Computer-Aided Design 52 (2014) 1–16

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Decomposition of geometric constraint graphs based on computing fundamental circuits. Correctness and complexity

R. Joan-Arinyo [*], M. Tarrés-Puertas, S. Vila-Marta

Grup d'Informàtica a l'Enginyeria. Universitat Politècnica de Catalunya. Barcelona, Catalonia

HIGHLIGHTS

- A new algorithm to solve the 2D geometric constraint problem is described.
- The graph is decomposed according to a set of fundamental circuits.
- Fundamental circuits are induced by a spanning tree.
- We prove the algorithm soundness.
- The worst running time is quadratic with the number of geometric elements.

ARTICLE INFO

Article history:
Received 30 April 2013
Accepted 17 February 2014

Keywords:
Geometric constraint solving
Graph decomposition
Fundamental circuits
Bridges
Planar embeddings

ABSTRACT

In geometric constraint solving, Decomposition–Recombination solvers (DR-solvers) refer to a general solving approach where the problem is divided into a set of sub-problems, each sub-problem is recursively divided until reaching basic problems which are solved by a dedicated equational solver. Then the solution to the starting problem is computed by merging the solutions to the sub-problems.

Triangle- or tree-decomposition is one of the most widely used approaches in the decomposition step in DR-solvers. It may be seen as decomposing a graph into three subgraphs such that subgraphs pairwise share one graph vertex. Shared vertices are called *hinges*. Then a merging step places the geometry in each sub-problem with respect to the other two.

In this work we report on a new algorithm to decompose biconnected geometric constraint graphs by searching for hinges in fundamental circuits of a specific planar embedding of the constraint graph. We prove that the algorithm is correct.

1. Introduction

Geometric constraint solving was developed as a core technology in parametric and feature-based Computer-Aided Design systems. Lately it has found a number of applications in fields like linkage design, chemical molecular modeling, computer vision and dynamic geometry.

Many attempts to provide general, powerful and efficient techniques to solve the geometric constraint problem have been reported in the literature. For an extensive review refer to [1] and the references therein.

Among the different approaches to geometric constraint solving, we are interested here in a divide and conquer approach consisting of three steps, [2]

1. Divide the problem P into several sub-problems, say $P_1, P_2, \dots, P_n$.
2. Solve each sub-problem P_i recursively with known algorithms.
3. Build the solution to problem P by merging the solutions to sub-problems, $P_1, P_2, \dots, P_n$.

Constraint solvers that fit in this description are known as *Decomposition–Recombination solvers* (DR solvers). This approach has been particularly successful when the decomposition into sub-problems and subsequent recombination of solutions to these sub-problems can be described by a plan generated *a priori*, that is, a plan generated as a preprocessing step without actually solving the subsystems. The plan output by the DR-planner remains unchanged as numerical values of parameters change. Such a plan is known as the *DR-plan* and the unit in the solver that generates it is the *DR-planner*, [3]. In this setting, the DR-plan is then used to drive the actual solving process, that is, computing specific

* Corresponding author.
E-mail addresses: robert@lsi.upc.edu (R. Joan-Arinyo), mtarres@dipse.upc.edu (M. Tarrés-Puertas), sebas@dipse.upc.edu (S. Vila-Marta).

http://dx.doi.org/10.1016/j.cad.2014.02.006
0010-4485/© 2014 Elsevier Ltd. All rights reserved.

coordinates that properly place geometric objects with respect to each other.

Our interest focuses on DR-solvers where the geometric constraint problem is abstracted as a graph and thus, solving the problem generically corresponds to solving the corresponding graph. Therefore the DR-planner is based on graph analysis whose operation will be purely combinatorial. We consider well-constrained problems, loosely speaking, problems where resulting geometric objects are well-defined.

In a pioneering work, Hoffmann et al., [3], classify graph-based DR-planners into two main categories. Planners in one category, called *Constraint Shape Recognition*, work by recognizing specific solvable subgraphs of known shape, most commonly, triangular patterns. DR-planners have been widely used and related literature is profuse. Examples are [4–16]. DR-planners in the second category are called *Generalized Maximum Matching* planners. Planners in this category work in two steps. First they isolate certain solvable subgraphs by transforming the constraint graph into a bipartite graph where a maximum generalized matching is found. Then the DR-plan is figured out by connectivity analysis. To this category belong DR-planners reported in [17–20].

In [21], Hoffmann et al. reported on two new DR-planners that exhibit features of both categories. The *Condensed Algorithm* applies repeatedly a flow-based algorithm to find minimal dense solvable subgraphs, [22], that are subsequently extended by adding more geometric objects one at a time. The *Frontier Algorithm* and the *Modified Frontier Algorithm* are essentially improvements of the *Condensed Algorithm*. These algorithms have been further developed for problems considering purely distance constraint systems by Lomonosov, [23] and by Zhou, [24]. Surveys can be found in [25] and [26]. A discussion on geometric constraint solving progress and research directions can be found in [27].

