MESSAGE PASSING ALGORITHMS FOR FACILITY LOCATION PROBLEMS

by

Nevena Lazic

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

Copyright © 2011 by Nevena Lazic
Abstract

Message Passing Algorithms for Facility Location Problems

Nevena Lazic
Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto
2011

Discrete location analysis is one of the most widely studied branches of operations research, whose applications arise in a wide variety of settings. This thesis describes a powerful new approach to facility location problems - that of message passing inference in probabilistic graphical models. Using this framework, we develop new heuristic algorithms, as well as a new approximation algorithm for a particular problem type.

In machine learning applications, facility location can be seen a discrete formulation of clustering and mixture modeling problems. We apply the developed algorithms to such problems in computer vision. We tackle the problem of motion segmentation in video sequences by formulating it as a facility location instance and demonstrate the advantages of message passing algorithms over current segmentation methods.
Dedication

To my mother, Slavica Lazic.
Acknowledgements

I would first like to thank my Ph.D. supervisor Parham Aarabi for guiding and supporting me throughout my graduate studies. I would also like to thank my co-supervisor Brendan J. Frey for his insights, advice and help, as well as for welcoming me to the PSI research group.

I am very grateful to the members of the thesis examination committee Wei Yu and Kostas Plataniotis and the external examiner Shai Ben-David for their time and constructive criticism. I would also like to acknowledge Inmar Givoni and Danny Tarlow for their research insights.

I would like to thank the past and present members of the APL and PSI research groups, as well as other officemates, for making my graduate studies a fun and memorable experience - especially Sam Mavandadi, Steven Rennie, Ron Appel, Hayssam Dahrouj and Danilo Silva. I would also like to thank my family and friends for their support and understanding.

Finally, I would like to thank the University of Toronto, the Rogers family, and the Natural Sciences and Engineering Council of Canada for supporting this research.
# Contents

1 Introduction .................................................. 1
   1.1 Discrete Facility Location Problems .......................... 2
   1.2 Inference in Probabilistic Graphical Models ................. 4
   1.3 Applications to Machine Learning and Computer Vision .... 6
   1.4 Thesis Outline .............................................. 8

2 Background .................................................. 10
   2.1 Discrete Facility Location .................................. 10
      2.1.1 Facility Location Problem Variants ..................... 10
      2.1.2 The Uncapacitated Facility Location Problem .......... 13
      2.1.3 Facility Location and Machine Learning ............... 16
   2.2 Probabilistic Graphical Models ............................. 19
      2.2.1 Factor Graphs and the Max-Product Algorithm ........ 20
      2.2.2 Max-Product Linear Programming ....................... 22
   2.3 Affinity Propagation .................................... 24

3 Max-Product Algorithm for Facility Location Problems .......... 27
   3.1 Uncapacitated Facility Location ............................ 28
   3.2 Capacitated Facility Location .............................. 32
   3.3 $k$-Facilities ............................................ 34
   3.4 Clustering Applications of FL Algorithms .................. 38
6 FLoSS: Facility Location for Subspace Segmentation

6.1 Subspace Segmentation
   6.1.1 Previous Work
   6.1.2 Experiments on Synthetic Data

6.2 Multibody Motion Segmentation
   6.2.1 3D Motion Geometry
   6.2.2 Hopkins155 Motion Segmentation Dataset
   6.2.3 Hopkins155 Experiments

6.3 Discussion

7 Conclusions and Future Directions
## List of Tables

4.1 MPLP constraints and corresponding dual variables ........................................ 56

5.1 ORLIB parameters .............................................................................................. 67

5.2 Bilde-Krarup Sequences ($q = 1, \ldots, 10$) .................................................... 69

5.3 $M^*$ parameters ................................................................................................. 70

5.4 Galvao-Raggi sequences ..................................................................................... 70

5.5 Number of instances solved to optimality for each data set ................................. 72

5.6 Percentage error, measured as the amount by which the obtained cost exceeds the optimal cost or its lower bound, averaged over all instances in each data set. ........................................................................................................... 72

5.7 Average number of iterations required to convergence for each data set. ............ 73

6.1 Motion segmentation percent error, 2 objects .................................................... 87

6.2 Motion segmentation percent error, 3 objects .................................................... 87

7.1 Table of Acronyms ................................................................................................. 90
List of Figures

1.1 An example of a facility location problem, where the goal is to open new schools (facilities) in Toronto at a subset of available locations, represented by academic hats. The schools serve Toronto’s residential neighborhoods (customers), represented by stick figures. There is a cost associated with building and running each school, which may vary according to location. There is also a cost associated with assigning a neighborhood to the district of each school. The maximum number of students that can be served by any given school is its \textit{capacity}. The task is to open enough schools to accommodate all students in the most cost-effective manner; one possible solution is marked in red. .................................................. 3

1.2 Exemplar-based clustering problem is the task of grouping data points into clusters, where each cluster is represented by a single exemplar, as illustrated here on a subset of images from the Olivetti database [66]. The problem can be seen as an instance of facility location, where the customer and facility sets are the same. .................................................. 7
1.3 Clustering results on a toy data set containing 5 clusters, shown as a function of preferences $p$ for standard Affinity Propagation (top) and Affinity Propagation constrained to find exactly 5 clusters (bottom). When the number of clusters is known, the added model flexibility of constrained AP helps circumvent the search over the preference parameters in order to find the correct number of clusters. 8

1.4 3-D motion segmentation in video is the task of grouping tracked points lying on moving objects according to object. The figure shows example frames and keypoints from the benchmark Hopkins155 [78] motion segmentation database. 9

2.1 An illustration of a FL problem, where smileys represent customers, houses represent facilities, and facility and connection costs are $F_j$ and costs $c_{ij}$, respectively. The goal is to connect customers to one facility each at minimal total cost. Solid edges show one possible solution, where facilities 1 and 2 are open. 12

2.2 In metric FL problems, connection costs $c_{ij}$ satisfy the triangle inequality $c_{ij} \leq c_{ik} + c_{lk} + c_{lj}$, illustrated in the figure. 13

2.3 An example of model selection applied to polynomial regression, illustrating the importance of the tradeoff between complexity (polynomial degree) and goodness-of-fit (point-curve distance). The data points are perfectly explained by the high-order polynomials; however, they are simply noisy observations of a linear model. 18

2.4 An example of data coming from multiple models of different complexities - a straight line and a polynomial of degree 3. 18
2.5 Example of a factor graph, where circles represent variable nodes and squares represent factor nodes. The factor graph represents a distribution that factorizes according to $\theta$ as $p(x_1, x_2, x_3) \propto \exp(-\theta_1(x_1, x_2)) \exp(-\theta_2(x_2, x_3))$.

2.6 An illustration of the LP relaxation of the MAP inference problem. For each potential $\theta_c(x_c)$, we introduce a distribution $\mu_c(x_c)$ over all configurations of $x_c$ and perform the optimization with respect to $\mu$. The distributions $\mu$ are constrained to be positive, normalized, and to agree on intersection variable sets. In the figure, $\mu_1(x_1, x_2)$ and $\mu_2(x_2, x_3)$ agree on the marginal $\mu(x_2)$, while $\mu_2(x_2, x_3)$ and $\mu_3(x_3, x_4)$ agree on the marginal $\mu(x_3)$. When the LP solution $\mu^*$ is integral, the relaxation is tight and the corresponding $x^*$ is optimal.

