Dynamic Programming and Graph Algorithms in Computer Vision[*]

Pedro F. Felzenszwalb and Ramin Zabih

Abstract

Optimization is a powerful paradigm for expressing and solving problems in a wide range of areas, and has been successfully applied to many vision problems. Discrete optimization techniques are especially interesting, since by carefully exploiting problem structure they often provide non-trivial guarantees concerning solution quality. In this paper we briefly review dynamic programming and graph algorithms, and discuss representative examples of how these discrete optimization techniques have been applied to some classical vision problems. We focus on the low-level vision problem of stereo; the mid-level problem of interactive object segmentation; and the high-level problem of model-based recognition.

Keywords: Combinatorial Algorithms, Vision and Scene Understanding, Artificial Intelligence, Computing Methodologies

1 Optimization in computer vision

Optimization methods play an important role in computer vision. Vision problems are inherently ambiguous and in general one can only expect to make "educated guesses" about the content of images. A natural approach to address this issue is to devise an objective function that measures the quality of a potential solution and then use an optimization method to find the best solution. This approach can often be justified in terms of statistical inference, where we look for the hypothesis about the content of an image with the highest probability.

It is widely accepted that the optimization framework is well suited to deal with noise and other sources of uncertainty such as ambiguities in the image data. Moreover, by formulating

[*]Pedro F. Felzenszwalb is with the Computer Science Department, University of Chicago, Chicago, IL 60637. E-mail: `pff@cs.uchicago.edu`. Ramin Zabih is with the Computer Science Department, Cornell University, Ithaca, NY 14853. E-mail: `rdz@cs.cornell.edu`

problems in terms of optimization we can easily take into account multiple sources of information. Another advantage of the optimization approach is that in principle it provides a clean separation between how a problem is formulated (the objective function) and how the problem is solved (the optimization algorithm). Unfortunately the optimization problems that arise in vision are often very hard to solve.

In the past decade there has been a new emphasis on discrete optimization methods, such as dynamic programming or graph algorithms, for solving computer vision problems. There are two main differences between discrete optimization methods and the more classical continuous optimization approaches commonly used in vision [83]. First, of course, these methods work with discrete solutions. Second, discrete methods can often provide non-trivial guarantees about the quality of the solutions they find. These theoretical guarantees are often accompanied by strong performance in practice. A good example is the formulation of low-level vision problems such as stereo using Markov Random Fields [92, 97] (we will discuss this in detail in section 7).

1.1 Overview

This survey covers some of the main discrete optimization methods commonly used in computer vision, along with a few selected applications where discrete optimization methods have been particularly important. Optimization is an enormous field, and even within computer vision it is a ubiquitous presence. To provide a coherent narrative rather than an annotated bibliography, our survey is inherently somewhat selective, and there are a number of related topics that we have omitted. We chose to focus on a dynamic programming and on graph algorithms, since they share two key properties: first, they draw on a body of well-established, closely inter-related techniques, which are typically covered in an undergraduate course on algorithms (using a textbook such [24, 60]); and second, these methods have had a significant and increasing impact within computer vision over the last decade. Some topics which we do not explicitly cover include constrained optimization methods (such as linear programming) and message passing algorithms (such as belief propagation).

We begin by reviewing some basic concepts of discrete optimization and introducing some notation. We then summarize two classical techniques for discrete optimization, namely graph algorithms (section 3) and dynamic programming (section 4); along with each technique, we present an example application relevant to computer vision. In section 5 we describe how discrete optimization techniques can be used to perform interactive object segmentation. In section 6 we discuss the role of discrete optimization for the high-level vision problem of model-based recognition. In section 7 we focus on the low-level vision problem of stereo. Finally in section 8 we describe a few important recent developments.

2 Discrete optimization concepts

An optimization problem involves a set of candidate solutions $\mathcal{S}$ and an objective function $E : \mathcal{S} \to \mathbb{R}$ that measures the quality of a solution. In general the search space $\mathcal{S}$ is defined implicitly and consists of a very large number of candidate solutions. The objective function E can either measure the goodness or badness of a solution; when E measures badness, the optimization problem is often referred to as an *energy minimization* problem, and E is called an *energy function.* Since many papers in computer vision refer to optimization as energy minimization, we will follow this convention and assume that an optimal solution $x \in \mathcal{S}$ is one minimizing an energy function E.

The ideal solution to an optimization problem would be a candidate solution that is a global minimum of the energy function, $x^* = \arg\min_{x \in \mathcal{S}} E(x)$. It is tempting to view the energy function and the optimization methods as completely independent. This suggests designing an energy function that fully captures the constraints of a vision problem, and then applying a general-purpose energy minimization technique.

However, such an approach fails to pay heed to computational issues, which have enormous practical importance. No algorithm can find the global minimum of an arbitrary energy function without exhaustively enumerating the search space, since any candidate that was not evaluated could turn out to be the best one. As a result, any completely general minimization method, such as genetic algorithms [47] or MCMC [76], is equivalent to exhaustive search. On the other hand, it is sometimes possible to design an efficient algorithm that solves problems in a particular class, by exploiting the structure of the search space $\mathcal{S}$ and the energy function E. All the algorithms described in this survey are of this nature.

Efficient algorithms for restricted classes of problems can be very useful since vision problems can often be plausibly formulated in several different ways. One of these problem formulations, in turn, might admit an efficient algorithm, which would make this formulation preferable. A natural example, which we will discuss in section 7, comes from pixel labeling problems such as stereo.

Note that in general a global minimum of an energy function might not give the best results in terms of accuracy, because the energy function might not capture the right constraints. But if we use an energy minimization algorithm that provides no theoretical guarantees it can be difficult to decide if poor performance is due to the choice of energy function as opposed to weaknesses in the minimization algorithm.

There is a long history of formulating pixel labeling problems in terms of optimization, and some very complex and elegant energy functions have been proposed (see [41] for some early examples). Yet the experimental performance of these methods was poor, since they relied on general-purpose optimization techniques such as simulated annealing [6, 41]. In the last decade, researchers have developed efficient discrete algorithms for a relatively simple class of energy functions, and these optimization-based schemes are viewed as the state of the art for stereo [21, 92].

2.1 Common classes of energy functions in vision

As previously mentioned, the problems that arise in computer vision are inevitably ambiguous, with multiple candidate solutions that are consistent with the observed data. In order to eliminate ambiguities it is usually necessary to introduce a bias towards certain candidate solutions. There are many ways to describe and justify such a bias. In the context of statistical inference the bias is usually called a prior. We will use this term informally to refer to any terms in an energy function that impose such a bias.

Formulating a computer vision problem in terms of optimization often makes the prior clear. Most of the energy functions that are used in vision have a general form that reflects the observed data and the need for a prior,

$$E(x) \quad = \quad E_{\text{data}}(x) + E_{\text{prior}}(x). \tag{1}$$

The first term of such an energy function penalizes candidate solutions that are inconsistent with the observed data, while the second term imposes the prior.

We will often consider an n-dimensional search space of the form $\mathcal{S} = \mathcal{L}^n$, where $\mathcal{L}$ is an arbitrary finite set. We will refer to $\mathcal{L}$ as a set of labels, and will use k to denote the size of $\mathcal{L}$. This search space has an exponential number, k^n, of possible solutions. A candidate solution $x \in \mathcal{L}^n$ will be written as $(x_1, \ldots, x_n)$ where $x_i \in \mathcal{L}$. We will also consider search spaces $\mathcal{S} = \mathcal{L}_1 \times \cdots \times \mathcal{L}_n$, which simply allows the use of a different label set for each x_i.

A particularly common class of energy functions in computer vision can be written as

$$E(x_1, \ldots, x_n) \quad = \quad \sum_i D_i(x_i) \quad + \quad \sum_{i,j} V_{i,j}(x_i, x_j). \tag{2}$$

Many of the energy functions we will consider will be of this form. In general the terms D_i are used to ensure that the label x_i is consistent with the image data, while the $V_{i,j}$ terms ensure that the labels x_i and x_j are compatible.

Energy functions of the form given in equation (2) have a long history in computer vision. They are particularly common in pixel labeling problems, where a candidate solution assigns a label to each pixel in the image, and the prior captures a spatial smoothness assumption.

For these low-level vision problems, the labels might represent quantities such as stereo depth (see section 7), intensity, motion, texture, etc. In all of these problems, the underlying data is ambiguous, and if each pixel is viewed in isolation it is unclear what the correct label should be.

One approach to deal with the ambiguity at each pixel is to examine the data at nearby pixels and then independently choose its label. If the labels are intensities, this often corresponds to a simple filtering algorithm (e.g., convolution with a Gaussian). For stereo or motion, such algorithms are usually referred to as area-based; a relatively early example is provided by [46]. Unfortunately, these methods have severe problems near object boundaries, since the majority of pixels near a pixel p may not support the correct label for p. This shows up in the very poor performance of area-based methods on the standard stereo benchmarks [92].

Optimization provides a natural way to address pixel labeling problems. The use of optimization for such a problem appears to date to the groundbreaking work of [48] on motion. They proposed a continuous energy function, where a data term measures how consistent the candidate labeling is with the observations at each pixel, and a prior ensures the smoothness of the solution. Note that their prior preferred labelings that are globally smooth; this caused motion boundaries to be oversmoothed, which is a very important issue in pixel labeling problems. The energy minimization problem was solved by a continuous technique, namely the calculus of variations.

Following [48], many energy functions were proposed, both for motion and for other pixel labeling problems. An interesting snapshot of the field is provided by [82], who point out the ties between the smoothness term used by [48] and the use of Tikhonov regularization to solve inverse problems [98]. Another important development was [41], which showed that such energy functions have a natural statistical interpretation as maximizing the posterior probability of a Markov Random Field (MRF). This paper was so influential that to this day many authors use the terminology of MRF's to describe an optimization approach to pixel labeling problems (see [18, 52, 63, 97] for a few examples).

A very different application of energy functions of the form in equation (2) involves object recognition and matching with deformable models. One of the earliest examples is the pictorial structures formulation for part-based modeling in [36]. In this case we have an object with many parts and a solution corresponds to an assignment of image locations to each part. In the energy function from [36] a data term measures how much the image data under each part agrees with a model of the part's appearance, and the prior enforces geometrical constraints between different pairs of parts. A related application involves boundary detection using active contour models [3, 56]. In this case the data term measures the evidence for a boundary at a particular location in the image, and the prior enforces smoothness of the boundary. Discrete optimization methods are quite effective both for pictorial structures (see section 6.3) and for active contour models (section 5.2).

The optimization approach for object detection and recognition makes it possible to combine different sources of information (such as the appearance of parts and their geometric arrangement) into a single objective function. In the case of part-based modeling this makes it possible to aggregate information from different parts without relying on binary decision coming from an individual part detector. The formulation can also be understood in terms of statistical estimation, where the most probable object configuration is found [34].

2.2 Guarantees concerning solution quality

One of the most important properties of discrete optimization methods is that they often provide some kind of guarantee concerning the quality of the solutions they find. Ideally, the algorithm can be shown to compute the global minimum of the energy function. While we will discuss a number of methods that can do this, many of the optimization problems that arise in vision are NP-hard (see the appendix of [20] for an important example). Instead, many algorithms look for a local minimum — a candidate that is better than all "nearby" alternatives. A general view of a local minimum, which we take from [2], is to define a neighborhood system $N : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ that specifies for any candidate solution x the set of nearby candidates $N(x)$. Using this notation, a local minimum solution with respect to the neighborhood system N is a candidate x^* such that $E(x^*) \leq \min_{x \in N(x^*)} E(x)$.

In computer vision, it is common to deal with energy functions with multiple local minima. Problems with a single minimum can often be addressed via the large body of work in convex optimization (see [13] for a recent textbook). Note that for some optimization problems, if we pick the neighborhood system N carefully a local minimum is also a global minimum.

A local minimum is usually computed by an iterative process, where an initial candidate is improved by explicitly or implicitly searching the neighborhood of nearby solutions. This subproblem of finding the best solution within the local neighborhood is quite important. If we have a fast algorithm for finding the best nearby solution within a large neighborhood, we can perform this local improvement step repeatedly.[1] There are a number of important vision applications where such an iterated local improvement method has been employed. For example, object segmentation via active contour models [56] can be performed in this way, where the local improvement step uses dynamic programming [3] (see section 5.2 for details). Similarly, low-level vision problems such as stereo can be solved by using min-cut algorithms in the local improvement step [20] (see section 7).

While proving that an algorithm converges to a strong local minimum is important, this does not directly imply that the solution is close to optimal. In contrast, an approximation algorithm for an optimization problem is a polynomial time algorithm that computes a solution $\hat{x}$ whose energy is within a multiplicative factor of the global minimum. It is generally quite difficult to prove such a bound, and approximation algorithms are a major research area in theoretical computer science (see [60] for a particularly clear exposition of the topic). Very few approximation algorithms have been developed in computer vision; the best known is the expansion move algorithm of [20]. An alternative approach is to provide per-instance bounds, where the algorithm produces a guarantee that the solution is within some factor of the global minimum, but where that factor varies with each problem instance.

