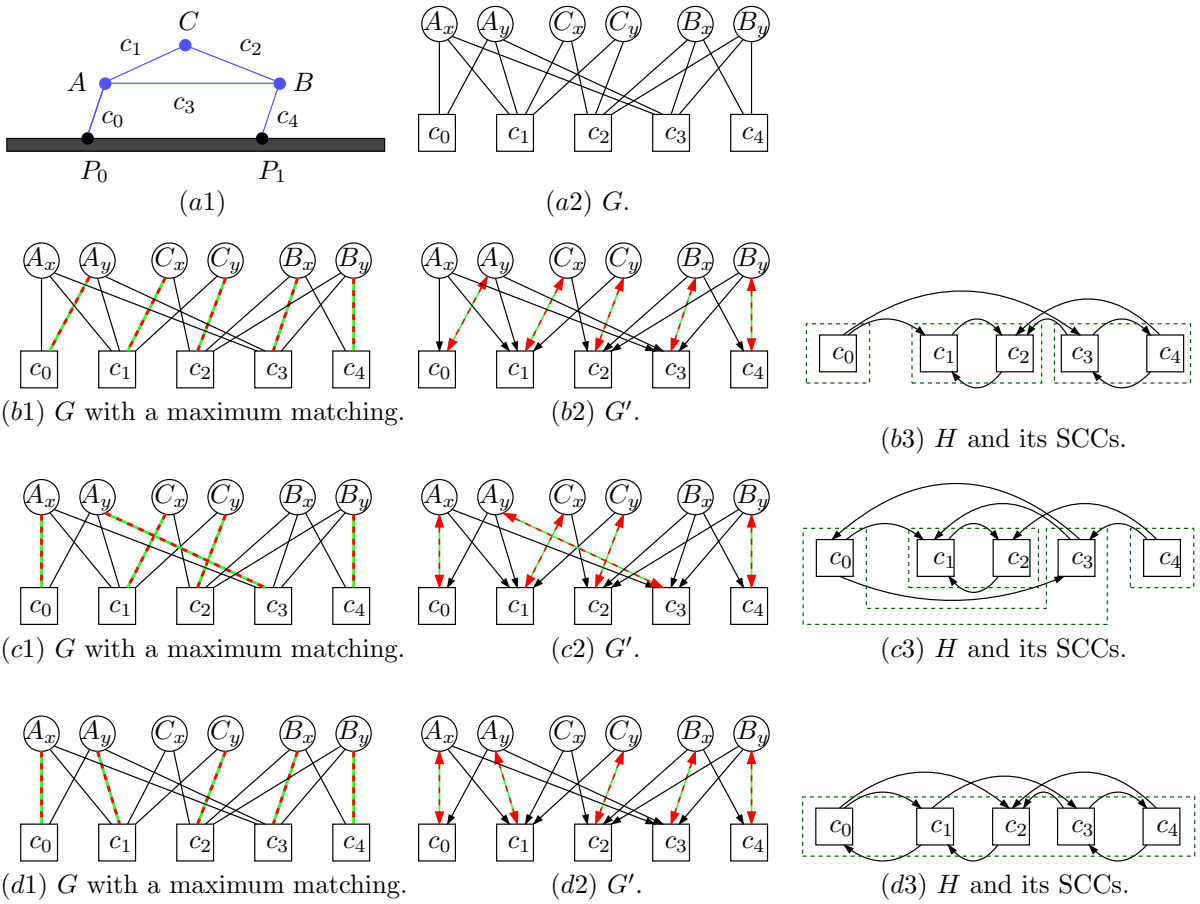
Algorithm 6 Plan solution algorithm

Input:
H_{Plan}: A solution plan;

Output:
A set of point (zero-dimensional) solutions or piecewise-linear univariate solutions, which solve the constraint system;

Algorithm:
1: Sol:=$\emptyset$;
2: $\{M_i\}_j$:= get next subsystem in the topological order of H_{Plan}, or stop if there are no more subsystems;
3: **for all** $M_i \in \{M_i\}_j$ **do**
4: $\hat{M}_i$:=Compose solved variables into M_i;
5: **end for**
6: $CurrSol$:=$Solve(\{\hat{M}_i\})$;
7: **for all** $\mathbf{t}_{solved}(u) \in CurrSol$ **do**
8: Sol:=$MergeAndReparameterize(Sol, \mathbf{t}_{solved}(u))$; // Reparameterize previously solved variables only if needed
9: Recursively call steps 2-9
10: **end for**

subsequent subsystems are solved, several steps need to be performed:



$(a1)$ $(a2)$ G.