2.7 Factor graph representation of Affinity Propagation.

3.1 UFL factor graph.

3.2 UFL message naming convention.

3.3 Factor graph representation of CFL.

3.4 Message naming convention for CFL.

3.5 k-facilities factor graph. The HMM over the $y_j$ variables counts the number of open facilities $\sum_j y_j$, and factor $G_{M+1}$ incorporates an arbitrary prior over it.

3.6 k-facilities message naming convention.

3.7 Illustrating the relationship between preferences and clustering granularity in AP on a toy data set, where similarities set to negative squared Euclidean distance and all preferences are equal. The top figure shows clusters obtained by AP for several preference settings, while the bottom figure plots the number of clusters vs. preference.
3.8 Clustering synthetic data via regular AP and $k$-AP over different preference settings. A clustering with 5 exemplars can be obtained by either varying the preference setting in AP, or enforcing $k = 5$ in the $k$-AP prior.

3.9 Clustering synthetic data via regular AP and $k$-AP over different preference settings. Clusterings with 2 or 6 exemplars can be obtained by either varying the preference setting in AP, or enforcing $k = 2$ or $k = 6$ in the $k$-AP prior.

3.10 NASA 25th Anniversary Show, Segment 01. The figure shows video summaries obtained via $k$-AP with $k \in \{5, 7, 9\}$, regular AP, and the Open-Video storyboard.

3.11 NASA 25th Anniversary Show, Segment 07. The figure shows video summaries obtained via $k$-AP with $k \in \{5, 7, 9\}$, regular AP, and the Open-Video storyboard.

4.1 UFL factor graph

4.2 Top: An MPLP fixed point with unresolved variables. Middle: an integral solution that satisfies (CS 1) but not (CS 2). Bottom: an integral solution that satisfies (CS 2), but not (CS 1).

4.3 Illustration of the greedy decoding algorithm. (A) Select the min-weight edge. (B) Open the corresponding facility and connect all customers. (C) Remove all facilities 2 edges away from the opened facility. (D) Once no facilities are available, connect remaining customers to the closest open facility.
4.4 Experimental results comparing belief decoding and greedy decoding on synthetic metric clustering problems. Connection costs $c_{ij}$ are set to pairwise Euclidean distances between $N$ points randomly generated in a unit square. Facility costs are set to $\sqrt{N}/10$ (top), $\sqrt{N}/100$ (middle), and $\sqrt{N}/1000$ (bottom). Error is measured as the percentage by which the obtained cost exceeds the LP lower bound. .......................... 54

4.5 Top: a MPLP fixed point with unresolved variables. Middle: arbitrarily assigning variables by maximizing beliefs can lead to bad solutions, and potentially all facilities being open. Bottom: greedy Alg. ?? opens facilities conservatively, but may not maximize all customer beliefs. .............. 55

6.1 Examples of data lying on multiple linear subspaces ......................... 75

6.2 Comparison of different algorithms on data sets consisting of planes, (a) RANSAC, (b) mpPCA, (c) GPCA, and (d) FLoSS .......................... 80

6.3 Mixed dimensionality subspaces, two noise levels: $\sigma^2 = 0.01$ (top row) and $\sigma^2 = 0.05$ (bottom row). (a) RANSAC, (b) mpPCA, (c) GPCA, and (d) FLoSS ......................................................... 81

6.4 Example frames with keypoints (left) and trajectories (right) of checkerboard, traffic, and articulated motion sequences from the Hopkins155 database. The keypoints colors denote hand labeled objects. ............. 84

6.5 Checkerboard sequence, first 3 principal components. (a) Ground truth, (b) FLoSS, (c) GPCA, and (d) LSA ............................... 86

6.6 Traffic sequence, first 3 principal components. (a) Ground truth, (b) FLoSS, (c) GPCA, and (d) LSA ...................................... 87
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL</td>
<td>Facility location</td>
</tr>
<tr>
<td>UFL</td>
<td>Uncapacitated facility location</td>
</tr>
<tr>
<td>CFL</td>
<td>Capacitated facility location</td>
</tr>
<tr>
<td>MAP</td>
<td>Maximum-a-posteriori</td>
</tr>
<tr>
<td>LP</td>
<td>Linear programming</td>
</tr>
<tr>
<td>MPLP</td>
<td>Max-product linear programming</td>
</tr>
<tr>
<td>BP</td>
<td>Belief propagation</td>
</tr>
<tr>
<td>ORLIB</td>
<td>Operations research library</td>
</tr>
</tbody>
</table>

Table 1: Table of Acronyms
Chapter 1

Introduction

This work describes a new approach to solving discrete facility location problems, which fall among the most widely studied questions in operations research. We tackle facility location using the powerful technique of message passing algorithms in probabilistic graphical models. For an important subfamily of facility location problems, we additionally provide approximation guarantees. Although certain clustering algorithms can be interpreted as solving special instances of facility location via inference in graphical models [16,26,28], this thesis contains the first systematic application and evaluation of message passing for facility location problems, as well as the first approximation algorithm that is based on this approach.

We show that a number of important tasks in machine learning can be described as facility location instances, and apply message passing algorithms to those problems. Using insights from facility location problems, we generalize Affinity Propagation [26], a well-known clustering algorithm. We also interpret the computer vision problem of motion segmentation in video as an instance of facility location, and demonstrate that message passing algorithms overcome some of the shortcomings of current segmentation methods.
Chapter 1. Introduction

1.1 Discrete Facility Location Problems

Facility location problems have occupied a central position in operations research and management science since the 1960s. They have been used to model the optimal placement of factories, warehouses, fire stations, hospitals, bus stops, subway stations, electronic switching centers and satellites, to only name a few examples [50, 57, 71]. As one researcher puts it, “humans have been analyzing the effectiveness of locational decisions since they inhabited their first cave” [19].

More recent applications of location analysis include network design [6, 58], self-configuration in wireless sensor networks [25], constructing treatment portfolios in medicine and biology [21] and motion segmentation in computer vision [51, 53]. Researchers have also recognized more general machine learning problems as facility location instances; these include variants of exemplar-based clustering, multiple model selection, and subspace segmentation [21, 51, 53].

In a discrete facility location (FL) problem, one is typically given the following information:

- A set of facilities $F$ which may be opened to serve customers, along with the opening and/or operating cost $F_j$ for each facility $j$.

- A set of customers $C$, where each customer has a demand for goods or services from an open facility. There is a cost $c_{ij}$ associated with connecting customer $i$ to facility $j$.

The goal is to open a subset of facilities and assign customers to one facility each such that the customer demand is met at minimum total cost. An illustration of the problem is shown in Fig. 1.1.