The fundamental difference between continuous and discrete optimization methods concerns the nature of the search space. In a continuous optimization problem we are usually looking for a set of real numbers and the number of candidate solutions is uncountable; in a discrete optimization problem we are looking for an element from a discrete (and often, finite) set. Note that there is ongoing work (e.g., [14]) that explores the relationship between continuous and discrete optimization methods for vision.

While continuous methods are quite powerful, it is uncommon for them to produce guarantees concerning the absolute quality of the solutions they find (unless, of course, the energy function is convex). Instead they tend to provide guarantees about speed of convergence towards a local minimum. Whether or not guarantees concerning solution quality are important depends on

[1][2] refers to such algorithms as very large-scale neighborhood search techniques; in vision they are sometimes called move-making algorithms [71].

the particular problem and context. Moreover, despite their lack of such guarantees, continuous methods can perform quite well in practice.

2.3 Relaxations

The complexity of minimizing a particular energy function clearly depends on the set it is being minimized over, but the form of this dependence can be counterintuitive. In particular, sometimes it is difficult to minimize E over the original discrete set $\mathcal{S}$, but easy to minimize E over a continuous set $\mathcal{S}'$ that contains $\mathcal{S}$. Minimizing E over $\mathcal{S}'$ instead of $\mathcal{S}$ is called a relaxation of the original problem. If the solution to the relaxation (i.e., the global minimum of E over $\mathcal{S}'$) happens to occur at an element of $\mathcal{S}$, then by solving the relaxation we have solved the original problem. As a general rule, however, if the solution to the relaxation does not lie in $\mathcal{S}$, it provides no information about the original problem beyond a lower bound (since there can be no better solution within $\mathcal{S}$ than the best solution in the larger set $\mathcal{S}'$). In computer vision, the most widely used relaxations involve linear programming or spectral graph partitioning.

Linear programming provides an efficient way to minimize a linear objective function subject to linear inequality constraints (which define a polyhedral region). One of the best-known theorems concerning linear programming is that when an optimal solution exists it can be achieved at a vertex of the polyhedral region defined by the inequality constraints (see, e.g., [23]). A linear program can thus be viewed as a discrete optimization problem over the vertices of a polyhedron. Yet linear programming has some unusual features that distinguish it from the methods that we survey. For example, linear programming can provide a per-instance bound (see [67] for a nice application in low-level vision). Linear programming is also the basis for many approximation algorithms [60]. There is also a rich mathematical theory surrounding linear programs, including linear programming duality, which plays an important role in advanced optimization methods that don't explicitly use linear programming algorithms.

Linear programming has been used within vision for a number of problems, and one recent application that has received significant interest involves message passing methods. The convergence of belief propagation on the loopy graphs common in vision is not well understood, but several recent papers [101, 102] have exposed strong connections with linear programming.

Spectral partitioning methods can efficiently cluster the vertices of a graph by computing a specific eigenvector of an associated matrix (see, e.g., [54]). These methods are exemplified by the well-known normalized cut algorithm in computer vision [96]. Typically a discrete objective function is first written down, which is NP-hard to optimize. With careful use of spectral graph partitioning, a relaxation of the problem can be solved efficiently. Currently there is no technique that converts a solution to the relaxation into a solution to the original discrete problem that is provably close to the global minimum. However, despite the lack of such guarantee these methods can perform well in practice.

2.4 Problem reductions

A powerful technique, which we will use throughout this survey paper, is the classical computer science notion of a problem reduction [38]. In a problem reduction, we show how an algorithm that solves one problem can also be used to solve a different problem, usually by transforming an arbitrary instance of the second problem to an instance of the first.[2]

There are two ways that problem reductions are useful. Suppose that the second problem is know to be difficult (for example, it may be NP-hard, and hence widely believed to require exponential time). In this case, the problem reduction gives a proof that the first problem is at least as difficult as the second. However, discrete optimization methods typically perform a reduction in the opposite direction, by reducing a difficult appearing problem to one that can be solved quickly. This is the focus of our survey.

One immediate issue with optimization is that it is almost “too powerful” a paradigm. Many NP-hard problems fall within the realm of discrete optimization. On the other hand, problems with easy solutions can also be phrased in terms of optimization; for example, consider the problem of sorting n numbers. The search space consists of the set of $n!$ permutations, and the objective function might count the number of pairs that are mis-ordered. As the example of sorting should make clear, expressing a problem in terms of optimization can sometimes obscure the solution.

Since the optimization paradigm is so general we should not expect to find a single algorithm that works well on most optimization problem. This leads to the perhaps counter-intuitive notion that to solve any specific problem it is usually preferable to use the most specialized technique that can be applied. This ensures that we exploit the structure of the specific problem to the greatest extent possible.

[2]We will only consider reductions that do not significantly increase the size of the problem (see [38] for more discussion of this issue).

3 Graph algorithms

Many discrete optimization methods have a natural interpretation in terms of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, given by a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$ (vertices are sometimes called nodes, and edges are sometimes called links). In a directed graph, each edge e is an ordered pair of vertices $e = (u, v)$, while in an undirected graph the vertices in an edge $e = \{u, v\}$ are unordered. We will often associate a non-negative weight w_e to each edge e in a graph. We say that two vertices are neighbors if they are connected by an edge. In computer vision, the vertices $\mathcal{V}$ usually correspond to pixels or features such as interest points extracted from an image, while the edges $\mathcal{E}$ encode spatial relationships.

A path from a vertex u to a vertex v is a sequence of distinct vertices starting at u and ending at v such that for any two consecutive vertices a and b in the sequence either $(a, b) \in \mathcal{E}$ or $\{a, b\} \in \mathcal{E}$ depending on whether or not the graph is directed. The length of a path in a weighted graph is the sum of the weights associated with the edges connecting consecutive vertices in the path. A cycle is defined like a path except that the first and last vertices in the sequence are the same. We say that a graph is acyclic if it has no cycles. An undirected graph is connected if there is a path between any two vertices in $\mathcal{V}$. A tree is a connected acyclic undirected graph.

Now consider a directed graph with two distinguished vertices s and t called the terminals. An s-t cut is a partition of the vertices into two components S and T such that $s \in S$ and $t \in T$.[3] The cost of the cut is the sum of the weights on edges going from vertices in S to vertices in T.

A matching M in an undirected graph is a subset of the edges such that each vertex is in at most one edge in M. A perfect matching is one where every vertex is in some edge in M. The weight of a matching M in a weighted graph is the sum of the weights associated with edges in M. We say that a graph is bipartite if the vertices can be partitioned into two sets A and B such that every edge connects a vertex in A to a vertex in B. A perfect matching in a bipartite graph defines a one-to-one correspondence between vertices in A and vertices in B.

3.1 Shortest path algorithms

The shortest paths problem involves finding minimum length paths between pairs of vertices in a graph. The problem has many important applications and it is also used as a subroutine to solve a variety of other optimization problems (such as computing minimum cuts). There are two common

[3]There is also an equivalent definition of a cut as a set of edges, which some authors use [14, 20].

versions of the problem: (1) in the single-source case we want to find a shortest path from a source vertex s to every other vertex in a graph; (2) in the all-pairs case we look for a shortest path between every pair of vertices in a graph. Here we consider mainly the single-source case as this is the one most often used in computer vision. We discuss its application to interactive segmentation in section 5.1. We note that in general solving the single-source shortest paths problem is not actually harder than computing a shortest path between a single pair of vertices.

The main property that we can use to efficiently compute shortest paths is that they have an optimal substructure property: a subpath of a shortest path is itself a shortest path. All of the shortest paths algorithms use this fact.

The most well known algorithm for the single-source shortest paths problem is due to Dijkstra [31]. For a directed graph Dijkstra's algorithm runs in $O(|\mathcal{E}| \log |\mathcal{V}|)$ time assuming that there is at least one edge touching each node in the graph. The algorithm assumes that all edge weights are non-negative and builds shortest paths from a source in order of increasing length.

Dijkstra's algorithm assumes that all edge weights are non-negative. In fact, computing shortest-paths on an arbitrary graph with negative edge weights is NP-hard (since this would allow us to find hamiltonian paths [68]). However, there is an algorithm that can handle graphs with some negative edge weights, as long as there are no negative length cycles. The Bellman-Ford algorithm [24] can be used to solve the single-source problem on such graphs in $O(|\mathcal{E}||\mathcal{V}|)$ time. The method is based on dynamic programming (see section 4). It sequentially computes shortest paths that uses at most i edges in order of increasing i. The Bellman-Ford algorithm can also be used to detect negative length cycles on arbitrary graphs. This is an important subroutine for computing minimum ratio cycles and has been applied to image segmentation [53] and deformable shape matching [93].

The all-pairs shortest paths problem can be solved using multiple calls to Dijkstra's algorithm. We can simply solve the single-source problem starting at every possible node in the graph. This leads to an overall method that runs in $O(|\mathcal{E}||\mathcal{V}| \log |\mathcal{V}|)$ time. This is a good approach to solve the all-pairs shortest paths problem in sparse graphs. If the graph is dense we are better off using the Floyd-Warshall algorithm. That algorithm is based on dynamic programming and runs in $O(|\mathcal{V}|^3)$ time. Moreover, like the Bellman-Ford algorithm, it can be used as long as there are no negative length cycles in the graph. Another advantage of the Floyd-Warshall method is that it can be implemented in just a few lines of code and does not rely on an efficient priority queue data structure (see [24]).

3.2 Minimum cut algorithms

The minimum cut problem ("min-cut") is to find the minimum cost s-t cut in a directed graph. On the surface, this problem may look intractable. In fact, many variations on the problem, such as requiring that more than two terminals be separated, are NP-hard [28].

It is possible to show that there is a polynomial-time algorithm to compute the min-cut, by relying on results from submodular function optimization. Given a set $\mathcal{U}$ and a function f defined on all subsets of $\mathcal{U}$, consider the relationship between $f(A)+f(B)$, and $f(A\cap B)+f(A\cup B)$. If the two quantities are equal (such as when f is set cardinality), the function f is said to be modular. If $f(A)+f(B)\geq f(A\cap B)+f(A\cup B)$, f is said to be submodular. Submodular functions can be minimized in high-order polynomial time [91], and have recently found important applications in machine vision [64, 66] as well as areas such as spatial statistics [69].

There is a close relationship between submodular functions and graph cuts (see, e.g., [27]). Given a graph, let $\mathcal{U}$ be the nodes of the graph, and $f(A)$ be the sum of the weights of the edges from nodes in A to nodes not in A. Such an f is called a *cut function*, and the following argument shows it to be submodular. Each term in $f(A\cap B)+f(A\cup B)$ also appears in $f(A)+f(B)$. Now consider an edge that starts at a node that is in A, but not in B, and ends at a node in B, but not in A. Such an edge appears in $f(A)+f(B)$ but not in $f(A\cap B)+f(A\cup B)$. Thus, a cut function f is not modular. A very similar argument shows that the cost of a s-t cut is also a submodular function: $\mathcal{U}$ consists of the non-terminal nodes, and $f(A)$ is the weight of the outgoing edges from $A\cup\{s\}$, which is the cost of the cut.

While cut functions are submodular, the class of submodular functions is much more general. As we have pointed out, this suggests that special-purpose min-cut algorithms would be preferable to general-purpose submodular function minimization techniques [91] (which, in fact, are not yet practical for large problems). In fact, there are very efficient min-cut algorithms.

The key to computing min-cuts efficiently is the famous problem reduction of Ford and Fulkerson [37]. The reduction uses the maximum flow problem ("max-flow"), which is defined on the same directed graph, but the edge weights are now interpreted as capacity constraints. An s-t flow is an assignment of non-negative values to the edges in the graph, where the value of each edge is no more than its capacity. Furthermore, at every non-terminal vertex the sum of values assigned to incoming edges equals the sum of values assigned to outgoing edges.

Informally speaking, most max-flow algorithms repeatedly find a path between s and t with spare capacity, and then increase (augment) the flow along this path. For a particularly clear explanation of max-flow and its relationship to min-cut see [60]. From the standpoint of computer vision, max-flow algorithms are very fast, and can even be used for real-time systems (see, e.g., [65]). In fact, most of the graphs that arise in vision are largely grids, like the example shown in Figure 1. These graphs have many short paths from s to t, which in turn suggests the use of max-flow algorithms that are specially designed for vision problems, such as [16].

3.3 Example application: binary image restoration via graph cuts

There are a number of algorithms for solving low-level vision problems that compute a minimum s-t cut on an appropriately defined graph. This technique is usually called "graph cuts" (the term appears to originate in [19]).[4] The idea has been applied to a number of problems in computer vision, medical imaging and computer graphics (see [17] for a recent survey). We will describe the use of graph cuts for interactive object segmentation in section 5.3, and for stereo in section 7.1. However, their original application to vision was for binary image restoration [42].