$(b1)$ G with a maximum matching. $(b2)$ G'. $(b3)$ H and its SCCs.

$(c1)$ G with a maximum matching. $(c2)$ G'. $(c3)$ H and its SCCs.

$(d1)$ G with a maximum matching. $(d2)$ G'. $(d3)$ H and its SCCs.

Figure 3: A demonstration of the decomposition algorithm on an underconstrained problem (a wedge attached to one of the bars in a four-bar linkage). Row (a): $(a1)$ The 2D point-and-bar diagram of the system. $(a2)$ The variable-constraint graph G of the system. All c_i constraints are distance constraints between points. Rows $(b), (c), (d)$: $(b1), (c1), (d1)$ Three different maximum matchings in G. $(b2), (c2), (d2)$ The dependency graphs G' that result from the maximum matching $(b1), (c1), (d1)$ respectively. $(b3), (c3), (d3)$ The condensed dependency graph, H, of the graphs in $(b2), (c2), (d2)$, respectively, and the SCCs of H, marked by dashed green blocks. Note that different maximum matchings of the variable-constraint graph, $(b1), (c1), (d1)$ result in different decompositions of the problem which don't necessarily have the same number of SCCs. The decomposition in $(d3)$ has a single SCC containing the entire system, which is highly undesirable.

1. Before a subsystem can be solved, all the variables in the system for which there already are solutions, must be applied to the constraints in the subsystem. Recall that the constraints are represented as B-spline multivariates. Before the subsystem is solved, the B-spline multivariates are still dependent on variables for which we have solutions. For zero dimensional solutions, the constraint multivariates are reduced to isoparametric sub-multivariates by fixing the value of the variables for which we have a solution to the value of the solution, while preserving the topology of the solution. For univariate solutions, the preservation of the topology is achieved by the symbolic composition of the univariate solution B-spline into the constraint multivariates (as described in Section 3.2). This allows univariate solutions for one or more variables to be propagated into the constraints that need them, in order to be solved. If the variable vector of the problem is $\mathbf{t} = (t_0, t_1, ..., t_{k-1})$, and we have a univariate solution for $t_0, ..., t_{l-1}$, parameterized as $\boldsymbol{\tau}(v) = (t_0(v), ..., t_{l-1}(v))$, the constraint M_i undergoes the composition: $M_i(\boldsymbol{\tau}(v), t_l, ..., t_{k-1}) = M_{i_{comp}}(v, t_l, ..., t_{k-1})$ and the new constraints become $M_{i_{comp}}(v, t_l, ..., t_{k-1}) = 0$, for all constraints M_i in the current subsystem.
2. Once a subsystem is solved, if the new result is a univariate (this can happen regardless of the input to the current subsystem), it is again parameterized. The solver produces the solution to $\{M_{i_{comp}}(v, t_l, ..., t_{k-1})\}$, as $(v(u), t_l(u), ..., t_{k-1}(u))$. Therefore, the new parametrization, $v(u)$, needs to be applied to the previously solved variables, $\boldsymbol{\tau}(v)$. This is done, again, through function composition by computing: $\boldsymbol{\tau}(v(u))$.
3. Once all the solved variables are represented as a function of u, they can be combined into a single vector function with a single parameter, u.

The steps of composing the previously solved variables into the constraints of the current subsystem, solving the current subsystem, and reparameterizing the previously solved variables are repeated for all the subsystems, in a topological order, until the entire system is solved.