Different definitions of facility costs can give rise to different problem versions. In the uncapacitated facility location (UFL) problem, facility costs are fixed constants and each facility can serve an unlimited number of customers. In contrast, in capacitated
Figure 1.1: An example of a facility location problem, where the goal is to open new schools (facilities) in Toronto at a subset of available locations, represented by academic hats. The schools serve Toronto’s residential neighborhoods (customers), represented by stick figures. There is a cost associated with building and running each school, which may vary according to location. There is also a cost associated with assigning a neighborhood to the district of each school. The maximum number of students that can be served by any given school is its capacity. The task is to open enough schools to accommodate all students in the most cost-effective manner; one possible solution is marked in red.
facility location (CFL), the cost of a facility is a non-decreasing function of the number of customers it serves. In the related $k$-medians problem, the number of open facilities is constrained to be exactly $k$ and $\mathcal{C} = \mathcal{F}$. An extensive survey of problems, algorithms and applications can be found in [59].

Despite having a relatively simple formulation, facility location problems are computationally intractable in general; finding the optimal solution is $\mathcal{NP}$-hard even in the simplest, uncapacitated case. There are different ways of approaching such problems. Integer programming methods always find the optimal solution and are designed to be efficient on instances of interest, but there are no guarantees on their run time in general. Among polynomial time algorithms, some provide no theoretical guarantees on solution optimality, and researchers empirically demonstrate their effectiveness on instances of interest. On the other hand, $\rho$-approximation polynomial-time algorithms obtain solutions that are provably within a factor $\rho$ of optimal.

There has been extensive research in approximation algorithms for facility location problems, especially for metric UFL [11–13, 31, 38, 39, 56, 74]. This hardly surprising as there is a straightforward reduction to UFL from set cover, a classical question in computer science “whose study has led to the development of fundamental techniques for the entire field” of approximation algorithms [83]. UFL approximation algorithms are typically based on its linear programming (LP) relaxation, a related but tractable convex problem whose solution is a lower bound on the UFL optimum.

1.2 Inference in Probabilistic Graphical Models

In this work, we tackle facility location problems via message passing algorithms in probabilistic graphical models - an approach that has received little attention in the past. Probabilistic graphical models provide a powerful framework for visualizing complex dependencies between random variables in a multivariate distribution. They are widely
used in many fields, including communication theory, signal processing, control systems, computational biology and computer vision. The formalism of graphical models can also be applied to discrete optimization problems such as facility location, by treating the optimization objective as the joint log-likelihood of the optimization variables. The task of finding optimal solution then corresponds to finding the distribution mode, or maximum-a-posteriori (MAP) inference.

Efficient inference algorithms in graphical models can be described as iterative message passing operations between adjacent nodes in the graph. At each iteration, the product of messages received by each variable reflects the belief that it takes on a particular value in the solution. The most commonly used message passing algorithm for MAP inference is the max-product (belief propagation) algorithm [63], which is guaranteed to converge to the optimal solution on trees. Although there are no guarantees on convergence or optimality on loopy graphs in general, it is nevertheless widely used and has shown excellent empirical performance in many applications, most notably in the area of error correcting codes [8]. There has also been much work in developing similar message passing inference algorithms whose properties are better understood. Many recent such algorithms are based on a special LP relaxation of the MAP inference problem [29, 45, 46, 86].

In this work, we present graphical models and corresponding max-product belief propagation algorithms for different variants of the FL problem. As the graphical models are loopy, these algorithms can be seen as heuristics with no optimality guarantees. However, in extensive experiments on UFL, we observe that belief propagation typically outperforms other approaches in both the number of problem instances solved to optimality and the solution quality.

For the metric UFL subfamily of problems, we additionally describe a message-passing algorithm with a $\rho$-approximation guarantee. The approximation algorithm relies on max-product linear programming (MPLP) [29], a “convexified” version of max-product
belief propagation. We modify MPLP solutions using a greedy procedure and show that the resulting solutions are guaranteed to be within a factor 3 of optimal, as well as often having improved empirical performance. Although there exist UFL approximation algorithms with tighter approximation guarantees [11, 13, 37, 56, 69, 74], this is the first approximation algorithm that comes from a message passing approach. It offers insights into the relationship between MPLP and standard LP-based algorithms and suggests directions for improving MPLP solutions on other problems.

1.3 Applications to Machine Learning and Computer Vision

Two important machine problems that can be interpreted as instances of facility location are exemplar-based clustering and multiple model selection.

In exemplar-based clustering, the goal is to group data points into clusters and represent each cluster by a single *exemplar* data point, as illustrated in Fig. 1.2. This can be seen as an instance of facility location, where customers and facilities are the same set.

One notable exemplar-based clustering algorithm is Affinity Propagation (AP) [26], whose formulation corresponds to UFL, the simplest among facility location problems. AP finds exemplars using MAP inference on a probabilistic graphical model, and its success provides the motivation for this work. However, one of its shortcomings is that the solutions it obtains are largely governed by the facility costs (called *preferences* in AP), which are typically unavailable and must be set by hand. We show that CFL and *k*-medians correspond to generalization of AP that incorporates prior beliefs and/or constraints on the cluster number and size. This can circumvent the search over the preference parameters in problems where prior information is available, as illustrated in Fig. 1.3.

In multiple model selection, the goal is to choose a set of models that best explain
Chapter 1. Introduction

Figure 1.2: Exemplar-based clustering problem is the task of grouping data points into clusters, where each cluster is represented by a single exemplar, as illustrated here on a subset of images from the Olivetti database [66]. The problem can be seen as an instance of facility location, where the customer and facility sets are the same.
Figure 1.3: Clustering results on a toy data set containing 5 clusters, shown as a function of preferences $p$ for standard Affinity Propagation (top) and Affinity Propagation constrained to find exactly 5 clusters (bottom). When the number of clusters is known, the added model flexibility of constrained AP helps circumvent the search over the preference parameters in order to find the correct number of clusters.

data from a set of potential models. This too can be viewed as an instance of facility location, where candidate models are facilities and data points are customers. We apply the model selection FL framework to the problem of 3-D motion segmentation from point correspondences in video. In 3-D motion segmentation, the input is a video sequence containing several rigid bodies undergoing translation and/or rotation, with tracked points on each body and possibly background across all frames, as shown in Fig. 1.4. The goal is to group the points according to object. We develop an algorithm called FLoSS - facility location for subspace segmentation, achieving motion segmentation results comparable to the state-of-the-art.