Consider a binary image where each pixel has been independently corrupted (for example, the output of an error-prone document scanner). Our goal is to clean up the image. This can be naturally formulated as an optimization problem of the form defined by equation (2). The labels are binary, so the search space is $\{0,1\}^n$ where n is the number of pixels in the image, and there are 2^n possible solutions. The function D_i provides a cost for labeling the i-th pixel using one of two possible values; in practice, this cost would be zero for it to have the same label that was observed in the corrupted image, and a positive constant λ for it to have the opposite label. The simplest choice for $V_{i,j}(x_i, x_j)$ is a 0-1 valued function, which is 1 just in case pixels i and j are adjacent and $x_i \neq x_j$. Putting these together we see that $E(x_1 \dots, x_n)$ equals λ times the number of pixels that get assigned a value different from the one observed in the corrupted image, plus the number of adjacent pixels with different labels. By minimizing this energy we find a spatially coherent labeling that is similar to the observed data.

This energy function is often referred to as the Ising model (its generalization to more than 2 labels is called the Potts model). [42] showed that the problem of minimizing this energy can be reduced to the min-cut problem on an appropriately constructed graph.[5] The graph is shown in Figure 1, for a 9-pixel image. In this graph, the terminals s, t correspond to the labels 0 and

[4]Despite the similar names graph cuts are not closely related to normalized cuts.

[5]As [66] points out the actual construction dates back at least 40 years, but [42] first applied it to images.

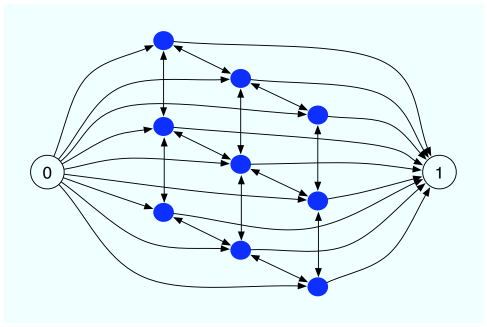

Figure 1: Cuts separating the terminals (labeled 0 and 1) in this graph correspond to binary labelings of a 3 by 3 image. The dark nodes are the pixel vertices. This graph assumes the usual 4-connected neighborhood system among pixels.

1. Besides the terminals, there is one vertex per pixel; each such pixel vertex is connected to the adjacent pixel vertices, and to each terminal vertex. A cut in this graph leaves each pixel vertex connected to exactly one terminal vertex. This naturally corresponds to a binary labeling. With the appropriate choice of edge weights, the cost of a cut is the energy of the corresponding labeling.[6] The weight of edges cut between pixel vertices will add to the number of adjacent pixels with different labels; the weight of edges cut between pixel vertices and terminals will sum to λ times the number of pixels with opposite labels to those observed in the input data.

It is important to realize that this construction depends on the specific form of the Ising model energy function. Since the terminals in the graph correspond to the labels, the construction is restricted to 2 labels. As mentioned above, the natural multi-terminal variants of the min-cut problem are NP-hard [28]. In addition, we cannot expect to be able to efficiently solve even simple generalizations of the Ising energy energy function, since minimizing the Potts energy function with 3 labels is NP-hard [20].

3.4 Bipartite matching algorithms

In the minimum weight bipartite matching problem we have a bipartite graph with $\mathcal{V} = A \cup B$ and weights w_e associated with each edge. The goal is to find a perfect matching M with minimum total

[6]There are two ways to encode the correspondence between cuts and labelings. If pixel p gets labeled 0, this can be encoded by cutting the edge between the vertex for p and the terminal vertex for 0, or to leaving this edge intact (and thus cutting the edge to the terminal vertex for 1). The different encodings lead to slightly different weights for the edges between terminals and pixels.

weight. Perfect matchings in bipartite graphs are particularly interesting because they represent one-to-one correspondences between elements in A and elements in B. In computer vision the problem of finding correspondences between two sets of elements has many applications ranging from three-dimensional reconstruction [30] to object recognition [9].

The bipartite matching problem described here is also known as the *assignment problem.* We also note that there is a version of the problem that involves finding a *maximum* weight matching in a bipartite graph, without requiring that the matching be perfect. That problem is essentially equivalent to the one described here (they can be reduced to each other) [68].

We can solve the bipartite matching problem in $O(n^3)$ time using the Hungarian algorithm [80], where $n = |A| = |B|$. That algorithm relies on linear programming duality and the relationship between matchings and vertex covers to sequentially solve a set of intermediate problems, eventually leading to a solution of the original matching problem.

One class of practical algorithm for bipartite matching rely on the following generalization of the maximum-flow problem. Let $\mathcal{G}$ be a directed graph with a capacity c_e and weight w_e associated with each edge $e \in \mathcal{E}$. Let s and t be two terminal nodes (the source and the sink) and let f be an s-t flow with value $f(e)$ on edge e. The cost of the flow is defined as $\sum_{e \in E}(f(e) * w_e)$. In the *minimum-cost flow problem* we fix a value for the flow and search for a flow of that value with minimum cost. This problem can be solved using an approach similar to the Ford-Fulkerson method for the classical max-flow problem, by sequentially finding augmenting paths in a weighted residual graph. When the capacities are integral this approach is guaranteed to find a minimum-cost flow with integer values in each edge.

The bipartite matching problem on a graph can be reduced to a minimum-cost flow problem. We simply direct the edges in $\mathcal{E}$ so they go from A to B and add two nodes s and t such that there is an edge of weight zero from s to every node in A and to t from every node in B. The capacities are all set to one. An integer flow of value n in the modified graph ($n = |A| = |B|$) corresponds to a perfect matching in the original graph, and the cost of the flow is the weight of the matching.

4 Dynamic programming

Dynamic programming [8] is a powerful general technique for developing efficient discrete optimization algorithms. In computer vision it has been used to solve a variety of problems including curve detection [3, 39, 77], contour completion [95], stereo matching [5, 79], and deformable object matching [4, 7, 25, 34, 32].

The basic idea of dynamic programming is to decompose a problem into a set of subproblems, such that (A) given a solution to the subproblems, we can quickly compute a solution to the original problem, and (B) the subproblems can be efficiently solved recursively in terms of each other. An important aspect of dynamic programming is that the solution of a single subproblem is often used multiple times, for solving several larger subproblems. Thus in a dynamic programming algorithm we need to “cache” solutions to avoid recomputing them later on. This is what makes dynamic programming algorithms different from classical recursive methods.

Similar to shortest paths algorithms, dynamic programming relies on an optimal substructure property. This makes it possible to solve a subproblem using solutions of smaller subproblems.

In practice we can think of a dynamic programming algorithm as a method for filling in a table, where each table entry corresponds to one of the subproblems we need to solve. The dynamic programming algorithm iterates over table entries and computes a value for the current entry using values of entries that it has already computed. Often there is a simple recursive equation which defines the value of an entry in terms of values of other entries, but sometimes determining a value involves a more complex computation.

4.1 Dynamic programming on a sequence

Suppose we have a sequence of n elements and we want to assign a label from $\mathcal{L}$ to each element in the sequence. In this case we have a search space $\mathcal{S} = \mathcal{L}^n$ where a candidate solution $(x_1, \ldots, x_n)$ assigns label x_i to the i'th element of the sequence. A natural energy function for sequence labeling can be defined as follows. Let $D_i(x_i)$ be a cost for assigning the label x_i to the i'th element of the sequence. Let $V(x_i, x_{i+1})$ be a cost for assigning two particular labels to a pair of neighboring elements. Now the cost of a candidate solution can be defined as

$$E(x_1, \ldots, x_n) = \sum_{i=1}^{n} D_i(x_i) + \sum_{i=1}^{n-1} V(x_i, x_{i+1}). \tag{3}$$

This objective function measures the cost of assigning a label to each element and penalizes “incompatible” assignments between neighboring elements. Usually D_i is used to capture a cost per element that depends on some input data, while V encodes a prior that is fixed over all inputs. Optimization problems of this form are common in a number of vision applications, including in some formulations of the stereo problem as discussed in section 7.2, and in visual tracking [29].

The MAP estimation problem for a general hidden Markov model [84] also leads to an optimization problem of this form.

Now we show how to use dynamic programming to quickly compute a solution $(x_1^*, \ldots, x_n^*)$ minimizing an energy function of the form in equation (3). The algorithm works filling in tables storing costs of optimal partial solutions of increasing size. Let $B_i[x_i]$ denote the cost of the best assignment of labels to the first i elements in the sequence with the constraint that the i'th element has the label x_i. These are the "subproblems" used by the dynamic programming procedure. We can fill in the tables B_i in order of increasing i using the recursive equations,

$$B_1[x_1] = D_1(x_1), \tag{4}$$

$$B_i[x_i] = D_i(x_i) + \min_{x_{i-1}} \left(B_{i-1}[x_{i-1}] + V(x_{i-1}, x_i)\right). \tag{5}$$

Once the B_i are computed, the optimal solution to the overall problem can be obtained by taking $x_n^* = \arg\min_{x_n} B_n[x_n]$ and tracing back in order of decreasing i,

$$x_i^* = \arg\min_{x_i} \left(B_i[x_i] + V(x_i, x_{i+1}^*)\right). \tag{6}$$

Another common approach for tracing back the optimal solution involves caching the optimal $(i-1)$'th label as a function of the i'th label in a separate table when computing B_i,

$$T_i[x_i] = \operatorname*{argmin}_{x_{i-1}} \left(B_{i-1}[x_{i-1}] + V(x_{i-1}, x_i)\right). \tag{7}$$

Then, after we compute x_n^* we can track back by taking $x_{i-1}^* = T_i[x_i^*]$ starting at $i = n$. This simplifies the tracing back procedure at the expense of a higher memory requirement.

The dynamic programming algorithm runs in $O(nk^2)$ time. The running time is dominated by the computation of the tables B_i. There are $O(n)$ tables to be filled in, each table has $O(k)$ entries and it takes $O(k)$ time to fill in one table entry using the recursive equation (5).

In many important cases the dynamic programming solution described here can be sped up using the distance transform methods in [33]. In particular in many situations (including the signal restoration application described below) $\mathcal{L}$ is a subset of $\mathbb{R}$ and the pairwise cost is of the form $V(x_i, x_j) = |x_i - x_j|^p$ or $V(x_i, x_j) = \min(|x_i - x_j|^p, \tau)$ for some $p > 1$ and $\tau > 0$. In this case the distance transform methods can be used to compute all entries in B_i (after B_{i-1} is computed) in $O(k)$ time. This leads to an overall algorithm that runs in $O(nk)$ time, which is optimal assuming the input is given by an arbitrary set of nk costs $D_i(x_i)$. In this case any algorithm would have to look at each of those costs, which takes at least $O(nk)$ time.

4.2 Relationship to shortest paths

Many dynamic programming algorithms can be viewed as solving a shortest path problem in a graph. Here we outline the construction for the dynamic programming method from above, for minimizing the energy in equation (3). Figure 2 illustrates the underlying graph. One could solve the optimization problem by building the graph described here and using a generic shortest paths algorithm. However, both theoretically and in practice it is often more efficient to implement the dynamic programming solution directly. While the dynamic programming method runs in $O(nk^2)$ time, using a generic algorithm such as Dijkstra's shortest paths we get an $O(nk^2 \log(nk))$ method. The problem is that Dijkstra's algorithm is more general than necessary since it can handle graphs with cycles. This illustrates the benefit of using the most specific algorithm possible to solve an optimization problem.

Let $\mathcal{G}$ be a directed graph with $nk + 2$ vertices $\mathcal{V} = \{(i, x_i) \mid 1 \leq i \leq n,\ x_i \in \mathcal{L}\} \cup \{s, t\}$. The case for $n = 5$ and $k = 4$ is shown in Figure 2. Except for the two distinguished vertices s and t, each vertex (i, x_i) in this graph corresponds an assignment of a label to an element of the sequence. We have edges from (i, x_i) to $(i+1, x_{i+1})$ with weight $D_{i+1}(x_{i+1}) + V(x_i, x_{i+1})$. We also have edges from s to $(1, x_1)$ with weight $D_1(x_1)$. Finally we have edges from (n, x_n) to t with weight zero. In this graph a path from s to t contains exactly one vertex per element in the sequence, which corresponds to assigning a label to each element. It is easy to show that the length of the path is exactly the cost of the corresponding assignment. More generally, the length of a shortest path from s to (i, x_i) equals the value $B_i[x_i]$ in equation (5). We conclude that a shortest path from s to t corresponds to a global minimum of the energy function E.

4.3 Extensions

The dynamic programming procedure described above can be generalized to broader class of energy functions. For example suppose each element in a sequence can get a label from a different set, i.e. $\mathcal{S} = \mathcal{L}_1 \times \cdots \times \mathcal{L}_n$. Let $D_i(x_i)$ be a cost for assigning the label $x_i \in \mathcal{L}_i$ to the i'th element. Let $V_i(x_i, x_{i+1})$ be a cost for assigning the labels $x_i \in \mathcal{L}_i$ and $x_{i+1} \in \mathcal{L}_{i+1}$ to the i'th and $i + 1$'th elements respectively. In this case there is a different pairwise cost for each pair of neighboring elements. We can define an energy function assigning a cost to a candidate solution $(x_1, \ldots, x_n)$ by taking the sum of the individual costs D_i and the pairwise costs V_i. A global minimum of this energy can be found using a modified version of the recursive equations shown above. Let $k_i = |\mathcal{L}_i|$.