Since non-linear systems often have more than one solution, whenever a subsystem is solved *all* its solutions need

to be considered. Therefore, for every solution of a subsystem, the vector of previously solved variables is duplicated, and the current subsystem's solution is merged into its own copy of the solution vector (including the reparametrization mentioned above). The solution process therefore has a recursive branching form. At the end of the recursion, when the last subsystem is solved, the set of full solutions from all the computation branches need to be collected to form the complete solution of the entire system.

The inequality constraints also have an effect on the recursive branching of the solution process. Our framework is designed to add the inequality constraints to the first subsystem in which all the variables required to check the inequalities are either being solved, or already have solutions. Therefore, as soon as the algorithm has enough information to determine that a partial solution completely fails one or more of the inequality constraints, the partial solution, along with all the recursive computation that follows from it, is pruned from the recursive branching process.

4. Results and examples

We start by demonstrating the advantages of inequality constraint on two simple examples, one well-constrained and one underconstrained. First, let us revisit the problem in Figure 1, a parallelogram with a diagonal, but this time we will assign values to the constants. The algebraic representation of the problem is:

$$\begin{cases} |A-P_0|^2 - c_0^2 = 0, & c_0 = 0.9, \\ |A-P_1|^2 - c_1^2 = 0, & c_1 = 2.5, \\ |A-B|^2 - c_2^2 = 0, & c_2 = 2, \\ |B-P_1|^2 - c_3^2 = 0, & c_3 = 0.9, \end{cases}$$
$$for\ P_0 = (-1,0),\ P_1 = (1,0).$$

As demonstrated in Figure 2, the algorithm first finds A by solving the subsystem $\{c_0, c_1\}$, resulting in the solutions:

$$\{(A_x = -1.36, A_y = 0.825), (A_x = -1.36, A_y = -0.825)\}.$$

Then, each of these solutions is used to set the values of A_x and A_y in constraint c_2, and together with c_3 it is used to find the values of B. There are two such solutions for each. The resulting solutions are (see Figure 4):

$$(A_x, A_y, B_x, B_y) = \{(-1.36, 0.825, 0.205, -0.421),$$
$$(-1.36, 0.825, 0.64, 0.825), (-1.36, -0.825, 0.205, 0.421),$$
$$(-1.36, -0.825, 0.64, -0.825)\}.$$

In one of the solutions (Figure 4 (b)), the entire structure is under the x-axis line, and in two of the other solutions (Figures 4 (a), and 4 (c)), the AB line intersects the x-axis line, forming two partially overlapping triangles with a common base instead of a parallelogram. Out of all these solutions, the user has to choose the solution (or solutions) which correctly satisfies for user's requirements.

Alternatively, we can add inequality constraints to efficiently filter out the solutions we a-priori decide are incorrect. We can add two constraints: $A_y \geq 0$, which makes sure that A is above the x-axis line, and $z((B-P_1)\times(A-P_1)) \geq 0$, where $z(\mathbf{q})$ denotes the z element of the vector expression $\mathbf{q}$, which makes sure the triangle ABP_1 is constructed above the AB line. In the decomposition process, the constraint $A_y \geq 0$ is added to the first subsystem (because it only depends on A_y), and therefore the only solution to the subsystem is $A_x = -1.36, A_y = 0.825$. The inequality constraint $z((B-P_1)\times(A-P_1)) \geq 0$ depends on all the variables, and must therefore be added to the second subsystem. These two inequality constraints make sure that only the solution in Figure 4 (d) is returned.

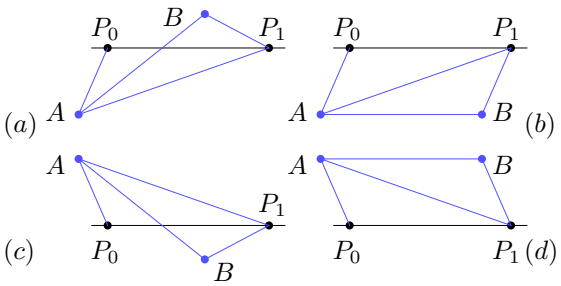


Figure 4: All the solutions to the parallelogram with a diagonal problem. See also Figure 1.