1.4 Thesis Outline

The thesis is organized as follows. Chapter 2 provides an extensive background on both facility location and inference in probabilistic graphical models. Chapter 3 contains the graphical models and max-product inference algorithms corresponding to different
Figure 1.4: 3-D motion segmentation in video is the task of grouping tracked points lying on moving objects according to object. The figure shows example frames and keypoints from the benchmark Hopkins155 [78] motion segmentation database.

variants of the FL problem. Chapter 4 describes the MPLP algorithm for UFL, as well as a greedy algorithm that produces solutions with an approximation guarantee from MPLP. Chapter 5 contains an experimental comparison of message passing to other approaches in literature. Chapter 6 describes an application of the developed message passing algorithms to the problem of motion segmentation in video. Chapter 7 contains a summary of contributions and directions for future work.
Chapter 2

Background

2.1 Discrete Facility Location

2.1.1 Facility Location Problem Variants

In the most general setting of the facility location problem, we are given a set of customers \( \mathcal{C} \) and a set of facilities \( \mathcal{F} \) that can be opened to serve them. The cost of opening a facility \( j \in \mathcal{F} \) is \( F_j(u_j) \), where \( u_j \) is the number of customers assigned to \( j \), and the cost of assigning a customer \( i \) to facility \( j \) is \( c_{ij} \). The goal is to open a subset of facilities and connect customers to one facility each at minimal total cost. An illustration of FL is given in Fig. 2.1.

Let \( x_{ij}, i \in \mathcal{C}, j \in \mathcal{F} \) be a binary indicator variable equal to 1 if customer \( i \) is assigned to facility \( j \) and 0 otherwise. FL can be written as the following integer program:
\[
\min_x \sum_i \sum_j c_{ij} x_{ij} + \sum_j F_j(u_j) \tag{2.1}
\]

s.t. \[
\sum_i x_{ij} = u_j \quad \forall j \in \mathcal{F}, \forall i \in \mathcal{C} \tag{2.2}
\]
\[
\sum_j x_{ij} = 1 \quad \forall i \in \mathcal{C} \tag{2.3}
\]
\[
x_{ij} \in \{0, 1\}, u_j \in \mathcal{Z} \tag{2.4}
\]

In the *metric* problem version, the connection costs \(c_{ij}\) also satisfy the triangle inequality, illustrated in Fig. 2.2:

\[
c_{ij} \leq c_{ik} + c_{lk} + c_{lj} \quad \forall i, l \in \mathcal{C}, \forall j, k \in \mathcal{F} \tag{2.5}
\]

Different facility location problems arise from different definitions of the costs \(F_j(u_j)\). Some of the most common versions are:

- **Uncapacitated**: \(F_j(u_j) = f_j I[u_j > 0]\). There is a fixed cost \(f_j\) for opening a facility \(j\), and an unlimited number of customers can be assigned to each facility.

- **Capacitated**: \(F_j(u_j) = f_j I[u_j > 0] + \infty I[u_j > S_j]\). Here, at most \(S_j\) customers can be assigned to a facility \(j\). \(S_j\) is referred to as \(j\)'s *capacity*.

- **Soft-capacitated**: \(F_j(u_j) = f_j \lceil u_j / S_j \rceil\). An unlimited number of facilities of capacity \(S_j\) can be opened at cost \(f_j\).

- **Linear-cost**: \(F_j(u_j) = f_j I[u_j > 0] + \sigma_j u_j\). The cost is linear in the number of assigned customers.

- **Concave-cost**: \(F_j(u_j)\) are arbitrary concave functions of the number of assigned customers.
A closely related problem to facility location is \( k \)-medians, where \( C = F \) and there is an additional constraint that the number of open facilities is exactly \( k \). We will call \( k \)-facilities a FL problem with an additional cost that depends on the total number of open facilities in the solution. Letting \( z \) be the number of open facilities, this can be written as:

\[
\begin{align*}
\min_{\mathbf{x}} & \quad \sum_i \sum_j c_{ij} x_{ij} + \sum_j F_j(u_j) + G(z) \\
\text{s.t.} & \quad \sum_i x_{ij} = u_j \quad \forall j \in F, \forall i \in C \\
& \quad \sum_j \max_i x_{ij} = z \\
& \quad \sum_j x_{ij} = 1 \quad \forall i \in C \\
& \quad x_{ij} \in \{0, 1\}, u_j \in \mathbb{Z}
\end{align*}
\]
Figure 2.2: In metric FL problems, connection costs $c_{ij}$ satisfy the triangle inequality $c_{ij} \leq c_{ik} + c_{lk} + c_{lj}$, illustrated in the figure.

### 2.1.2 The Uncapacitated Facility Location Problem

The uncapacitated facility location (UFL) problem is one of the most widely studied discrete location problems, to which a substantial part of this work will be devoted. In this section, we review the some UFL properties and previous approaches.

#### UFL Complexity and Approximability

UFL can be stated as the following integer linear program (ILP):

$$\min_{x, y} E(x, y) = \sum_{i} \sum_{j} c_{ij} x_{ij} + \sum_{j} f_{j} y_{j} \quad \text{(2.11)}$$

subject to:

$$\sum_{j} x_{ij} = 1 \quad \forall i \in C \quad \text{(2.12)}$$

$$y_{j} - x_{ij} \geq 0 \quad \forall i \in C, j \in F \quad \text{(2.13)}$$

$$x_{ij}, y_{j} \in \{0, 1\} \quad \forall i \in C, j \in F \quad \text{(2.14)}$$

It can be shown that UFL is $\mathcal{NP}$-hard by reduction from the set cover problem, a classical question in complexity theory and one of Karp’s 21 NP-complete problems [42]. In the optimization version of set cover, the inputs are a universe $\mathcal{U}$ and a family $\mathcal{S}$ of subsets of $\mathcal{U}$. The goal is to find the subfamily $\mathcal{S'} \subseteq \mathcal{S}$ of subsets whose union is
Chapter 2. Background

\( \mathcal{U} \) and that uses the fewest sets. This corresponds to a facility location problem where facilities are subsets (\( \mathcal{F} = \mathcal{S} \)) and elements are customers (\( \mathcal{C} = \mathcal{U} \)), with unit facility and connection costs.

Among algorithms for both facility location and set cover, an important and well-researched class consists of polynomial time approximation algorithms (PTAAs). A \( \rho \)-approximation algorithm for an optimization problem is a PTA whose solution is provably within a factor \( \rho \) of optimal, where \( \rho \) is called the approximation ratio. As shown by [4], many optimization problems have no approximation algorithms with constant \( \rho \) unless \( \mathcal{P} = \mathcal{NP} \). This is the case for the UFL in general; [54] and [22] show that the \( O(\ln |\mathcal{C}|) \)-approximation of Hochbaum [35] cannot be improved unless unless \( \mathcal{NP} \subseteq DTIME(n^{O(\log \log n)}) \). However, Guha and Khuller [31] have shown that metric UFL admits polynomial-time approximation algorithms with constant \( \rho \), and that \( \rho > 1.463 \) unless \( \mathcal{NP} \subseteq DTIME(n^{O(\log \log n)}) \); Sviridenko [74] later showed that \( \rho > 1.463 \) unless \( \mathcal{P} = \mathcal{NP} \). Researchers also frequently consider \((\rho_f, \rho_c)\)-approximation algorithms for UFL [12], which obtain a solution of cost of at most \( \rho_f F^* + \rho_c C^* \), where \( F^* \) and \( C^* \) are the optimal facility and customer costs, respectively. Jain et. al. [38] have shown that there exists no \((\rho_f, \rho_c)\)-approximation algorithm with \( \rho_c < 1 + 2 \exp^{−\rho_f} \), unless \( \mathcal{NP} \subseteq DTIME(n^{O(\log \log n)}) \).