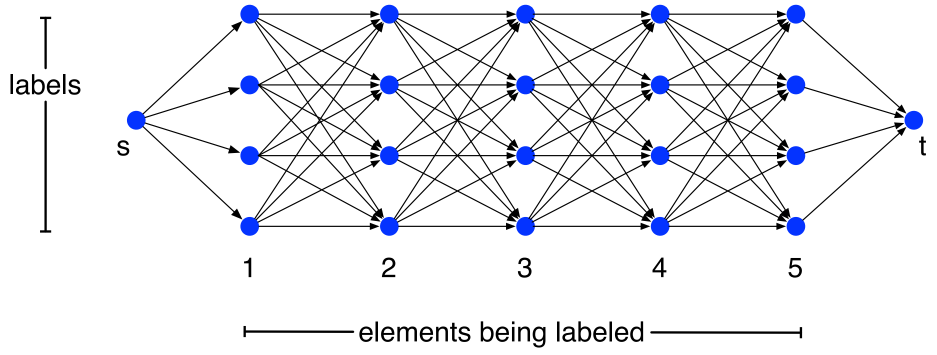

Figure 2: Paths in this graph correspond to assignments of labels to 5 elements in a sequence. Each of the columns represents an element. Each of the rows represents a label. A path from s to t goes through one vertex per column, which corresponds to an assignment of labels to elements. A shortest path from s to t corresponds to a labeling minimizing the energy in equation (3)

Now the table B_i has k_i entries and it takes $O(k_{i-1})$ time to fill in one entry in this table. The algorithm runs in $O(nk^2)$ time, where $k = \max k_i$.

Another class of energy functions that can be solved using a modified version of the method described here is one where the prior also includes an explicit cost, $H(x_i, x_{i+1}, x_{i+2})$, for assigning particular labels to three consecutive elements in a sequence.

Consider an energy function of the form

$$E(x_1, \ldots, x_n) = \sum_{i=1}^{n} D_i(x_i) + \sum_{i=1}^{n-1} V(x_i, x_{i+1}) + \sum_{i=1}^{n-2} H(x_i, x_{i+1}, x_{i+2}) \tag{8}$$

This type of energy has been used in active contour models [3], where D is defined by the image data, V captures a spring model for the contour and H captures a thin-plate spline model (see section 5.2). In this case the dynamic programming algorithm builds $n-1$ tables with k^2 entries each. The entry $B_i[x_{i-1}, x_i]$ denotes the cost of the best assignment of values to the first i elements in the sequence with the constraint that the $i-1$'th element has the label x_{i-1} and the i'th element has the label x_i. These entries can be defined using recursive equations,

$$B_2[x_1, x_2] \quad = \quad D_1(x_1) + D_2(x_2) + V(x_1, x_2), \tag{9}$$

$$B_i[x_{i-1}, x_i] \quad = \quad D_i(x_i) + V(x_{i-1}, x_i) + \min_{x_{i-2}} \left(B_{i-1}[x_{i-2}, x_{i-1}] + H(x_{i-2}, x_{i-1}, x_i)\right). \tag{10}$$

Now the B_i can be computed in order of increasing i as before. After the tables are filled in we can find a global minimum of the energy function by picking $(x^*_{n-1}, x^*_n) = \arg\min_{(x_{n-1}, x_n)} B_n[x_{n-1}, x_n]$

and tracing back in order of decreasing i,

$$x_i^* = \arg\min_{x_i} \left(B_{i+1}[x_i, x_{i+1}^*] + H(x_i, x_{i+1}^*, x_{i+2}^*) \right).$$

This method takes $O(nk^3)$ time. In general we can minimize energy functions that explicitly take into account the labels of m consecutive elements in $O(nk^m)$ time.

4.4 Example application: 1-D signal restoration via dynamic programming

As an example consider the problem of restoring a one-dimensional signal. We assume that a signal s is defined by a finite sequence of regularly spaced samples $s[i]$ for i from 1 to n. Figures 3(a) and 3(b) show a clean signal and a noisy version of that signal. The goal here to estimate the original signal $s[i]$ from its noisy version $n[i]$. In practice we look for a signal $s[i]$ which is similar to $n[i]$ but is smooth in some sense. The problem can be formulated using an objective function of the form in equation (3).

We can define the restored signal in terms of an assignment of labels to the sequence of sample points. Let $\mathcal{L}$ be a discretization of the domain of s (a finite subset of $\mathbb{R}$). Now take $D_i(x_i) = \lambda(x_i - n[i])^2$ to ensure that the value we assign to the i'th sample point is close to $n[i]$. Here λ is a non-negative constant controlling the relative weighting between the data and prior cost. The form of V depends on the type of smoothness constraint we want to impose on s. If we assume s is smooth everywhere we can take $V(x_i, x_{i+1}) = (x_i - x_{i+1})^2$. If we assume that s is piecewise-smooth we can take $V(x_i, x_{i+1}) = \min((x_i - x_{i+1})^2, \tau)$ where τ controls the maximum cost we assign to a "discontinuity" in the signal. This second choice of V is often called the weak string model in vision, following [11], who popularized this model and its solution via dynamic programming.

Figure 3 shows the result of restoring a noisy signal using these two choices for the pairwise costs. Note that with either choice for the pairwise cost we can use the distance transform methods mentioned above to find an optimal solution to the restoration problem in $O(nk)$ time. Here n is the number of samples in the signal and k is the number of discrete choices for the value of the signal in each sample point.

4.5 Dynamic programming on a tree

The dynamic programming techniques described above generalize to the situation where the set of elements we want to label are connected together in a tree structure. We will discuss an application of this technique for object recognition in section 6.3.

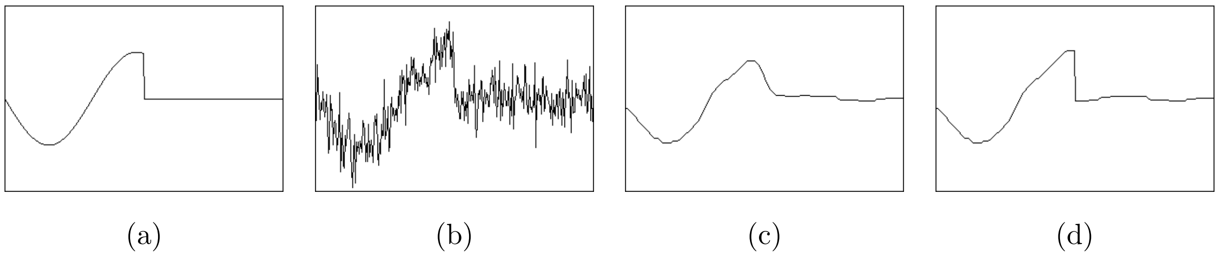

Figure 3: (a) A one-dimensional signal. (b) Noisy version of the signal. (c) Restoration using the prior $V(x_i, x_{i+1}) = (x_i - x_{i+1})^2$. (d) Restoration using the prior $V(x_i, x_{i+1}) = \min((x_i - x_{i+1})^2, \tau)$.

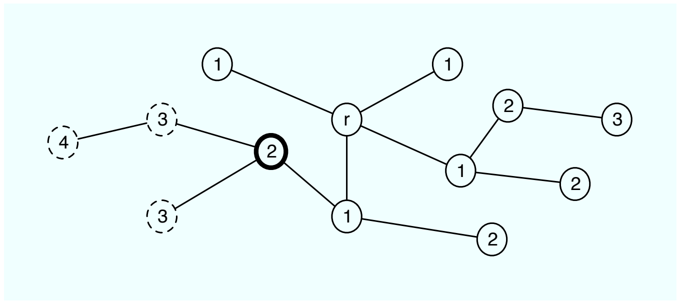

Figure 4: Dynamic programming on a tree. A tree where the root is labeled r and every other node is labeled with its depth. The dashed vertices are the descendents of the bold vertex. The best labels for the dashed verticies can be computed as a function of the label of the bold vertex.

Let $\mathcal{G}$ be tree where the vertices $\mathcal{V} = \{v_1 \ldots, v_n\}$ are the elements we want to label and the edges in $\mathcal{E}$ indicate which elements are directly related to each other. As before, let $D_i(x_i)$ be a cost for assigning label x_i to the i'th element. For each edge $\{v_i, v_j\} \in \mathcal{E}$ we let $V_{ij}(x_i, x_j) = V_{ji}(x_j, x_i)$ be a cost for assigning two particular labels x_i and x_j to the i'th and j'th element respectively. Now an objective function can be defined by analogy to the sequence case,

$$E(x_1, \ldots, x_n) = \sum_{i=1}^{n} D_i(x_i) + \sum_{\{v_i, v_j\} \in \mathcal{E}} V_{ij}(x_i, x_j). \tag{11}$$

An optimal solution can be found as follows. Let v_r be an arbitrary root vertex in the graph (the choice does not affect the solution). From this root, each vertex v_i has a depth d_i which is the number of edges between it and v_r (the depth of v_r is zero). The children, C_i, of vertex v_i are the vertices connected to v_i that have depth $d_i + 1$. Every vertex v_i other than the root has a unique parent, which is the neighboring vertex of depth $d_i - 1$. Figure 4 illustrates these concepts.

We define n tables B_i each with $k = |\mathcal{L}|$ entries such that $B_i[x_i]$ denotes the cost of the best labeling of v_i and its descendents assuming the label of v_i is x_i. These values can be defined using the recursive equation,

$$B_i[x_i] \quad = \quad D_i(x_i) + \sum_{v_j \in C_i} \min_{x_j} \left(B_j[x_j] + V_{ij}(x_i, x_j) \right). \tag{12}$$

For vertices with no children we have, $B_i[x_i] = D_i(x_i)$. The other tables can be computed in terms of each other in decreasing depth order. Note that $B_r[x_r]$ is the cost of the best labeling of the whole graph, assuming the label of the root is x_r.

After all tables are computed we can find a global minimum of the energy function by picking $x_r^* = \arg\min_{x_r} B_r[x_r]$ and tracing back in order of increasing depth,

$$x_i^* = \arg\min_{x_i} \left(B_i[x_i] + V_{ij}(x_i, x_j^*) \right),$$

where v_j is the parent of v_i. The overall algorithm runs in $O(nk^2)$ time. As in the case for sequences, distance transform techniques can often be used to obtain an $O(nk)$ time algorithm [34]. This speedup is particularly important for the pictorial structure matching problem discussed in section 6.3. In that problem k is on the order of the number of pixels in an image and a quadratic algorithm is not practical.

We note that dynamic programming can also be used to minimize energy functions defined over certain graphs with cycles [4, 32]. Although the resulting algorithms are less efficient they still run in polynomial time. This method is called non-serial dynamic programming. Its asymptotic running time depends exponentially on a measure of graph connectivity called treewidth; trees have treewidth 1, while a square grid with n nodes has treewidth $O(\sqrt{n})$. When the underlying graph is not a path, the problem can not usually be solved via shortest paths algorithms, because there is no way to encode the cost of a solution in terms of the length of a path in a reasonably sized graph. However, a generalization of Dijkstra's shortest paths algorithm to such cases was first proposed in [61] and more recently explored in [35].

4.6 Matching sequences via dynamic programming

Let $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{m-1})$ be two sequences of elements. Here we consider the problem of finding a set of correspondences between A and B that respects the natural ordering of the elements. This is an important problem that arises both in stereo matching [5, 79] (see section 7.2) and in matching deformable curves [7, 94] (see section 6.2).

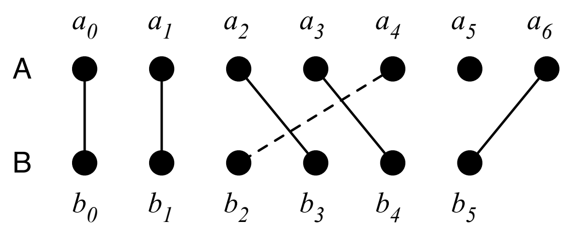

Figure 5: Matching two sequences, A and B, with the ordering constraint. Matchings satisfying the ordering constraint lead to pairings with no crossing. The solid lines show a valid matching. The dotted line shows an extra correspondance that would violate the ordering constraint.

Intuitively we want to consider correspondences between A and B that do not involve crossings, as shown in Figure 5. The precise definition is that if a_i is matched to b_j and $i' > i$ then $a_{i'}$ can only be matched to $b_{j'}$ such that $j' > j$. This is often called the *ordering constraint*.

Correspondences that satisfy the ordering constraint are equivalent to alignments defined by putting the two sequences on top of each other after inserting spaces in each one. This is exactly the sequence alignment approach commonly used in biology for comparing DNA sequences. It is also related to dynamic time warping methods used in speech recognition to compare one-dimensional signals. When a person says a word in two different speeds there is a correspondence between the two sounds that satisfies the ordering constraint. In vision time warping methods have been used to recognize activities such as hand gestures [81].