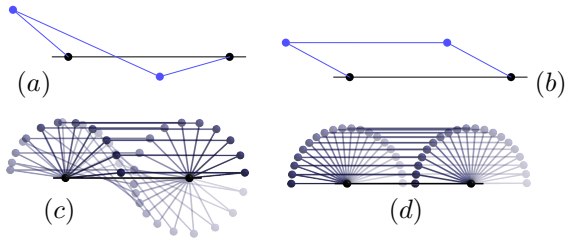


Figure 5: (a) A single frame from the the motion range of the 4-bar linkage without inequality constraints, and (b) with inequality constraints. (c) Samples from the motion range without inequality constraints, and (d) with inequality constraints .

By removing the diagonal constraint c_1, from Figure 1, we get the 4-bar linkage underconstrained problem. As expected, there are two possible solution plans: solving for A first, and solving for B first, out of which our framework chooses, arbitrarily, since they are symmetrical, to solve for B first. However, our framework returns two solutions, out of which in one of them, the AB line intersects the x-axis (see Figure 5 $(a), (c)$), like in the previous problem. If we add the same two inequality constraints as before: $A_y \geq 0$, $\quad z((B-P_1)\times(A-P_1)) \geq 0$, we observe two changes: first, we get only the part of the motion range in which A is above the x-axis, and second, the AB line does not intersect the x-axis (see Figure 5 $(b), (d)$). This shows that in problems with univariate solutions, inequality constraints can not only filter entire solutions, as in zero-dimensional solutions, but also limit the range of univariate solutions.

Next, we look at a more complex and realistic kinematic problem: the Jansen's linkage [24, 25]. Jansen's linkage (see Figure 6) is a mechanism for the legs of walker robots designed by Theo Jansen which transforms circular motion at the axis of the leg into nearly linear motion at the 'foot'. We used our framework to define a Jansen's linkage leg through its constraints and produce a simulation of its walking motion. Our framework decomposed

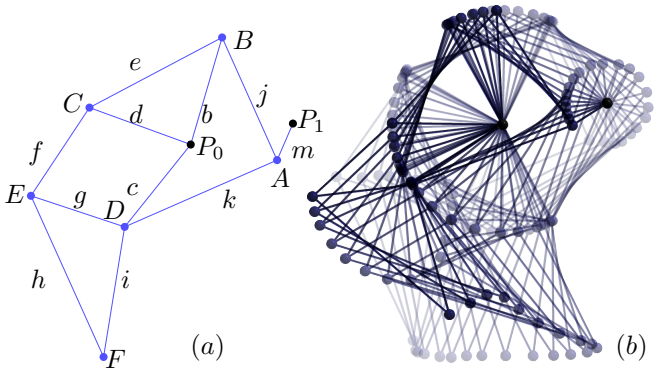


Figure 6: (a) The definition of Jansen's linkage by the distance constraints of its bars [25]. (b) The motion range of Jansen's linkage.

and problem into 6 subsystems, and found 38 solutions to the constraint system, most of them having intersections between the various bars of the mechanism. In order to get the correct solution, we added the following inequality constraints:

$$\begin{cases} (P_0 - B) \times (A - B) \geq 0, \\ (A - D) \times (P_0 - D) \geq 0, \\ (C - B) \times (P_0 - B) \geq 0, \\ (E - D) \times (F - D) \geq 0, \\ (D - E) \times (C - E) \geq 0. \end{cases}$$

The univariate solution can be seen in Figure 6 (b).

The next problem we present is another kinematic problem with Euclidean distance constraints, however, this time the distance constraints are between B-spline curves and surfaces in 3D. As a consequence of this, all the constraints are high-degree piecewise polynomial functions. The problem is presented in Figure 7. The problem has a fixed point P_0, two curves $\mathbf{c}_1, \mathbf{c}_2$, and a surface S. The variables of the problem are a point $A = \mathbf{c}_1(t_A)$, a point $B = (B_x, B_y)$, that is free to move on the XY plane, two points $C = \mathbf{c}_2(t_C), D = \mathbf{c}_2(t_D)$, and a point $E = S(u_E, v_E)$. Its six constraints in seven unknowns $(t_A, t_C, t_D, B_x, B_y, u_E, v_E$. L_i are constants) are:

$$\begin{cases} |B - P_0|^2 - L_1^2 = 0, \\ |\mathbf{c}_1(t_A) - B|^2 - L_2^2 = 0, \\ |\mathbf{c}_2(t_C) - B|^2 - L_3^2 = 0, \\ |\mathbf{c}_2(t_D) - B|^2 - L_4^2 = 0, \\ |\mathbf{c}_2(t_C) - S(u_E, v_E)|^2 - L_5^2 = 0, \\ |\mathbf{c}_2(t_D) - S(u_E, v_E)|^2 - L_6^2 = 0. \end{cases}$$