Approximation Algorithms for Metric UFL

Techniques for designing approximation algorithms for metric UFL are primarily based on its linear programming (LP) relaxation, where the integrality constraints \( x_{ij}, y_j \in \{0, 1\} \) are replaced by the weaker non-negativity constraints \( x_{ij}, y_j \geq 0 \). The LP is solvable in polynomial time and its solution gives a lower bound on the optimal integral solution, \( E(x^{LP}, y^{LP}) \leq E(x^{OPT}, y^{OPT}) \). A \( \rho \)-approximation algorithm constructs an integral solution \((x^*, y^*)\) that is at most \( \rho \) times worse than the LP solution, and hence at most \( \rho \) times worse than the optimal ILP solution:
Two common approaches to constructing approximation algorithms are LP rounding and primal-dual methods. Rounding algorithms typically start by solving the LP. If the obtained LP solution is integral, it is also optimal for the ILP as it achieves the lower bound; otherwise, clever techniques are used to round fractional solution values. For UFL, a popular approach is to construct the solution support graph - a bipartite graph in which nodes represent customers and facilities and weighted edges connect each customer-facility pair \((i,j)\) for which \(x_{ij} > 0\) in the LP solution. An integral solution is obtained by greedily clustering the customer nodes, and assigning all cluster members to the cluster center’s closest facility. Optimality claims are proven using the LP solution and the triangle inequality. LP rounding algorithms of \([11,13,69,74]\) differ in the greedy criterion for choosing cluster centers and in graph pre-processing.

Primal-dual approximation algorithms are based on the primal-dual method for solving LPs. An LP is a convex problem whose dual problem is another LP; it is solved to optimality when there exist feasible primal and dual solutions \(p^*\) and \(d^*\) for which the primal and dual objectives are equal, and complementary slackness conditions hold. In the primal-dual method for solving LPs, one starts with a feasible dual solution \(d\) and attempts to find a feasible primal solution \(p\) that satisfies complementary slackness conditions with respect to \(d\). If none exists, \(d\) is modified and the process repeated.

Primal-dual approximation algorithms use a similar approach, but construct integral primal solutions \(p\), while relaxing a subset of the complementary slackness conditions. One such algorithm is the 3-approximation algorithm of \([39]\), where the integral solution \((x,y)\) is based on a support graph with edges connecting customer-facility pairs for which the complementary slackness constraints are tight. \([37]\) implicitly use primal-dual analysis to prove that their greedy heuristic guarantees an approximation ratio of 1.61. \([56]\) further combine the algorithm of \([37]\) with the greedy augmentation procedure introduced by \([31]\)
to obtain a 1.52-approximation guarantee.

We note that there exist a number of polynomial run-time algorithms with no theoretical guarantees that have empirically been shown to yield excellent results on UFL benchmarks. Such approaches include simulated annealing [3], genetic algorithms [48], tabu searches [2,33,73], and local searches [5,12,47].

2.1.3 Facility Location and Machine Learning

Many important theoretical and practical problems in machine learning can be formulated as facility location instances. We describe two notable examples: multiple model selection and exemplar-based clustering.

Multiple model selection

Model selection is the task of selecting a suitable data model from a set of potential models, given a set of observations. For example, in polynomial regression problems, the goal is to fit a polynomial curve of a suitable degree such that the data is explained well without overfitting, as illustrated in Fig. 2.3.

Popular criteria for model selection, such as the Bayesian Information Criterion (BIC), Akaike Information Criterion (AIC), and Minimum Message Length (MML), typically balance the goodness-of-fit of data with model complexity in order to find the simplest model that explains the data well. Given a set of $N$ data points $\mathbf{d} = \{d_1, \ldots, d_N\}$ and $M$ models, where each model $m_j$ has $k_j$ parameters, the BIC selects the best model $m^*$ according to:

$$m^* = \arg \min_{m_j} -2L(\mathbf{d}|m_j) + k_j \ln(N)$$

(2.16)

where $L(\mathbf{d}|m_j)$ is the maximized log-likelihood of the data under model $m_j$.

Now suppose the data comes from several models of different complexities, as in the
example in Fig. 2.4. Suppose also that the data points are independent given the model, so that likelihood decomposes as $L(d|m_j) = \sum_{i=1}^{N} \ln p(d_i|m_j)$. In this case, multiple model selection using the BIC can be written as:

$$\min_{y, x} \sum_{ij} -2 \ln p(d_i|m_j)x_{ij} + \sum_j k_j \ln(N_j)y_j$$

s.t.  
$$y_j \geq x_{ij} \quad \forall i, j$$  
$$\sum_i x_{ij} = N_j \quad \forall i, j$$
$$x_{ij}, y_j \in \{0, 1\}$$

where variables $y_j$ indicate whether model $j$ has been selected, and variables $x_{ij}$ indicate whether point $i$ comes from model $j$. This is an instance of facility location, with costs set to $F_j = k_j y_j \ln N_j$ and $c_{ij} = -2 \ln p(d_i|m_j)$. The same framework can be also be used in conjunction with other model selection criteria such as AIC and MML, and has been applied to 3-D motion segmentation in computer vision [51, 53].

**Exemplar-based clustering**

Clustering is a fundamental problem in unsupervised learning with broad applications. The goal is to discover categories of data points by grouping them into clusters with low intra-cluster and high inter-cluster variability. Many widely used methods such as $k$-means and Gaussian mixture models seek underlying cluster means, such that cluster members lie close to their mean and the means are far from one another. However, for high-dimensional datasets such as images and videos, the cluster average may not always be meaningful in itself, and may in fact lie far from the cluster members. Methods such as spectral clustering [68], [61] attempt to circumvent these difficulties by mapping data to a low-dimensional manifold prior to clustering. On the other hand, **exemplar-based clustering** methods represent each cluster by an exemplar - a data point that is repre-
Figure 2.3: An example of model selection applied to polynomial regression, illustrating the importance of the tradeoff between complexity (polynomial degree) and goodness-of-fit (point-curve distance). The data points are perfectly explained by the high-order polynomials; however, they are simply noisy observations of a linear model.

Figure 2.4: An example of data coming from multiple models of different complexities - a straight line and a polynomial of degree 3.
sentative of the other cluster members. Although the later approach frames clustering as a combinatorial optimization problem of choosing exemplars, efficient algorithms for finding approximate solutions have recently been developed [26], [15].

In exemplar-based clustering, the input is typically a set of pairwise similarities $s_{ij}$ between data points. The goal is to select exemplars such that the sum of similarities between points and their exemplars is maximized. Unfortunately, unconstrained maximization of this objective would result in each point being its own exemplar, as any point is certainly most similar to itself. For that reason, the optimization objective also needs to include a regularization term that penalizes large exemplar sets. When the regularization is linear or additive in the number of selected exemplars, exemplar-based clustering corresponds to a facility location problem with $C = F$ and connection costs $c_{ij} = -s_{ij}$.