Here we assume that there is a cost $C(a_i, b_j)$ for matching a_i to b_j. There is also a cost ϵ for leaving an element of A or B unmatched. Our goal is to find a valid set of correspondences that minimizes the total sum of costs. This can be done by finding a shortest path in a graph.

Let $\mathcal{G}$ be a directed graph where the nodes in $\mathcal{V}$ are the points in an $n+1$ by $m+1$ grid. There are two types of edges in $\mathcal{E}$. Diagonal edges connect nodes (i, j) to nodes $(i+1, j+1)$. These edges have weight $C(a_i, b_j)$. Horizontal and vertical edges connect nodes (i, j) to nodes $(i+1, j)$ and $(i, j+1)$. These edges have weight ϵ. Paths from $(0, 0)$ to (n, m) in $\mathcal{G}$ correspond to valid matchings between A and B in a simple way: if a path follows a diagonal edge out of (i, j) we match a_i with b_j. If a path takes a horizontal edge out of (i, j) we leave a_i unmatched, while a vertical step out of (i, j) corresponds to leaving b_j unmatched. Its not hard to show that the length of a path is equal to the cost of the corresponding matching. Since the graph is acylic we can compute shortest paths from $(0, 0)$ to (n, m) in $O(nm)$ time using dynamic programming. The graph described here does not have to be constructed explicitly. In practice we define a table of values $D[i,j]$ to represent the distance from $(0,0)$ to node (i,j). We can fill in the entries in the table in order of increasing i and j value using the recursive equation,

$$D[i,j] = \min(D[i-1,j-1] + C(a_{i-1}, b_{j-1}), D[i-1,j] + \epsilon, D[i,j-1] + \epsilon).$$

After D is filled in we can find the optimal set of correspondences between A and B by implicitly tracing the shortest path from $(0,0)$ to (n,m).

5 Discrete optimization algorithms for interactive segmentation

Interactive segmentation is an important problem in computer vision where discrete optimization techniques have had a significant impact. The goal is to allow a user to quickly separate a particular object from the rest of the image. This is done via an iterative process, where the user has the opportunity to correct or improve the current segmentation as the algorithm progresses. We will describe three different ways in which discrete optimization techniques can be used for interactive segmentation: intelligent scissors [78], which relies on shortest paths; active contour models [3, 56], which use dynamic programming; and GrabCut [87], which uses graph cuts.

Segmentation, of course, is a major topic in computer vision, and there are a number of common issues that segmentation algorithms face. For example, it is very difficult to evaluate a segmentation algorithm.[7] One of the advantages of interactive segmentation, which makes it more tractable, is that it involves a human user, rather than being fully automatic.

5.1 Intelligent scissors (shortest paths)

One way to perform interactive segmentation is for the user to select a few points on the boundary of the target object while an algorithm completes the gaps between selected points. However, a user may not know in advance how many points they need to select before the algorithm can complete the gaps accurately. With intelligent scissors a user starts by selecting a single point on the object boundary and then moves the mouse around. As the mouse pointer moves, a curve is drawn from the initial boundary point to the current mouse location; hopefully the curve will lie along the object boundary. [78] pointed out that this problem can solved via shortest paths in a graph. The graph in question has one vertex per pixel, and the edges connect a pixel to the 8 nearby pixels.

[7] Note that two recent papers [65, 75] provided hand-labeled segmentations for hundreds of images.

Let $\mathcal{G}$ be a graph where the vertices $\mathcal{V}$ are the pixels in the image, and the edges $\mathcal{E}$ connect each pixel to its 8 closest neighbors. The whole trick to intelligent scissors is to make the weight of an edge small when it lies along an object boundary. There are a number of ways in which this can be accomplished, but the basic idea of [78] is to use features that detect intensity edges, since these tend to occur on the boundary. For example, we can let the weight of an edge connecting neighboring pixels be near zero if the image gradient is high along this edge, while the weight is high if the image gradient is low. This encourages shortest paths in $\mathcal{G}$ to go along areas of the image with high gradient magnitude. It also encourages the path to be straight.

Given an initial pixel p we can solve a single-source shortest paths problem in $\mathcal{G}$ to get an optimal curve from p to every other image pixel. As the user moves the mouse around we draw the optimal curve from p to the current mouse position. At any point in time the user can click the mouse button to select the current curve. The process then restarts with a new initial point defined by the current mouse location. In practice this process allows a user to select accurate boundaries with very little interaction. Further ideas described in [78] include on-the-fly training of the specific type of boundary being traced, and a mechanism for segmentation without mouse clicks.

Figure 6 shows an example of boundary tracing using the approach, created using the livewire plugin for the ImageJ toolbox.

5.2 Active contour models (dynamic programming)

Active contour models [56] are a popular tool for interactive segmentation, specially in medical image analysis. In this setting the user specifies an initial contour close to a target object. The active contour then seeks a nearby boundary that is consistent with the local image features (which typically means that it lies along intensity edges), and which is also spatially smooth.

While the original paper [56] viewed active contours as continuous curves, there is an advantage to taking a discrete approach, since the problem can then be solved via dynamic programming [3]. Under this view, a contour is represented by a sequence of n control points that define a closed or open curve (via a polygon, or some type of spline). A candidate solution specifies a position for each control point, from a set of k possible positions. We can write such a candidate solution as $(x_1, \ldots, x_n)$ and there are k^n candidate solutions.

There is a simple energy function we can define to formalize the problem. A per-element cost $D_i(x_i)$ indicates how good a particular position is for the i'th control point. Typically D_i is low

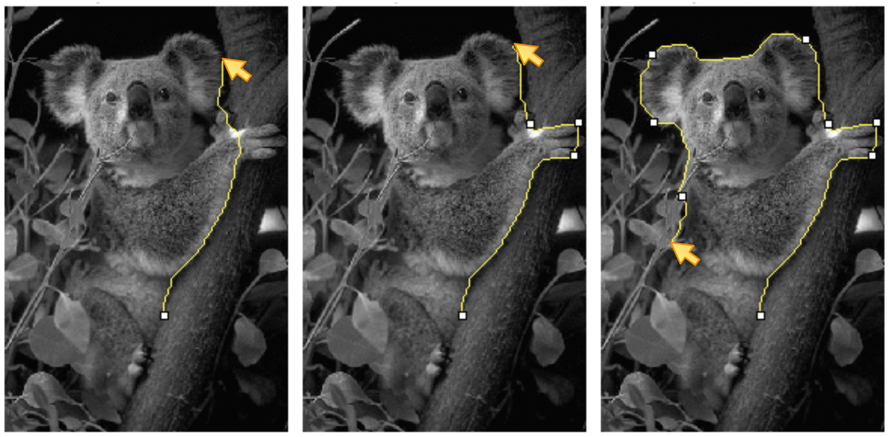

Figure 6: Tracing a boundary using intelligent scissors. Selected points (mouse clicks) are marked by white squares, while the current mouse position is marked by the arrow. The image on the left shows the shortest path from the first selected point to a point on the ear. Selecting this path would cut through the paw of the Koala. The image on the center shows how the paw can be selected with a few more clicks. The last image shows the result after more interaction.

where the image gradient is high, so the contour is attracted to intensity boundaries. A pairwise cost V encodes a spring-like prior, which is usually of the form $V(x_i, x_{i+1}) = \alpha||x_i - x_{i+1}||^2$. This term encourages the contour to be short.[8] Finally a cost H encodes a "thin plate spline" prior, $H(x_i, x_{i+1}, x_{i+2}) = \beta||x_i - 2 * x_{i+1} + x_{i+2}||^2$. This term penalizes curves with high curvature.

Both V and H can be understood in terms of finite difference approximations to derivative operators. Note that we are assuming for the moment that we have an open curve. The energy function is of the form given in equation (8), and thus can be solved in $O(nk^3)$ time with dynamic programming.

Active contour models provide a good example of the tradeoffs between local and global minimization. At one extreme, suppose that the set of possible positions for each control point was very large (for example, each control point could be anywhere on the image). In this case the solution is independent of the initial placement of the contour. However, because k is so large the cost of computing an optimal solution is high. In practice, we usually consider positions for each

[8]The pairwise cost can also take into account the image data underneath the line segment connecting two control points. This would encourage the entire polygonal boundary to be on top of certain image features.

control point that are very near their initial position. The process is iterated several times, with the previous solution defining the search neighborhood for the next iteration. This is effectively a local improvement method with a neighborhood system where each solution has k^n neighbors. As we increase k, we find the optimal solution within a larger neighborhood, but at a greater computational cost.

A similar tradeoff occurs once we consider closed curves. In this case there should be terms $V(x_n, x_1)$, $H(x_{n-1}, x_n, x_1)$ and $H(x_n, x_1, x_2)$ in the energy function. Naively, this would prohibit the use of dynamic programming because the terms introduce cyclic dependencies in the energy function. However, we can fix the position of x_1 and x_2 and find the optimal position for the other control points using dynamic programming, and then minimize this over all k^2 possible positions of x_1 and x_2. This would compute the optimal solution for a closed curve in $O(nk^5)$ time. In practice this is often too expensive, and another option is to pick two control points that are next to each other and fix them in their current position. In $O(nk^3)$ time, this produces a curve that is optimal over a smaller set of candidates, where all control points except two are allowed to move to any of their k nearby positions. Thus, by reducing the neighborhood system among candidate solutions, we can obtain a more efficient algorithm.

In practice, these optimization steps are done repeatedly until convergence. The set of possible positions, k, is relatively small for a given iteration, but multiple iterations are performed until convergence. If we handle closed curves by freezing the position of two control points at each iteration, it is natural to change the chosen points between iterations. Figure 7 illustrates two iterations of this process.

5.3 GrabCut (graph cuts)

In section 3.3 we pointed out that the global minimum of the Ising model energy function can be efficiently computed [42]. This is a very natural energy function for capturing spatially coherent binary labelings, of which binary image restoration is a special case. Another compelling application of this result is the interactive segmentation technique introduced by [15] and further refined in an application called GrabCut [87].

The graph cuts construction for spatially coherent binary labelings is easy to apply to an image; essentially all that is required is the data term $D_i(x_i)$ for the two possible labels of each pixel. For many segmentation tasks, the natural labels are foreground and background. In [15], the user

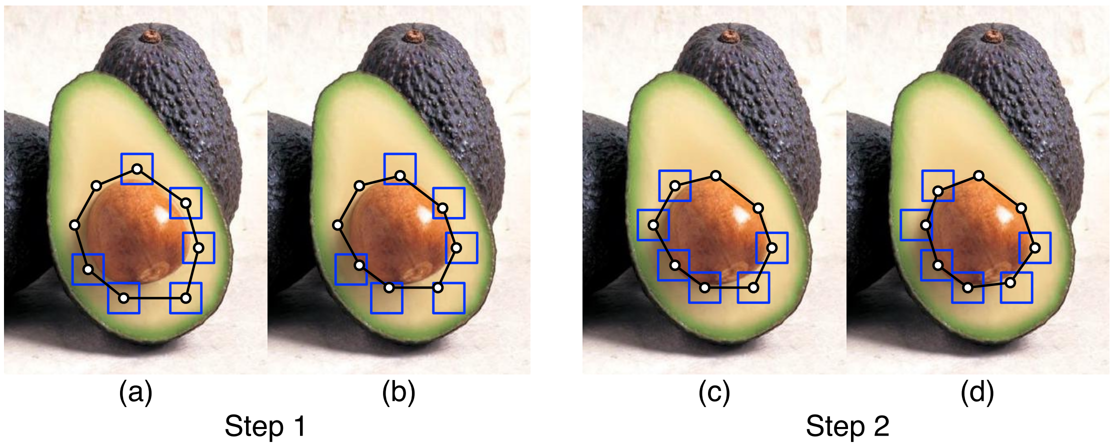

Figure 7: Segmentation with an active contour model (snake). In each step of the optimization two control points are fixed, and we optimize the locations of the other control points, while constraining them to be near their previous position. Images (a) and (c) show the current position of the control points and the search neighborhoods for the non-fixed control points in blue boxes. Images (b) and (d) show the result of finding a global optimum of the energy within the allowable positions for the control points. The runtime of a step depends on the size of the search neighborhoods.

marks certain pixels as foreground or background. To ensure that these pixels are given the correct labels, the algorithm makes the relevant D_i costs large enough to guarantee that the incorrect label is never chosen. For other pixels, the data term is computed based on the marked pixels. There are a number of ways that this can be accomplished, but typically a simple model of foreground and background intensity is used. For example, intensity histograms can be computed from marked foreground and background pixels, and the cost $D_i(x_i)$ reflects how popular the intensity at pixel i is among pixels marked as foreground and background. It is also reasonable to take into account the distance from the nearest user-marked pixel.

The algorithm of [15] can be run interactively, which is important for many applications. Once the user has marked a set of pixels, an initial segmentation is computed. The user can then refine this segmentation by marking additional pixels as foreground or background, which in turn updates the costs D_i. Since the graph changes little from iteration to iteration, [15] describes a way to re-use much of the underlying max-flow computation (a more general technique to accomplish this was provided by [63]). An recent overview of the use of graph cuts for segmentation, that discusses these issues and quite a few others, is given in [14].