Due to the irregularity of the freeforms in the problem, our framework finds multiple (four, in this case) disjoint solutions.

A slightly different type of problem we have solved with our framework is the inverse kinematic problem, again, over freeform shapes. Instead of defining a mechanism only by its constraints and using our framework to find its motion paths, we define a path along which one of the parts of the mechanism must move, and solve for the rest of the mechanism. Specifically, we embedded the letter "C" in the body of the Utah teapot, and computed the motion

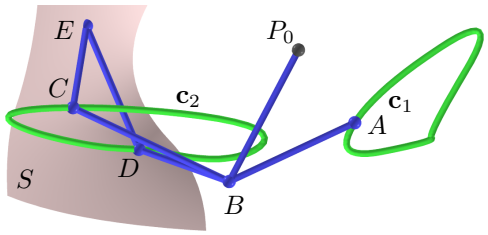


Figure 7: The kinematics over splines problem. The positions of the points are sampled from one of the solutions to the problem.

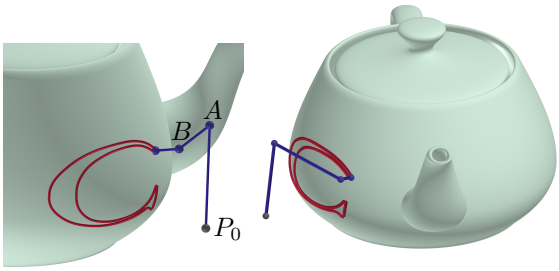


Figure 8: A frame from the motion range of the teapot engraving inverse kinematic problem, from two viewing angles.

path of a 5-DoF robotic arm that is needed to engrave the letter "C" on the teapot (see Figure 8). We represented the surface of the body of the teapot as $\mathbf{S}(u, v)$, the letter "C" as $\mathbf{c}(t)$ (a planar curve), and the two middle joints of the robotic arm as $A = (A_x, A_y, A_z), B = (B_x, B_y, B_z)$. We embedded $\mathbf{c}(t)$ in $S(u, v)$ by functional composition: $\mathbf{c}_S(t) = \mathbf{S}(\mathbf{c}(t))$. Hence, The six constraints in seven unknowns $(t, A_x, A_y, A_z, B_x, B_y, B_z)$ are as follows:

$$\begin{cases} |\mathbf{c}_S(t) - B|^2 = L_3^2, \\ \langle \mathbf{S}_u(\mathbf{c}(t)), B - \mathbf{c}_S(t) \rangle = 0, \\ \langle \mathbf{S}_v(\mathbf{c}(t)), B - \mathbf{c}_S(t) \rangle = 0, \\ |B - A|^2 = L_2^2, \\ |A - P_0|^2 = L_1^2, \\ z((B - A) \times (P_0 - A)) = 0, \end{cases} \tag{4}$$

where L_i *are some constants, and* $\mathbf{S}_u = \dfrac{\partial \mathbf{S}}{\partial u}, \mathbf{S}_v = \dfrac{\partial \mathbf{S}}{\partial v}$.

The first constraint in Equation (4) requires that the distance of the start of the engraving head maintains a constant distance from the text it engraves. The next two constraints require the engraving head to be perpendicular to $\mathbf{S}$ at the point of contact with the surface of the teapot. The next two constraints set the lengths of the arm segments of the robot, i.e. the distances between the joints. Finally, the last constraint makes sure that the plane containing the points A, B, P_0 is vertical, i.e. its normal's z element is zero. This constraint is required because the base joint of the robotic arm can rotate and tilt up and down, but can't roll. Since the planar letter "C" is a quadratic curve, and the body of the teapot is a bi-cubic surface, the maximal polynomial orders of the first three constraints are somewhat high: 25, 23, and 23 (respectively).