\section{2.2 Probabilistic Graphical Models}

Probabilistic graphical models are widely used in many fields, including communication theory, signal processing, control systems, computational biology and computer vision. They provide a powerful framework for describing complex dependencies between random variables in a multivariate distribution. Basic inference tasks, such as computing variable marginals or the distribution mode, can be performed efficiently through recursive operations on the graph.

The formalism of graphical models is also applicable to combinatorial optimization problems such as facility location. Given an optimization objective $E(x)$, the variables are endowed with a Gibbs distribution $p(x) = \exp(-E(x))$. The task of finding an optimal solution $x^* = \arg\min_x E(x)$ is equivalent to finding the distribution mode, or performing maximum-a-posteriori (MAP) inference. In this context, the objective $E(x)$ is often referred to as energy.

There are several different types of graphical models. Directed graphs such as Bayesian
networks [64] are typically used to represent hierarchical dependencies between random variables. Undirected graphs such as Markov random fields [43] are more suitable for energy minimization problems such as FL. Both Bayesian networks and Markov random fields can be converted into the factor graph [49] representation, which is more convenient for describing message-passing inference algorithms. In this section, we give an overview of factor graphs and two related inference algorithms: max-product and max-product linear programming.

2.2.1 Factor Graphs and the Max-Product Algorithm

A factor graph [49] is a bipartite graph, consisting of variable nodes $\mathbf{x} = \{x_1, \ldots, x_N\}$ and factor nodes $\theta = \{\theta_1, \ldots, \theta_C\}$. By convention, variables are represented by circles and factors by squares, as in the example shown in Fig. 2.5. Each factor $\theta_c$ corresponds a potential function $\theta_c(\mathbf{x}_c)$ over the subset of variables $\mathbf{x}_c$ that are its neighbors in the graph. The joint distribution described by the graph factorizes according to the factors as:

$$p(\mathbf{x}) \propto \prod_c \exp(-\theta_c(\mathbf{x}_c)) = \exp(-E(\mathbf{x}))$$ (2.21)
Given the factor graph representation of a distribution, the most common inference tasks are:

- **Marginalization**: \( p(x_i) = \sum_{x \neq x_i} p(x) \)
- **MAP inference**: \( x^* = \arg \max_x p(x) = \arg \min_x E(x) \)

When the variables are discrete, performing inference using brute-force summation or maximization is generally intractable. However, the amount of computation can be reduced by exploiting the graph structure and the distributive property of the sum (max) operator over the product (sum) operator to rearrange the order of operations. For example, for the graph in Fig. 2.5, the maximizations can be rearranged as:

\[
\max_{x_1, x_2, x_3, x_4} \ln p(x_1, x_2, x_3) = \min_{x_1, x_2, x_3} \left[ \theta_1(x_1, x_2) + \theta_2(x_2, x_3) + \theta_3(x_3, x_4) \right] \\
= \min_{x_1, x_2} \left[ \theta_1(x_1, x_2) + \min_{x_3} \left[ \theta_2(x_2, x_3) + \min_{x_4} \theta_3(x_3, x_4) \right] \right]
\]

For binary \( x_i \), rearranging the order of maximizations in the problem of Fig. 2.5 converts the problem from minimizing over \( 2^4 = 16 \) variable configurations to sequentially minimizing over the four binary variables, resulting in 8 evaluations. The *max-product* belief propagation algorithm [63] exploits this factorization to efficiently perform MAP inference; we describe its log-domain equivalent, min-sum. The iterative updates of max-product can be described as message passing operations between adjacent vertices in the factor graph. The general form of messages between a factor \( \theta_c \) and a variable \( x \in x_c \) is [10]:

\[
m_{\theta_{c} \rightarrow x}(x) \leftarrow \max_{x_c \setminus x} \left[ -\theta_c(x_c) + \sum_{x_i \in x_c \setminus x} m_{x_i \rightarrow \theta_c}(x_i) \right] \quad (2.22) \\
m_{x \rightarrow \theta}(x) \leftarrow \sum_{\theta_l \in \text{ne}(x) \setminus \theta_c} m_{\theta_l \rightarrow x}(x) \quad (2.23)
\]
where \( ne(x) \) denotes all neighbors of a vertex \( x \). The algorithm is said to converge once the message values no longer change. Upon convergence, each variable \( x \) is assigned to the value \( x^* \) that maximizes the sum of its incoming messages \( b(x) \), known as the belief.

\[
b(x) = \sum_{\theta_l \in ne(x)} m_{\theta_l \to x}(x) \quad (2.24)
\]

\[
x^* = \arg \max_x b(x) \quad (2.25)
\]

When the graphical model is a tree, max-product is guaranteed to converge to the optimal solution and can be seen as a dynamic programming algorithm. On graphs with cycles, there are no guarantees of convergence or optimality in general. Nevertheless, loopy belief propagation has empirically shown excellent performance in numerous applications [60], most notably in the area of error-correcting codes [8]. Furthermore, there exists a number of practical methods of ensuring convergence when messages oscillate between several values. One common solution is to damp the updates with a constant \( \lambda \in [0, 1) \). The damped message updates \( m_{\text{damp}} \) relate to original updates \( m_{BP} \) as

\[
m_{\text{damp}}^{(t+1)} \leftarrow \lambda m_{\text{damp}}^{(t)} + (1 - \lambda) m_{BP} \quad (2.26)
\]

### 2.2.2 Max-Product Linear Programming

The excellent empirical performance of the max-product algorithm on loopy graphs despite the lack of theoretical guarantees has led to the development of a number of related inference algorithms whose properties are better understood. Many recent such algorithms are based on the LP relaxation of the MAP inference problem [29, 45, 46, 86]. In this work, we will use the max-product linear programming (MPLP) algorithm of Globerson and Jaakkola [29] to find facility location solutions. Although the MPLP iterative message updates are quite similar to those of max-product, MPLP also has several desirable properties: it is guaranteed to converge, its objective function is monotonically
non-increasing over iterations, and it gives an upper bound on the optimal MAP solution at each iteration.

Similarly to other LP-based message-passing algorithms, MPLP is based on the following LP relaxation of the MAP inference problem $x^* = \arg \min_x \sum_c \theta_c(x)$, first introduced by [85]:

$$\text{MAP-LP: } \min_{\mu \in \mathcal{M}} \sum_c \sum_{x_c} \mu_c(x_c)\theta_c(x_c)$$

(2.27)

Here, $\mathcal{M}$ is the set of all distributions $\mu$ over configurations variables in each factor $x_c$ such that (1) each $\mu_c(x_c)$ is non-negative and normalized, and (2) any two distributions $\mu_{c_1}(x_{c_1})$ and $\mu_{c_2}(x_{c_2})$ agree on the marginal over their overlap variables $x_{c_1} \cap x_{c_2}$, as illustrated in Fig. 2.6.

In comparison to the MAP optimization problem, MAP-LP maximizes the weighted sum of potentials $\theta_c(x_c)$ summed over all configurations of $x_c$, and the maximization is performed over the weights $\mu$. This is an interesting LP relaxation in which the original variables $x$ remain binary. As in all LP relaxations, the MAP-LP solution is an upper bound on the original problem and MAP-optimal when the solution $\mu^*$ is integral.