The GrabCut application [87] adds a number of refinements to the basic interactive segmentation scheme of [15]. GrabCut iteratively alternates between estimating intensity models for the foreground and background, and computing a segmentation. This EM-style use of graph cuts has proven to be very powerful, and has been exploited in a number of applications [57, 58, 72] since it was first used [10]. GrabCuts also supports drawing a rough boundary instead of the user marking points inside and outside of the object, and uses sophisticated techniques for border matting.

6 Discrete optimization algorithms for object recognition

Modern object recognition algorithms often rely on discrete optimization methods to search over correspondences between two sets of features, such as features extracted from an image and features extracted from a model. Sections 6.1 and 6.2 describe two examples of this approach. Another class of recognition algorithms use discrete optimization methods to implicitly search over a very large set of possible object configurations in an image. An example of this approach is discussed in section 6.3.

6.1 Recognition using shape contexts

[9] described a method for comparing edge maps (or binary images) which has been used in a variety of applications. The approach is based on a three stage process: (1) a set of correspondences are found between the points in two edge maps, (2) the correspondences are used to estimate a non-linear transformation aligning the edge maps, and (3) a similarity measure between the two edge maps is computed which takes into account both the similarity between corresponding points and the amount of deformation introduced by the aligning transformation.

Let A and B be two edge maps. In the first stage of the process we want to find a mapping $\pi : A \to B$ putting each point in A in correspondence to its "best" matching point in B. For each pair of points $p_i \in A$ and $p_j \in B$ a cost $C(p_i, p_j)$ for mapping p_i to p_j is defined which takes into account the geometric distribution of edge points around p_i and p_j. This cost is based on a local descriptor called the *shape context* of a point. The descriptor is carefully constructed so that it is invariant to translations and fairly insensitive to small deformations. Figure 8 illustrates two points on different shapes with similar shape contexts.

Given the set of matching costs between pairs of points in A and B we can look for a map π minimizing the total cost, $H(\pi) = \sum_{p_i \in A} C(p_i, \pi(p_i))$, subject to the constraint that π is one-to-one

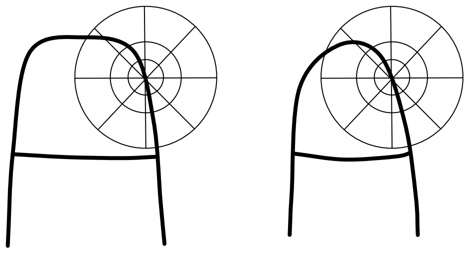

Figure 8: The shape context at a point p is a histogram of the positions of the remaining points relative to the position of p. Using histogram bins that are uniform over log-polar space we obtain a reasonable amount of invariance to deformations. Here we show two points on different shapes with similar descriptors. In this case the cost of matching the two points would be small.

and onto. This is precisely the weighted bipartite matching problem from section 3.4. To solve the problem we build a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that $\mathcal{V} = A \cup B$ and there is an edge between each vertex p_i in A to each vertex p_j in B with weight $C(p_i, p_j)$. Perfect matchings in this graph correspond to valid maps π and the weight of the matching is exactly the total cost of the map.

To handle outliers and to find correspondences between edge maps with different numbers of points we can add "dummy" vertices to each side of the bipartite graph to obtain a new set of vertices $\mathcal{V}' = A' \cup B'$. We connect a dummy vertex in one side of the graph to each non-dummy vertex in the other side using edges with a fixed positive weight, while dummy vertices are connected to each other by edges of weight zero. Whenever a point in one edge map has no good correspondence in the other edge map it will be matched to a dummy vertex. This is interpreted as leaving the point unmatched. The number of dummy vertices in each side is chosen so that $|A'| = |B'|$. This ensures that the graph has a perfect matching. In the case where A and B have different sizes some vertices in the larger set will always be matched to dummy vertices.

6.2 Elastic curve matching

Now consider the problem of finding a set of correspondences between two curves. This is a problem that arises naturally when we want to measure the similarity between two shapes. Here we describe a particularly simple formulation of this problem. Similar methods have been used in [7, 73, 94].

Let $A = (a_0, \ldots, a_{n-1})$ and $B = (b_0, \ldots, b_{m-1})$ be two sequences of points along two open curves. It is natural to look for a set of correspondences between A and B that respect the natural ordering of the sample points. Figure 9 shows an example. This is exactly the sequence matching

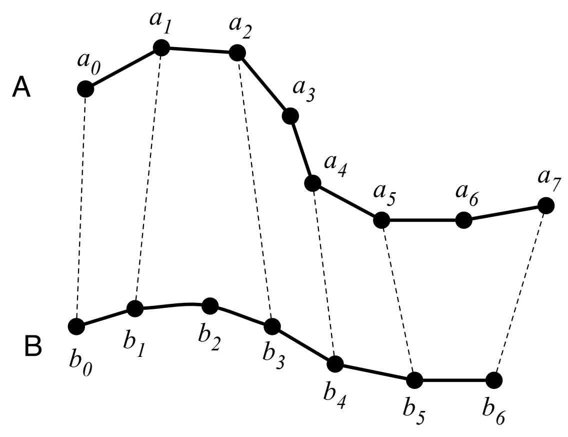

Figure 9: Matching two curves. We can think of the goal of matching as bending and stretching the curves to make them identical. The cost of bending is captured by the matching cost between two points. An unmatched point in one curve corresponds to a local stretch in the other. In this case a_3, a_6 and b_2 are left unmatched.

problem described in section 4.6.

Different curve matching methods can be defined by choosing how to measure the cost of matching two points on different curves, $C(a_i, b_j)$. There should also be a cost ϵ for leaving a point in A or B unmatched. The simplest approach for defining $C(a_i, b_j)$ is to measure the difference in the curvature of A at a_i and B at b_j. In this case the cost $C(a_i, b_j)$ will be low if the two curves look similar in the neighborhood of a_i and b_j.

When $C(a_i, b_j)$ measures difference in curvature, the minimum cost of a matching between A and B has an intuitive interpretation. It measures the amount of bending and stretching necessary to turn one curve into the other. The costs $C(a_i, b_j)$ measure amount the bending while the "gap-costs" ϵ measure the amount of stretching.

In the case of closed curves the order among points in each curve is only defined up to cyclic permutations. There are at least two different approaches for handling the matching problem in this case. The most commonly used approach is to independently solve the problem for every cyclic shift of one of the curves. This leads to an $O(mnk)$ algorithm, where $k = \min(m, n)$. An elegant and much more efficient algorithm is described in [74] which solves all of these problems simultaneously in $O(mn \log k)$ time. That algorithm is based on a combination of dynamic programming and divide and conquer.

6.3 Pictorial structures

Pictorial structures [36, 34] describe objects in terms of a small number of parts arranged in a deformable configuration. A pictorial structure model can be represented by an undirected graph $\mathcal{G}$ where the vertices correspond to the object parts and the edges represent geometric relationships between pairs of parts. An instance of the object is given by a configuration of its parts $x = (x_1, \ldots, x_n)$ where $x_i \in \mathcal{L}$ specifies the location of the i'th part. Here $\mathcal{L}$ might be the set of image pixels, or a more complex parameterization. For example, in estimating the configuration of a human body, $x_i \in \mathcal{L}$ could specify a position, orientation and amount of foreshortening for a limb. Let $D_i(x_i)$ be a cost for placing the i'th part at location x_i in an image. The form of D_i depends on the particular kinds of objects being modeled (though typically it measures the change in appearance); we will simply assume that it can be computed in a small amount of time. For a pair of connected parts let $V_{ij}(x_i, x_j)$ measure the deformation of a virtual spring between parts i and j when they are placed at x_i and x_j respectively.

The matching problem for pictorial structures involves moving a model around an image, looking for a configuration where each part looks like the image patch below it, and the springs are not too stretched. This is captured by an energy function of the form in equation (2). A configuration with low energy indicates a good hypothesis for the object location.

Let n be the number of parts in the model and k be the number of locations in $\mathcal{L}$. [34] showed how to solve the optimization problem for pictorial structures in $O(nk)$ time when the set of connections between parts forms a tree and the deformation costs, V_{ij}, are of a particular form. Note how this running time is optimal since it takes $O(nk)$ to evaluate $D_i(x_i)$ for each part at each location. This means that we can find the best coherent configuration for the object in the same (asymptotic) time it would take to find the best location for each part individually.

For tree models the energy function for pictorial structures has exactly the same the form as the energy in equation (11). The algorithm from [34] works by speeding up the dynamic programming solution outlined in section 4.5 using generalized distance transforms.

Let f be a function from locations in a grid to $\mathbb{R}$. The quadratic distance transform, $\mathcal{D}_f$, of f is another function from grid locations to $\mathbb{R}$, $\mathcal{D}_f(x) = \min_y(||x-y||^2 + f(y))$. If f and $\mathcal{D}_f$ are defined in a regular grid with k locations then D_f can be computed in $O(k)$ time using computational geometry techniques [33].

Consider the case where $\mathcal{L}$ corresponds to image pixels and for each pair of connected parts v_i

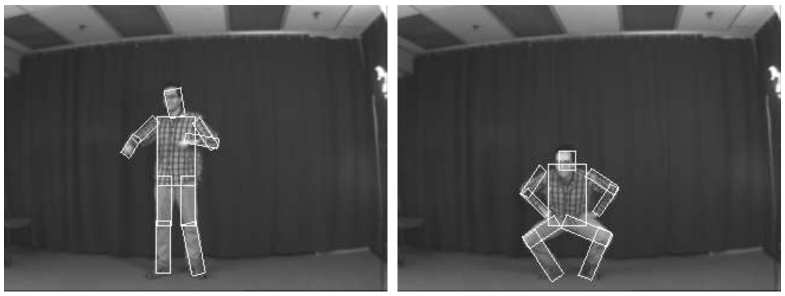
Figure 10: Matching a human body model to two images using the pictorial structures formulation. The model has 10 parts connected together in a tree-structure. The connections correspond to some of the main joints of the human body. A matching assigns a location for each part. In this case the location of a part specifies its position, orientantion and foreshortening.

and v_j there is an ideal relative displacement l_{ij} between them. That is, ideally $x_j \approx x_i + l_{ij}$. We can define a deformation cost $V_{ij}(x_i, x_j) = ||(x_i + l_{ij}) - x_j||^2$. We can think of the tables B_i used in the dynamic programming algorithm from section 4.5 as functions from image pixels to costs. For the special type of deformation costs defined here, equation (12) can be expressed as,

$$B_i(x_i) = D_i(x_i) + \sum_{v_j \in C_i} \mathcal{D}_{B_j}(x_i + l_{ij}).$$

In this form B_i can be computed in amortized $O(k)$ time once $\mathcal{D}_{B_j}$ are computed.

Figure 10 shows the result of matching a model of the human body to two images, using a model from [34]. In this case the model has 10 body parts that are connected together in a tree-structure corresponding to some of the main joints in a human body. A label for a part specifies a 2D location, scale, orientation and foreshortening.

7 Discrete optimization algorithms for stereo

As described in section 2.1, energy functions of the form given in equation (2) have been applied to pixel labeling problems for many years. Stereo is perhaps the vision problem where discrete optimization methods have had their largest impact; historically, it was one of the earliest examples where dynamic programming was applied [5, 79], while many top-performing methods rely on graph cuts [92]. Most of the issues that we consider here arise in a wide range of pixel labeling problems in early vision. This is particularly true of the graph cuts techniques we discuss, which have been

Name	$V_{i,j}(x_i, x_j)$	Bounded?	Metric?	Algorithms
L_2 distance	$(x_i - x_j)^2$	N	N	[52]
L_1 distance	$\|x_i - x_j\|$	N	Y	[51, 52, 90]
Truncated L_1	$\min(\|x_i - x_j\|, K)$	Y	Y	[20, 43, 59]
Potts model	$T[x_i \neq x_j]$	Y	Y	[20, 59]

Figure 11: Popular choices of V and their properties, along with a few important algorithms. Except for the Potts model, we assume that the labels x_i, x_j are scalars. For the first two choice of V, the global minimum can be efficiently computed via graph cuts [51, 52, 90], under some reasonable assumptions about the label set. The last two problems are NP-hard [20].

applied to many problems outside of stereo, as well as to problems in related fields such as computer graphics (see [17] for a survey).

The stereo problem is easily formulated as a pixel labeling problem of the form in equation (2), where the labels correspond to disparities. There is usually a simple way to measure the local evidence for a label in terms of intensity differences between pixels in different images, while a prior is used to aggregate evidence from nearby pixels.

In stereo, and pixel labeling in general, it is important to have a prior that does not oversmooth. Both the complexity of the energy minimization problem and the accuracy of the solutions turn out to depend heavily on the choice of prior. Here we assume that $V_{ij} = V$ for a pair of neighboring pixels and zero otherwise. If V never exceeds some value, we will refer to it as bounded. Such choices of V are sometime referred to a re-descending, robust or discontinuity-preserving (the first two terms come from robust statistics [45]). If V is not bounded, then adjacent pixels will be forbidden to have dramatically different labels. Such a V is believed to oversmooth boundaries, which accounts for the generally poor performance of such methods on the standard stereo benchmarks [92] (also see [100, Figure 3.10] for a good example of this oversmoothing effect). Some popular choices of V are summarized in Figure 11.