The last problem we examine is the computation of flecnodal curves of a surface. A *flecnodal curve* is defined as a

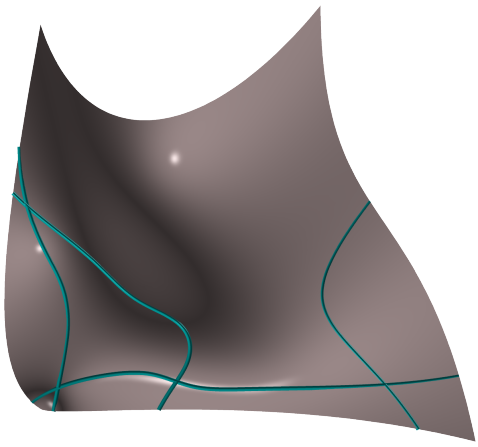


Figure 9: A surface, and its flecnodal curves.

locus of all the points at which the surface has a third-order contact with a ray [26]. Let $\mathbf{S}(u,v)$ be a C^3-continuous surface, and $\mathbf{n}(u,v)$, the unnormalized normal to the surface at u,v. Then, the flecnodal curves of the surface are the solutions to the following system [27], of three constraints and four unknowns (u,v,a,b):

$$\begin{cases} \left\langle a^2\mathbf{S}_{uu}(u,v) + 2ab\mathbf{S}_{uv}(u,v) + b^2\mathbf{S}_{vv}(u,v), \mathbf{n}(u,v)\right\rangle = 0, \\ \left\langle a^3\mathbf{S}_{uuu}(u,v) + 3a^2b\mathbf{S}_{uuv}(u,v) + \right. \\ \qquad \left. 3ab^2\mathbf{S}_{uvv}(u,v) + b^3\mathbf{S}_{vvv}(u,v), \mathbf{n}(u,v)\right\rangle = 0, \\ a^2+b^2-1=0. \end{cases} \tag{5}$$

Decomposing constraint system (5), our framework first solved the constraint $a^2+b^2-1=0$, replacing the variables a,b with a unit circle. Then, it propagated the solution into the first two constraints, allowing them to be solved more efficiently. An example of a surface with its flecnodal curves, computed by our framework, is presented in Figure 9.

4.1. Performance

The decomposition of complex constraint systems into smaller, simpler, subsystems allows our framework to solve problems significantly faster than the solver would have, without decomposition. As mentioned earlier, the running time of many subdivision-based solvers, including the solver we used, scales exponentially with the number of variables in the problem (k in Definition 3.1). Qualitatively, the order of magnitude of the total number of subdivision operations in the solution process without decomposition can be approximated by c^k, where c is the average number of subdivisions done on each variable. Due to the decomposition step which we have introduced, and assuming a uniform size decomposition into d subsystems, the order of magnitude of the number of operations is reduced to $dc^{\frac{k}{d}}$. In Table 1, we present the running times for the problems discussed in this paper, on a Windows 7 PC with a 3.7GHz CPU, on a single thread, and the speedup achieved by our decomposition algorithm. Problems with inequality constraints are marked with *ineq*.

Problem	Time (seconds)		Speedup Factor
	No Decomposition	With Decomposition	
Figure 4	0.000474	0.000449	1.05
Figure 4, with ineq.	0.00034	0.0003	1.13
Figure 5	0.27	0.21	1.32
Figure 5, with ineq.	0.09	0.074	1.21
Figure 6	19730	61.2	322.3
Figure 6, with ineq.	9690.4	2.7	3530.2
Figure 7	4544.2	7.6	599.3
Figure 8	84.9	3.5	24.5
Figure 9	0.70	0.36	1.96

Table 1: The performance of the framework with decomposition, compared to the subdivision-based solver without decomposition.