The MPLP iterative updates correspond to block co-ordinate descent steps in the dual LP, augmented with some redundant variables. We omit the details for now, as we will show them for the specific case of UFL. Let $N_c$ denote the number of all factor potentials at path length two from $\theta_c(x)$ in the factor graph (i.e. those factors $\theta_c$ that overlap with $\theta_c$ on some set of variables). The general form of the MPLP message updates is:

$$m_{\theta_c \rightarrow x}(x) \leftarrow -\left(1 - \frac{1}{N_c}\right)m_{x \rightarrow \theta} + \frac{1}{N_c} \max_{x_c \setminus x} [-\theta(x_c) + \sum_{x_i \in x_c \setminus x} m_{x_i \rightarrow \theta_c}(x_i)]$$

(2.28)

$$m_{x \rightarrow \theta}(x) \leftarrow \sum_{\theta_i \in \text{ne}(x) \setminus \theta_c} m_{\theta_i \rightarrow x}(x)$$

(2.29)
As in the regular max-product algorithm, once the messages converge, beliefs \( b(x) \) are calculated by summing the incoming messages for each variable \( x \). Variables are assigned to the value that maximizes their beliefs as \( x^* = \arg \max_x b(x) \).

If all beliefs have unique maximizers, the obtained solution \( x^* \) is guaranteed to be optimal [29]. When the beliefs are maximal for several variable settings (e.g. \( b(1) = b(0) \) for a binary \( x \)), we need to decide how to assign variables. It has been shown that in some special graphs, such as those with binary variables and submodular pairwise potentials, it is still possible to find the optimal \( x^* \) in polynomial time [70]. However, as expected, this is no longer the case in graphical models corresponding to \( NP \)-hard problems such as facility location.

### 2.3 Affinity Propagation

Affinity Propagation (AP) is an exemplar-based clustering algorithm whose objective corresponds to a special case of UFL, where \( C = F \). Let \( x_{ij} \) be a binary random variable indicating whether point \( j \) is \( i \)'s exemplar; AP finds solutions to the following integer program:

\[
\begin{align*}
\max \quad & x \sum_i \sum_{j \neq i} s_{ij} x_{ij} + \sum_j p_j x_{jj} \\
\text{s.t.} \quad & \sum_j x_{ij} = 1 \quad \forall i, j \\
& x_{jj} - x_{ij} \geq 0 \quad \forall i, j \\
& x_{ij} \in \{0, 1\}
\end{align*}
\]

This is an instance of the UFL with costs set to \( f_j = -p_j, c_{jj} = 0 \) and \( c_{ij} = -s_{ij}, i \neq j \). The iterative updates of AP correspond to the max-product algorithm on the factor graph shown in Fig. 2.7, where the factors are defined as:
Figure 2.6: An illustration of the LP relaxation of the MAP inference problem. For each potential $\theta_c(x_c)$, we introduce a distribution $\mu_c(x_c)$ over all configurations of $x_c$ and perform the optimization with respect to $\mu$. The distributions $\mu$ are constrained to be positive, normalized, and to agree on intersection variable sets. In the figure, $\mu_1(x_1, x_2)$ and $\mu_2(x_2, x_3)$ agree on the marginal $\mu(x_2)$, while $\mu_2(x_2, x_3)$ and $\mu_3(x_3, x_4)$ agree on the marginal $\mu(x_3)$. When the LP solution $\mu^*$ is integral, the relaxation is tight and the corresponding $x^*$ is optimal.

\[
S_{ij}(x_{ij}) = \begin{cases} -s_{ij}x_{ij}, & j \neq i \\ -p_jx_{jj}, & j = i \end{cases}
\]

(2.34)

\[
\theta_j^F(x_{1j}, \ldots, x_{N_j}) = \begin{cases} 0, & x_{jj} \geq x_{ij} \forall i \\ \infty, & \text{otherwise} \end{cases}
\]

(2.35)

\[
\theta_i^C(x_{i1}, \ldots, x_{iN}) = \begin{cases} 0, & \sum_j x_{ij} = 1 \\ \infty, & \text{otherwise} \end{cases}
\]

(2.36)
Factors $S_{ij}$ reflect the optimization objective, factors $\theta_i^C$ enforce the constraint that each point must be assigned to exactly one exemplar, and factors $\theta_j^F$ ensure that if a point is an exemplar, it is also its own exemplar.

AP has shown excellent empirical performance in comparison to related clustering algorithms, taking minutes to find solutions that take days for $k$-medians, and out-performing hierarchical agglomerative clustering [20]. The success of AP motivates both using the FL formulation to tackle problems in machine learning and the graphical model approach.
Chapter 3

Max-Product Algorithm for Facility Location Problems

In this chapter, we show factor graphs and corresponding message-passing inference algorithms for different facility location variants. For each problem, we construct a factor graph whose potentials $\theta_c(x_c)$ reflect the problem costs and constraints. Minimizing cost $E(x) = \sum_c \theta_c(x_c)$ then corresponds to finding the mode of the distribution described by the graph $P(x) = \exp(-E(x))$, or performing MAP inference. We find solutions by running the max-product belief propagation algorithm on each factor graph. As all factor graphs are loopy, max-product is not guaranteed to converge to the optimal solution, and can only be seen as a heuristic approach.

In the context of exemplar-based clustering, the developed algorithms can be seen as a generalization of the Affinity Propagation (AP) algorithm. We end the chapter by showing some applications of the developed algorithms to image clustering and video summarization.
3.1 Uncapacitated Facility Location

Recall that the UFL problem can be written as the following ILP:

$$\min_{x} \sum_{i} \sum_{j} c_{ij}x_{ij} + \sum_{j} f_{j}y_{j}$$  \hspace{1cm} (3.1)$$

s.t.  \hspace{1cm} \sum_{j} x_{ij} = 1 \quad \forall i \in C$$  \hspace{1cm} (3.2)$$

$$y_{j} \geq x_{ij} \quad \forall i \in C, j \in F$$  \hspace{1cm} (3.3)$$

$$x_{ij}, y_{j} \in \{0, 1\} \quad \forall i \in C, j \in F$$  \hspace{1cm} (3.4)$$

where variables $x_{ij}$ indicate whether a customer $i$ is connected to facility $j$, and variables $y_{j}$ indicate whether facility $j$ is open. The corresponding factor graph representation is shown in Fig. 3.1, where the factor potentials are as follows:

$$F_{j}(y_{j}) = f_{j}y_{j}$$  \hspace{1cm} (3.5)$$

$$C_{ij}(x_{ij}) = c_{ij}x_{ij}$$  \hspace{1cm} (3.6)$$

$$\theta^{F}_{j}(x_{j}, y_{j}) = \begin{cases} 0, & y_{j} \geq \max_{i} x_{ij} \\ \infty, & \text{otherwise.} \end{cases}$$  \hspace{1cm} (3.7)$$

$$\theta^{C}_{i}(x_{i}) = \begin{cases} 0, & \sum_{j} x_{ij} = 1 \\ \infty, & \text{otherwise.} \end{cases}$$  \hspace{1cm} (3.8)$$

where we have used the notation $x_{j} = \{x_{1j}, \ldots, x_{Nj}\}$ and $x_{i} = \{x_{i1}, \ldots, x_{iM}\}$. The single-node factors $F_{j}$ and $C_{ij}$ reflect the optimization objective, while the $\theta^{C}_{i}(x_{i})$ and $\theta^{F}_{j}(x_{j}, y_{j})$ enforce the constraints 3.2 and 3.3.