If we assume V is a metric, then the energy minimization problem is called the metric labeling problem, which was introduced by [59] and has been studied in the algorithms community. Constant factor approximation algorithms were given by [20, 59] for the Potts model, and by [43] for the truncated L_1 distance. The approximation algorithms of [59] are based on linear programming, and do not appear to be practical for realistic sized images, while the approximation algorithms of

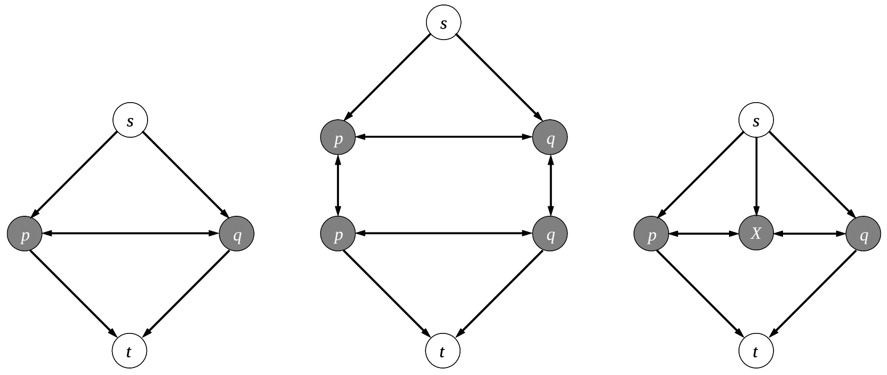

Figure 12: The basic graph cut constructions, for two pixels. (A) The graph used in section 3.3, and by [64]. (B) The graph used by [51], for 3 labels. (C) The graph used in the expansion move algorithm of [20].

[20, 43] rely on graph cuts.

7.1 Graph cut algorithms for pixel labeling problems

The construction that was used to reduce binary labeling problems to min-cuts, which we described in section 3.3, uses edges between pixels and terminals to encode labelings. For a pixel p, there is an edge between p and each terminal, and the choice of which of these edges to cut determines the label of p. There are two known methods that generalize this construction to handle more labels without introducing more terminals. One approach is to build a graph with additional edges per pixel. The other is to keep the graph more or less the same, but to change the meaning of the two labels represented by the terminals.

If we consider adding more edges per pixel, the simplest solution is to construct a chain of $k-1$ vertices per pixel, which gives rise to k possible edges that may be cut, one per label. Such an encoding is the basis of the graph cut algorithms of [50, 51, 52, 89, 90]. The most general version of these algorithms is due to [52], who showed how to find the global minimum for an arbitrary choice of D_i as long as V is a convex function of the difference between labels. The special case where V is the L_1 distance was originally solved by [51], relying on a construction very similar to that proposed by [90]; this construction is shown for 3 labels in Figure 12(B).

The main limitation of this approach is that it is restricted to convex V, which typically leads to oversmoothing. In addition, the graph has nk nodes, which makes its memory requirements problematic for applications with a large number of labels.

An alternative approach is to apply graph cuts iteratively, to define local improvement algorithms where the set of candidate solutions considered in each step is very large. These methods are defined by the notion of a "move" from an initial labeling, where every pixel can either keep its current label or adopt a new label. Such "move-making" techniques are related to the active contour technique described in section 5.2 and to the classical line-search subroutine used in continuous optimization [83], as well as to the idea from [2] of a local minimum within a large neighborhood.

The primary difference among these algorithms concerns how a move is defined. The best-known algorithm performs expansion moves [20], where in each step a pixel can either keep its current label or adopt another label α, and α changes at each iteration. Other interesting algorithms include swap moves [20] and jump moves [100].

Consider minimizing an energy function of the form given in equation (2), where we have a current labeling x, and we seek the move with the lowest energy. Every pixel is faced with a binary decision, either to keep its old label under x or to adopt a new label, so a move can be viewed as a binary image $(b_1, \ldots, b_n)$. The energy of a move starting at x, which is the function we seek to minimize, can be written as

$$E_x(b_1, \ldots, b_n) \quad = \quad \sum_i B_i(b_i) \quad + \quad \sum_{i,j} B_{i,j}(b_i, b_j). \tag{13}$$

Here, B_i comes from D in the original energy E, while $B_{i,j}$ comes from V. The notation E_x emphasizes the dependence of the energy function on the initial labeling x.

The problem of minimizing energy functions of this form has been studied for a number of years in the Operations Research community, principally by P. Hammer and his colleagues (see [12] for a recent survey). For arbitrary $B_{i,j}$ the problem is NP-hard; however, if

$$B_{i,j}(0,1) + B_{i,j}(1,0) \quad \geq \quad B_{i,j}(0,0) + B_{i,j}(1,1) \tag{14}$$

it can be minimized via graph cuts [12, 64]. This inequality is called regularity by [64], and is a special case of submodularity.[9] There is no restriction on B_i. For the expansion move, if V is a metric then equation (14) holds. [64] gives a general-purpose graph construction for minimizing E_x, assuming regularity. The graph has the same structure shown in Figures 1 and 12(A).

For expansion moves there is a different graph construction to minimize E_x that was originally given [20], which is shown in Figure 12(C). Recall that in an expansion move the binary label 0

[9]Consider subsets of $\{a, b\}$. Submodularity says that $f(\{a\}) + f(\{b\}) \geq f(\emptyset) + f(\{a, b\})$, which is regularity if we represent the sets with indicator vectors.

represents a pixel keeping its current label in x, while the binary label 1 represents adapting the new label α. These two choices are asymmetric; for example, two neighboring pixels that get the binary label 1 will end up with the same label α, and hence no cost from V, while two pixels that get the binary label 0 can have a cost from V if they have different labels in x. This asymmetry is reflected in the graph construction, which includes a new node plus additional edges that impose the correct cost. This construction is slightly less efficient than applying the general-purpose technique of [64], due to the larger size of the graph.

The expansion move algorithm has proved to be perhaps the most effective graph cuts technique. Some of the most compelling evidence in favor of this method comes from a recent experimental comparison of discrete optimization methods [97], which found that this method had the best overall performance on a range of benchmarks. On their benchmarks, the expansion move algorithm obtained solutions whose energy is within 0.018% to 3.6% of the global minimum. The algorithm provides some interesting theoretical properties [20]; for example, if V is the Potts model, the algorithm produces a 2-approximation, while [67] gave a generalization of the expansion move algorithm that also provides per-instance lower bounds.

Given the success of the expansion move algorithm, there has been considerable interest in applying it broadly. The paper that proposed the expansion move algorithm [20] proved that it could be applied when V is a metric.[10] As [64] showed, we can find the best expansion move if E_x is regular, which is equivalent to

$$V(x_i,\alpha)+V(\alpha,x_j)\geq V(x_i,x_j)+V(\alpha,\alpha). \tag{15}$$

Equation (15) defines a Monge property, which has been studied for years in the theoretical computer science community, and is known to be closely related to submodularity (see [22] for a review). It is also a generalization of a metric; if we assume $V(\alpha,\alpha)=0$, then equation (15) is just the triangle inequality.

The largest class of energy functions where the optimal expansion move can be computed by graph cuts is provided by [88] and by [66]. [88] showed how to incorporate certain hard constraints (i.e., terms with infinite penalties) even though such a V violates equation (15). [66] points out that a relaxation due to [44, 12] can compute the optimal expansion move for another class of V's that violate equation (15). However, it is highly unlikely that the optimal expansion move can be

[10]If V is a metric, their choice of edge weights ensures that only one of the three edges connected to the new node in Figure 12(C) will be cut in the min-cut.

computed efficiently for an arbitrary choice of V, due to an NP-hardness result provided by [64].

It is possible to use the expansion move algorithm for an arbitrary choice of V, at the price of no longer guaranteeing that the expansion move computed is optimal, using the methods of [88] or [66]. In particular, it is possible to compute an expansion move that does not increase the energy. According to [88], their method is most likely to succeed on choices of V where few terms violate equation (15). This occurs (for instance) in some important graphics applications [1, 70] as well as a range of recent benchmarks [97].

7.2 Dynamic programming

Suppose we consider the the stereo correspondence problem one scanline at a time (i.e., we view it as a set of independent problems, where each problem is to find the disparities for the pixels along a given scanline). In this case, the pixel labeling problem we obtain for each scanline is equivalent to the sequence labeling problem from section 4.1, and it can be solved via dynamic programming. Classical algorithms for stereo using dynamic programming ([5, 79]) often impose an additional ordering constraint and solve the correspondence problem using the sequence matching approach discussed in section 4.6. This additional constraint implies that certain scene configurations, such as the "double nail", do not occur. This is often true in practice, but not always.

The assumption that the stereo problem can be solved independently for each scanline is appealing from a computational point of view. However, this assumption usually causes dynamic programming algorithms for stereo to give poor experimental results, since the inconsistency between adjacent scanlines produces a characteristic streaking artifact. This is clearly visible on the Middlebury stereo benchmarks [92].

Obviously, it is desirable to have a prior that imposes spatial coherence horizontally as well as vertically. There have been a number of attempts to enforce inter-scanline consistency while still making use of dynamic programming. A particularly interesting example is due to [99], who applied the tree-based dynamic programming technique described in section 4.5.

8 Recent developments

Since optimization is an important area of emphasis within computer vision, the pace of new developments is rapid. Most ongoing work can be broadly characterized as either (a) increasing the set of energy functions that can be efficiently solved; (b) making existing optimization methods more computationally efficient; or (c) showing how existing optimization techniques can be applied to new vision applications. Strikingly few optimization problems have exact solutions, and yet their impact has been substantial. So the highest payoff would likely come from developing methods to minimize entirely new energy functions, although this is technically extremely challenging.

Below we briefly describe representative examples of ongoing work. Since the impact of discrete optimization methods on computer vision comes primarily from their practical efficiency, any substantial speedup is important, whether or not it comes with mathematical guarantees. One promising approach exploits the structure of typical problem instances to reduce running time.

Finding new applications for existing discrete methods is perhaps the most active area of current research. Higher-order cliques are a natural generalization of the energy functions we emphasized, and serve to illustrate several important techniques. The techniques we describe below, which draw on relaxation methods, do not solve the discrete optimization problem exactly, and there is currently no proof that the solution computed is close to the global minimum.

8.1 Data-dependent speedups

Some existing methods can be made substantially faster in practice by taking advantages of usual properties of the input data. For example, in dynamic programming it is not always necessary to fill in the entire set of tables B_i described in section 4. For some problems if we use Dijkstra's algorithm, A* search or generalizations of these methods [35] many of the entries in these tables will remain empty when the optimal solution is found. Similarly, coarse-to-fine methods can often exploit regularities in the input data to good effect [86, 85].

A common property of these techniques is that they do not usually improve the worst case running time of the energy minimization problem. Instead, they may use heuristics that significantly speed up the computation on typical instances. For example, [40] considers the problem of road detection and gives an algorithm inspired in the "twenty questions" game for finding the most likely location of the road while doing as little computation as possible.

In some cases it is possible to prove good *expected* running time if we assume a particular distribution on the input. The method for road detection in [26] assumes the input comes from a particular distribution to obtain an algorithm that will quickly find an optimal or near optimal solution with high probability. This basic idea first appeared in [55] for the problem of finding an optimal path from the root to a leaf in a binary tree with random costs. If the costs on each edge are assumed to be independent [55] shows how a path of near-optimal cost can be found very quickly with high probability.

The method in [85] is particularly interesting in the way that it combines dynamic programming with coarse-to-fine computation to obtain an algorithm that is still polynomial in the worst case. Consider the sequence labeling problem from section 4.1. Dynamic programming can find an optimal labeling in $O(nk^2)$ time where n is the number of elements in the sequence and k is the number of possible labels. Even though the running time is polynomial in the number of labels, if this is very large the computation will be infeasible. The method in [85] constructs a series of coarse approximations to the original problem by aggregating possible labels for each element into "superlabels". Initially we have a small number of possible superlabels for each element. An optimal solution to the coarse problem is found using standard dynamic programming and the superlabel that is assigned to an element is refined into a set of smaller superlabels to define a new problem. The process is iterated until the optimal solution uses only singleton labels. If the coarse problems are defined in a particular way the final solution is a global minimum of the original problem. In many cases this leads to a much faster algorithm in practice. It is possible to prove the method has good expected running time if we assume the input data has certain regularities, but in the worst case the method can actually be slower than the dynamic programming solution to the original problem.

8.2 Efficient algorithms for higher-order cliques

Many of the energy functions we have discussed are of the form given in equation (2), where the prior involves pairs of adjacent pixels. Yet many natural priors cannot be expressed this way; for example, to limit the curvature of a boundary requires considering three or more adjacent pixels at a time. It is straightforward to incorporate such a constraint in a dynamic programming algorithm for sequence labeling, as described in section 4.3, at some additional computational cost. It is much more difficult to incorporate such constraints into the energy functions that arise in pixel labeling problems such as stereo.