5. Future work and conclusion

One of the main advantages of the presented work is the increase of the scalability of subdivision-based solvers. As shown in Section 4.1, our framework is capable of efficiently solving problems which are prohibitively time-consuming for the subdivision-based solver without decomposition, while also improving performance for small-scale problems. One potential limit to the scalability of our framework is the repeated reparametrization of the partial univariate solution (see Algorithm 6, step 8). In large systems which are decomposed into many subsystems, this will quickly increase the polynomial order and complexity (length of the control mesh) of the univariate solutions. A possible solution for this limitation is simplifying the partial univariate solutions when they get too complex, for example by least-squares approximation. This, however, may introduce errors, so a numerical improvement step should probably follow. Alternatively, it may be possible to optimize the size of the subproblems so that the decomposition doesn't contain subproblems that are too large, but also doesn't contain too many subproblems.

The obvious next step for future research is extending the capability of our framework to problems with a bivariate solution space. This presents several challenges:

1. A subdivision-based solver capable of solving problems with a bivariate solution space is needed. One such solver is presented in [28].
2. The subdivision-based solver presented in [28] represents its results as triangle meshes in $\mathbb{R}^n$. For propagating the results of one subsystem to the next, the results need to be parameterized and converted into a bivariate B-spline. This is a difficult problem, but it has been recently addressed in [29], among others.
3. In Section 3.2, we described functional composition for B-spline univariates into B-spline multivariates, $M(\boldsymbol{\tau}(v))$. This can be easily expanded for composition of B-spline bivariates into Bézier multivariates, $M_2(M_1(u,v))$, but if M_2 is a B-spline, then M_1 needs to be subdivided at the knot values of M_2. This is

a non-trivial task, as it doesn't necessarily result in rectangular tensor-product patches.

4. Finally, a better way for selecting the optimal decomposition, out of all the possible ones, would be beneficial. Using our exhaustive search approach from problems with a bivariate solution space can be inefficient. A heuristic for choosing the decomposition for such problems is needed.

Another issue, which can occur in mechanisms, is systems which have a degree, or multiple degrees, of motion, despite being well-constrained. This is caused by singularities in the constraint system which defines them. Such a case is presented in [30]. We aim to address the issue of singularities in the future. Handling mechanisms with contacts and motion across multiple surfaces, possibly with C^1 discontinuities is another interesting venue to explore.

5.1. Conclusion

In this paper, we proposed a framework for solving highly complex polynomial constraint systems, by adding a decomposition algorithm to a subdivision-based solver. The presented framework is capable of solving problems with either zero-dimensional or univariate solution spaces. Additionally, the framework can handle inequality constraints, which are rarely supported in other solvers with decomposition algorithms. We have demonstrated the capabilities of our system on a variety of problem, from simple "point-and-bar" static structures, through complex kinematic and inverse kinematic problems, to purely algebraic problems, and shown the improvement in performance and scalability over the regular subdivision-based solver. Finally, we outlined the steps that need to be taken to extend our framework to problems with a bivariate solution space.

6. Acknowledgments

This work was supported in part by DARPA, under contract HR0011-17-2-0028. All opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies, nor should any endorsement be inferred.

References

[1] I. E. Sutherland, Sketch pad a man-machine graphical communication system, in: Proceedings of the SHARE design automation workshop, ACM, 1964, pp. 6–329.

[2] C. Jermann, G. Trombettoni, B. Neveu, P. Mathis, Decomposition of geometric constraint systems: a survey, International Journal of Computational Geometry & Applications 16 (05n06) (2006) 379–414.

[3] J. C. Owen, Algebraic solution for geometry from dimensional constraints, in: Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications, ACM, 1991, pp. 397–407.

[4] I. Fudos, C. M. Hoffmann, A graph-constructive approach to solving systems of geometric constraints, ACM Transactions on Graphics (TOG) 16 (2) (1997) 179–216.

[5] X.-S. Gao, G.-F. Zhang, Geometric constraint solving based on connectivity of graph, MM Research Preprings (2003) 148–162.

[6] C. M. Hoffmann, A. Lomonosov, M. Sitharam, Finding solvable subsets of constraint graphs, in: International Conference on Principles and Practice of Constraint Programming, Springer, 1997, pp. 463–477.