We consider geometric constraint problems in the Euclidean plane consisting of a finite set of geometric objects and a finite set of constraints defined between them. The geometric objects are drawn from a fixed set of types such as points, straight lines, circles and arcs of circle. The constraints include topological constraints such as tangency, incidence and perpendicularity, and metric constraints such as point–point distance, perpendicular point–straight line distance and line–line angle. Geometric constraint problems are always well-constrained or minimally rigid, [28].

In this paper we develop the DR-planner introduced in [29] in three ways: (1) we give detailed algorithms, (2) we show that the approach is sound and, (3) we prove that the runtime is $O(n^2)$ where n is the number of geometric elements in the problem. Experimental results show that for problems with $3 \leq n \leq 200$, the runtime is $O(n)$.

The DR-planner belongs to the constraint shape recognition category and is inspired in the algorithm for finding triconnected components of a graph reported by Miller and Ramachandran in [30]. The strategy used to divide a graph into subgraphs is the tree- or triangle-decomposition approach reported in [31]. Decomposition may be understood as decomposing a problem into three sub-problems such that pairwise share a common geometric primitive, that we call *hinge*. Recombination is to place three primitives with respect to each other. In this approach, constraint problems consisting of three primitives are used as building blocks for larger constraint problems.

The DR-planner searches for a set of three hinges in fundamental circuits associated to a given spanning tree of the constraint graph. The algorithm has two main steps, (i) transforming the given graph into a simpler, planar graph and (ii) computing a planar embedding for the transformed graph where hinges are identified as a set of three vertices shared by two faces. We have proved that the algorithm is correct in the following sense. First the graph transformation preserves the hinges. Second hinges, if present, always belong to the common boundary of two faces in a planar embedding of the transformed constraint graph.

The rest of this paper is organized as follows. In Section 2 we recall some basic concepts from graph theory and planar embeddings that we shall use later on. For the sake of completeness, we include in Section 3 a short description of tree-decomposability, a basic concept in our DR-planner. In Section 4 we describe our DR-planner and in Section 5 we provide an additional case study to illustrate how the approach works. In Section 6 we prove that the algorithm is correct. The algorithm worst running time is analyzed in Section 7. Section 8 gives experimental algorithm runtime values. Finally, in Section 9 we offer a short discussion and a couple of issues that deserve further work.

2. Preliminaries

In this section we recall basic terminology of graph theory, the concept of geometric constraint graph associated to a geometric problem defined by constraints, and some definitions related to geometric constraint graphs. For more information on general graphs theory we refer the reader to Even [32] and to Thulasiraman and Swamy [33]. Concerning geometric constraint graphs the reader can check the works by Hoffman et al. [34] and Joan-Arinyo et al. [31].

2.1. Basic graph concepts

In this work, a *graph* $G = (V, E)$ is a finite set V of nodes or vertices and a collection of edges, E. An edge is an unordered pair (u, v) of distinct vertices $u, v \in V(G)$. In general $V(G)$ and $E(G)$ will denote respectively the set of vertices and edges of the graph G.

The degree of a vertex $v \in V(G)$ is the number of edges in $E(G)$ incident to v.

A *path* is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that (v_i, v_{i+1}) is an edge for $1 \leq i \leq n$. A path is *simple* if all vertices on the path are distinct.

In this work, a *circuit* is a simple path that does not contain any repeated vertices except v_1 and v_n which are the same. In what follows, we shall denote the set of vertices in a circuit as $C = \langle v_1, v_2, \ldots, v_{n-1} \rangle$ and we shall consider that vertices are circularly sorted. For example, $\langle b, d, e, c \rangle$ and $\langle d, e, c, b \rangle$ are equivalent descriptions for the circuit in Fig. 1. Whether vertices are sorted clockwise or counterclockwise is irrelevant.

A graph $G = (V, E)$ is *connected* if there exists a path between every pair of vertices in G, otherwise G is *disconnected*. The maximal connected subgraphs of a disconnected graph G are the *connected components* of G.

Let $G = (V, E)$ be a connected graph. A vertex $v \in V(G)$ is an *articulation vertex* if $V(G) - \{v\}$ is disconnected. If $v \in V(G)$ is an articulation vertex, there are vertices $u, w \in V(G)$, with $u \neq v$ and $w \neq v$ such that v is on every path connecting vertices u and w.

A *non-separable* or *biconnected* graph $G = (V, E)$ has no articulation vertices, otherwise it is *separable*. A *biconnected component* of a connected graph G is a maximal biconnected subgraph of G. A connected graph can be decomposed into biconnected components. For any biconnected graph $G = (V, E)$, given a pair of vertices $u, v \in V(G)$ with $u \neq v$, there are, at least, two disjoint paths connecting u and v.

The *connectivity* of a graph G is the minimum number k of vertices that must be removed to disconnect G. If the connectivity of G is k, we write $\kappa(G) = k$. For a disconnected graph G, $\kappa(G) = 0$. For a connected graph G, $\kappa(G) \geq 1$. A separable graph G has $\kappa(G) = 1$. A biconnected graph G has $\kappa(G) \geq 2$. In a similar way a graph G with $\kappa(G) \geq 3$ is called triconnected. Biconnected graphs can be decomposed into triconnected components.