Dynamic programming cannot be easily applied to pixel labeling problems because the grid has high treewidth. Priors that depend directly on more than two variables are referred to as "higher-order cliques" (the term comes from the MRF literature). Certain higher-order cliques can be handled by generalizing the graph cut methods of [20]; for example, [62] showed how to do this for a natural generalization of the Potts model, where a group of variables pays a uniform penalty unless they all take on the same value.

A representative example of current research is provided by [49], which draws on several important developments in graph cuts. To use a move-making technique, such as the expansion move algorithm, with higher-order cliques it is necessary to solve a binary problem that is more complex than equation (13). The main result of [49] is a transformation of this more complex binary problem into the form of equation (13), at the price of introducing additional binary variables.

There are two additional challenges that [49] addresses. First, the resulting binary problem is usually not regular, so max flow cannot be used to solve it exactly. [49] relies on a relaxation technique originally invented by [12], which was introduced into computer vision by [66]. This technique (called either roof duality or QPBO) uses max flow to compute a partially optimal solution to the binary problem. Only a subset of the variables are given a value, but if we start at an arbitrary labeling and assign this subset their corresponding values, the energy will not increase. This implies that there is an optimal solution that is an extension of these values. While this relaxation method can perform well in practice, there are few theoretical guarantees (for example, it is possible that no variable is given a value).

The second challenge that [49] addresses is that for some applications a better move-making algorithm is needed than expansion moves or swap moves. A generalization known as fusion moves [71] can be applied to many such problems. In a fusion move, two candidate solutions are merged and every pixel makes a binary decision as to which candidate solution's label to adopt. The expansion move algorithm can be viewed as a fusion move, where one candidate solution is the current labeling and the other assigns a single label to every pixel.

By relying on roof duality and fusion moves, [49]'s main result can be applied to several applications with higher-order cliques. While these methods use established graph cut techniques as subroutines, they do not guarantee an exact solution. Yet they can perform quite well for practical applications such as [103].

9 Conclusions

The power of discrete optimization techniques such as dynamic programming and graph algorithms arises from exploiting the specific structure of important problems. These algorithms often provide strong guarantees about the quality of the solutions they find, which allows for a clean separation between how a problem is formulated (the choice of objective function) and how the problem is solved (the choice of optimization algorithm). The methods are versatile, and arise in very different vision applications; for example, sequence matching via dynamic programming can be used for both shape recognition and for stereo. Similarly min-cut algorithms can be used both for stereo and binary image segmentation. These apparently unrelated problems share structure that can be exploited by discrete optimization methods.

Acknowledgments

We received help from many people with the content and presentation of this material, and especially wish to thank Jon Kleinberg, David Kriegman, Eva Tardos and Phil Torr, as well as the anonymous reviewers. This research has been supported by NSF grants IIS-0746569 and IIS-0803705 and by NIH grants R21EB008138 and P41RR023953.

References

[1] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive digital photomontage. *ACM Transactions on Graphics (SIGGRAPH)*, 23(3):292–300, 2004.

[2] Ravindra K. Ahuja, Özlem Ergun, James B. Orlin, and Abraham P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1–3):75–102, 2002.

[3] Amir Amini, Terry Weymouth, and Ramesh Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):855–867, 1990.

[4] Y. Amit and A. Kong. Graphical templates for model registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):225–236, 1996.

[5] H. H. Baker and T. O. Binford. Depth from edge and intensity based stereo. In *International Joint Conference on Artificial Intelligence*, pages 631–636, 1981.

[6] Stephen Barnard. Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1):17–32, 1989.

[7] R. Basri, L. Costa, D. Geiger, and D. Jacobs. Determining the similarity of deformable shapes. *Vision Research*, 38:2365–2385, 1998.

[8] R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[9] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, April 2002.

[10] S. Birchfield and C. Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *International Conference on Computer Vision (ICCV)*, pages 489–495, 1999.

[11] A. Blake and A. Zisserman. *Visual Reconstruction.* MIT Press, 1987.

[12] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1-3), 2002.

[13] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[14] Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient N-D image segmentation. *International Journal of Computer Vision*, 2006.

[15] Yuri Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *International Conference on Computer Vision (ICCV)*, pages I: 105–112, 2001.

[16] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[17] Yuri Boykov and Olga Veksler. Graph cuts in vision and graphics: Theories and applications. In N. Paragios, editor, *Mathematical Models in Computer Vision: The Handbook*, pages 79–95. Springer, 2005.

[18] Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov Random Fields with efficient approximations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 648–655, 1998.

[19] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision (ICCV)*, pages 377–384, 1999.

[20] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[21] Myron Z. Brown, Darius Burschka, and Gregory D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):993–1008, 2003.

[22] Rainer Burkard, Bettina Klinz, and Rudiger Rudolf. Perspectives of monge properties in optimization. *Discrete and Applied Math*, 70(2):95–161, 1996.

[23] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley & Sons, 1998.

[24] T. H. Cormen, C.E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1989.

[25] J. Coughlan, A. Yuille, C. English, and D. Snow. Efficient deformable template detection and localization without user initialization. *CVIU*, 78(3):303–319, June 2000.

[26] J. Coughlan and A. L. Yuille. Bayesian A* tree search with expected O(N) node expansions: Applications to road tracking. *Neural Computation*, 14(8):1929–1958, 2006.

[27] W. H. Cunningham. Minimum cuts, modular functions, and matroid polyhedra. *Networks*, 15:205–215, 1985.

[28] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[29] T.J. Darrell, D. Demirdjian, N. Checka, and P.F. Felzenszwalb. Plan-view trajectory estimation with dense stereo background models. In *International Conference on Computer Vision (ICCV)*, 2001.

[30] F. Dellaert. *Monte Carlo EM for Data-Association and its Applications in Computer Vision*. PhD thesis, CMU, September 2001.

[31] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1, 1959.

[32] P. F. Felzenszwalb. Representation and detection of deformable shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(2):208–220, February 2005.

[33] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, September 2004. TR2004-1963.

[34] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1), 2005.

[35] P. F. Felzenszwalb and D. McAllester. The generalized A* architecture. *Journal of Artificial Intelligence Research*, 29:153–190, 2007.

[36] M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computer*, 22(1), 1973.

[37] L. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[38] Michael Garey and David Johnson. *Computers and Intractability.* W. H. Freeman and Co., 1979.

[39] D. Geiger, A. Gupta, L. A. Costa, and J. Vlontzos. Dynamic-programming for detecting, tracking, and matching deformable contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(3):294–302, March 1995.

[40] D. Geman and B. Jedynak. An active testing model for tracking roads in satellite images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(1):1–14, January 1996.

[41] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[42] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 51(2):271–279, 1989.

[43] Anupam Gupta and Eva Tardos. A constant factor approximation algorithm for a class of classification problems. In *ACM Symposium on Theoretical Computer Science*, 2000.

[44] P. L. Hammer, P. Hansen, and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematical Programming*, 28:121–155, 1984.

[45] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel. *Robust Statistics: The Approach Based on Influence Functions.* Wiley, 1986.

[46] Marsha Jo Hanna. *Computer Matching of Areas in Stereo Images.* PhD thesis, Stanford University, 1974.

[47] John Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press, 1975.

[48] B. K. P. Horn and B. Schunk. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[49] H. Ishikawa. Higher-order clique reduction in binary graph cut. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[50] H. Ishikawa and D. Geiger. Occlusions, discontinuities, and epipolar lines in stereo. In *European Conference on Computer Vision (ECCV)*, pages 232–248, 1998.

[51] H. Ishikawa and D. Geiger. Segmentation by grouping junctions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 125–131, 1998.

[52] Hiroshi Ishikawa. Exact optimization for Markov Random Fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1333–1336, 2003.

[53] I.H. Jermyn and H. Ishikawa. Globally optimal regions and boundaries as minimum ratio weight cycles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1075–1088, October 2001.

[54] R. Kannan, S. Vempala, and A. Vetta. On clusterings: good, bad, and spectral. *Journal of the ACM*, 51(3):497–515, 2004.

[55] R. M. Karp and J. Pearl. Searching for an optimal path in a tree with random costs. *Artificial Intelligence*, 21(1):99–116, 1983.

[56] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

[57] Junhwan Kim, Vladimir Kolmogorov, and Ramin Zabih. Visual correspondence using energy minimization and mutual information. In *International Conference on Computer Vision (ICCV)*, pages 1033–1040, 2003.

[58] Junhwan Kim and Ramin Zabih. Automatic segmentation of contrast-enhanced image sequences. In *International Conference on Computer Vision (ICCV)*, pages 502–509, 2003.

[59] Jon Kleinberg and Eva Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and Markov Random Fields. *Journal of the ACM*, 49(5):616–639, 2002.

[60] Jon Kleinberg and Eva Tardos. *Algorithm Design.* Addison Wesley, 2005.

[61] D. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, February 1977.

[62] Pushmeet Kohli, M. Pawan Kumar, and Philip H.S. Torr. P3 and beyond: Move making algorithms for solving higher order functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:1645–1656, 2008.

[63] Pushmeet Kohli and Phil Torr. Efficiently solving dynamic Markov Random Fields using graph cuts. In *International Conference on Computer Vision*, 2005.

[64] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–59, 2004.

[65] Vladimir Kolmogorov, Antonio Criminisi, Andrew Blake, Geoffrey Cross, and Carsten Rother. Probabilistic fusion of stereo with color and contrast for bi-layer segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1480–1492, 2006.

[66] Vladimir Kolmogorov and Carsten Rother. Minimizing nonsubmodular functions with graph cuts-a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1274–1279, 2007.

[67] Nikos Komodakis and Georgios Tziritas. A new framework for approximate labeling via graph cuts. In *International Conference on Computer Vision (ICCV)*, 2005.

[68] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2005.

[69] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

[70] Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics (SIGGRAPH)*, 2003.

[71] V Lempitsky, C Rother, S Roth, and A Blake. Fusion moves for Markov Random Field optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010. To appear.

[72] M. H. Lin and C. Tomasi. Surfaces with occlusions from layered stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):1073–1078, 2004.

[73] H. Ling and D. W. Jacobs. Using the inner-distance for classification of articulated shapes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages II: 719–726, 2005.

[74] M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35(2), June 1990.

[75] D. R. Martin, C. C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):530–549, May 2004.

[76] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[77] U. Montanari. On the optimal detection of curves in noisy pictures. *Communications of the ACM*, 14(5), 1971.

[78] Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In *SIGGRAPH*, pages 191–198, 1995.

[79] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(2):139–154, 1985.

[80] C. Papadimitriou and K. Stieglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, 1982.

[81] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, Jul 1997.

[82] Tomaso Poggio, Vincent Torre, and Christof Koch. Computational vision and regularization theory. *Nature*, 317:314–319, 1985.

[83] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C.* Cambridge, 1992.

[84] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[85] C. Raphael. Coarse-to-fine dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:1379–1390, 2001.

[86] C. Raphael and S. Geman. A grammatical approach to mine detection. In *SPIE, 3079*, pages 316–337, 1997.

[87] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut" - interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (SIGGRAPH)*, 23(3):309–314, 2004.

[88] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake. Digital tapestry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[89] S. Roy. Stereo without epipolar lines: A maximum flow formulation. *International Journal of Computer Vision*, 1(2):1–15, 1999.

[90] S. Roy and I. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *International Conference on Computer Vision (ICCV)*, 1998.

[91] Iwata Satoru, Fleischer Lisa, and Fujishige Satoru. A combinatorial, strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48(4):761–777, 2001.

[92] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, 2002.

[93] T. Schoenemann and D. Cremers. Matching non-rigidly deformable shapes across images: A globally optimal solution. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6, 2008.

[94] T. B. Sebastian, P. N. Klein, and B. B. Kimia. On aligning curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1):116–124, January 2003.

[95] A. Shashua and S. Ullman. Structural saliency: The detection of globally salient structures using a locally connected network. In *IEEE International Conference on Computer Vision*, pages 321–327, 1988.

[96] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[97] Rick Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for Markov Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068–1080, 2008.

[98] A. Tikhonov and V. Arsenin. *Solutions of ill-posed problems.* Winston and Sons, 1977.

[99] O. Veksler. Stereo correspondence by dynamic programming on a tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 384–390, 2005.

[100] Olga Veksler. *Efficient Graph-based Energy Minimization Methods in Computer Vision.* PhD thesis, Cornell University, 1999.

[101] M. Wainwright, T. Jaakkola, and A. Willsky. Map estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 5(11):3697–3717, 2005.

[102] Y. Weiss, C. Yanover, and T. Meltzer. Map estimation, linear programming and belief propagation with convex free energies. In *Uncertainty in AI*, 2007.

[103] Oliver Woodford, Philip Torr, Ian Reid, and Andrew Fitzgibbon. Global stereo reconstruction under second-order smoothness priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:2115–2128, 2009.