[7] H. Gao, M. Sitharam, Characterizing 1-dof henneberg-i graphs with efficient configuration spaces, in: Proceedings of the 2009 ACM symposium on Applied Computing, ACM, 2009, pp. 1122–1126.

[8] S. Ait-Aoudia, R. Jegou, D. Michelucci, Reduction of constraint systems, in: COMPUGRAPHICS'93, 1993, pp. 331–340.

[9] P. Bunus, P. Fritzson, Methods for structural analysis and debugging of modelica models, in: Proceedings of the 2nd International Modelica Conference, Vol. 10, 2002, pp. 157–165.

[10] B. Vander Zanden, An incremental algorithm for satisfying hierarchies of multiway dataflow constraints, ACM Transactions on Programming Languages and Systems (TOPLAS) 18 (1) (1996) 30–72.

[11] M. Barton, N. Shragai, G. Elber, Kinematic simulation of planar and spatial mechanisms using a polynomial constraints solver, Computer-Aided Design and Applications 6 (1) (2009) 115–123.

[12] G. Hegedüs, J. Schicho, H.-P. Schröcker, The theory of bonds: A new method for the analysis of linkages, Mechanism and Machine Theory 70 (2013) 407–424.

[13] J. M. Rico, B. Ravani, On mobillity analysis of linkages using group theory, in: ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, 2002, pp. 429–446.

[14] M. J. Van Emmerik, Interactive design of 3d models with geometric constraints, The Visual Computer 7 (5) (1991) 309–325.

[15] H. Lamure, D. Michelucci, Solving geometric constraints by homotopy, in: Proceedings of the third ACM symposium on Solid modeling and applications, ACM, 1995, pp. 263–269.

[16] G. Elber, M.-S. Kim, Geometric constraint solver using multivariate rational spline functions, in: Proceedings of the sixth ACM symposium on Solid modeling and applications, ACM, 2001, pp. 1–10.

[17] I. Hanniel, G. Elber, Subdivision termination criteria in subdivision multivariate solvers using dual hyperplanes representations, Computer-Aided Design 39 (5) (2007) 369–378.

[18] M. Bartoň, G. Elber, I. Hanniel, Topologically guaranteed univariate solutions of underconstrained polynomial systems via no-loop and single-component tests, Computer-Aided Design 43 (8) (2011) 1035–1044.

[19] X. Chen, R. F. Riesenfeld, E. Cohen, An algorithm for direct multiplication of b-splines, IEEE transactions on automation science and engineering 6 (3) (2009) 433–442.

[20] G. Elber, Free form surface analysis using a hybrid of symbolic and numeric computation, Ph.D. thesis, The University of Utah (1992).

[21] T. H. Cormen, Introduction to Algorithms., 3rd Edition, The MIT Press, 2009.

[22] M. Gangnet, B. Rosenberg, Constraint programming and graph algorithms, Annals of Mathematics and Artificial Intelligence 8 (3-4) (1993) 271–284.

[23] J. J. Koelewijn, Graph-theoretical aspects of constraint solving in the sst project, Master's thesis, University of Twente (2011).

[24] E. A. M. García, Numerical Modelling in Robotics, OmniaScience, 2015, Ch. 15.2, pp. 385–396.

[25] T. Jansen, Strandbeest website (2014).
URL `http://www.strandbeest.com`

[26] J. J. Koenderink, Solid Shape, MIT Press, 1990, Ch. 6, pp. 281–283.

[27] G. Elber, X. Chen, E. Cohen, Mold accessibility via gauss map analysis, Journal of Computing and Information Science in Engineering 5 (2) (2005) 79–85.

[28] J. Mizrahi, G. Elber, Topologically guaranteed bivariate solutions of under-constrained multivariate piecewise polynomial systems, Computer-Aided Design 58 (2015) 210–219.

[29] Y. Zhang, J. Cao, Z. Chen, X. Li, X.-M. Zeng, B-spline surface fitting with knot position optimization, Computers & Graphics

58 (2016) 73–83.
[30] A. Karger, M. Husty, Classification of all self-motions of the original stewart-gough platform, Computer-aided design 30 (3) (1998) 205–215.