The max-product algorithm corresponding to the factor graph in Fig. 3.1 is provided in Alg. 1, where we have followed the message naming convention in Fig. 3.2. We use notation $\alpha$, $\eta$, $c$, $b^{x}$ and $x$ to represent matrices $[\alpha_{ij}]_{N \times M}$, $[\eta_{ij}]_{N \times M}$, $[b^{x}_{ij}]_{N \times M}$ and $[x_{ij}]_{N \times M}$, respectively.
Figure 3.1: UFL factor graph

Figure 3.2: UFL message naming convention
respectively. Similarly, we use $\nu$, $f$, $b^y$ and $y$ to represent $M \times 1$ vectors with entries $\nu_j$, $f_j$, $b^y_j$ and $y_j$, respectively. Alg. 1 also incorporates a few simplifications:

- The algorithm is expressed in terms of factor-to-variable messages only, as this is sufficient to calculate beliefs and make variable assignments.

- As all messages are functions of binary random variables, it suffices to only keep track of the difference between the two message values, $m \equiv m(1) - m(0)$, or equivalently to set all $m(0) = 0$.

- We only iteratively update messages $\alpha_{ij}$ and $\eta_{ij}$. Messages from singleton factors $F_j$ and $C_{ij}$ do not change over iterations, and messages $\nu_j$ are only required at convergence.

Upon message convergence, beliefs $b^x_{ij}$ and $b^y_j$ are computed by summing the incoming messages for variables $x_{ij}$ and $y_j$. Each variable is assigned a value of 1 or 0 according to whether its belief is positive or negative, respectively.

As the factor graph in Fig. 3.1 contains loops, it is often necessary to perform damped message updates for reasons of computational stability and convergence. For a constant $\lambda \in [0, 1)$, we can use the following damped updates in Alg. 1:

$$
\eta_{ij} \leftarrow -(1 - \lambda) \max_{k \neq j}(\alpha_{ik} - c_{ik}) + \lambda \eta_{ij} \quad (3.9)
$$

$$
\alpha_{ij} \leftarrow (1 - \lambda) \min[0, -f_j + \sum_{k \neq i} \max(0, \eta_{kj} - c_{kj})] + \lambda \alpha_{ij} \quad (3.10)
$$

In the case where the customer set $\mathcal{C}$ and the facility set $\mathcal{F}$ are the same, these updates correspond to the Affinity Propagation algorithm.
Algorithm 1 UFL MAX-PRODUCT

Initialize $\eta \leftarrow 0, \alpha \leftarrow 0, \nu \leftarrow 0$

Iteratively update messages:

repeat
  $\eta$-UPDATE
  $\alpha$-UPDATE
until convergence

$\nu$-UPDATE

Compute beliefs:

$b^x = \alpha + \eta - c$

$b^y = \nu - f$

Assign variables:

$x^* = I[b^x \geq 0]$

$y^* = I[b^y \geq 0]$

---

Procedure $\eta$-UPDATE
for all $i, j$ do
  $\eta_{ij} \leftarrow -\max_{k \neq j}(\alpha_{ik} - c_{ik})$
end for

---

Procedure $\alpha$-UPDATE:
for all $i, j$ do
  $\alpha_{ij} \leftarrow \min[0, -f_j + \sum_{k \neq i} \max(0, \eta_{kj} - c_{kj})]$
end for

---

Procedure $\nu$-UPDATE:
for all $j$ do
  $\nu_j \leftarrow \sum_k \max(0, \eta_{kj} - c_{kj})$
end for
3.2 Capacitated Facility Location

In capacitated facility location (CFL) problems, the facility cost is a monotonically non-decreasing function of the number of connected customers. Letting $u_j = \sum_j x_{ij}$ be the number of customers connected to facility $j$, some examples of CFL facility costs are:

- $F_j(u_j) = f_j I[u_j > 0] + \infty I[u_j > M_j]$. This is hard-capacitated facility location, where at most $M_j$ customers can be served by facility $j$.

- $F_j(u_j) = f_j \lceil u_j / M_j \rceil$. In this problem version, an unlimited number of facilities of capacity $M_j$ can be opened at location $j$.

- $F_j(u_j) = f_j I[u_j > 0] \ln(u_j)$. This is the facility cost corresponding to multiple model selection under the BIC criterion, where $f_j$ is the number of parameters of model $j$.

- $F_j(u_j) = f_j I[u_j > 0] + \ln \Gamma(\frac{u_j}{M_j})$. This cost was used in [16] to incorporate a Dirichlet prior on cluster size in a generative model framework of exemplar-based clustering.

We represent CFL problems using the factor graph of Fig. 3.3. In comparison with the UFL factor graph of Fig. 3.1, the binary variables $y_j$ are replaced by integer variables $u_j \in \{0, \ldots, N\}$ and the factor potentials $\theta_j^F$ are redefined to:

$$
\theta_j^F(x, u_j) = \begin{cases} 
0, & u_j = \sum_i x_{ij} \\
\infty, & \text{otherwise.}
\end{cases}
$$

(3.11)
Chapter 3. Max-Product Algorithm for Facility Location Problems

Figure 3.3: Factor graph representation of CFL

Figure 3.4: Message naming convention for CFL
Chapter 3. Max-Product Algorithm for Facility Location Problems

The $\eta_{ij}$ updates in CFL are the same as in UFL, while the $\nu_j(u_j)$ and $\alpha_{ij}(x_{ij})$ updates change to:

$$\nu_j(u_j) \leftarrow \max_{x_j, \sum_k x_{kj} = u_j} \sum_k x_{kj} (\eta_{kj} - c_{kj})$$  \hspace{1cm} (3.12)

$$\alpha_{ij}(1) \leftarrow \max_{x_{-ij}} [F_j(1 + \sum_{k \neq i} x_{kj}) + \sum_{k \neq i} x_{kj}(\eta_{kj} - c_{kj})] + \sum_{k \neq i} \eta_{kj}(0)$$  \hspace{1cm} (3.13)

$$\alpha_{ij}(0) \leftarrow \max [0, \max_{x_{-ij}} (F_j(\sum_{k \neq i} x_{kj}) + \sum_{k \neq i} x_{kj}(\eta_{kj} - c_{kj}))] + \sum_{k \neq i} \eta_{kj}(0)$$  \hspace{1cm} (3.14)

where we have used the notation $x_{-ij} = x_{ij} \setminus x_{ij}$. We will also use $\nu_j$ to denote the $(N + 1) \times 1$ message vector for each of the possible values of the message argument $u_j = \sum_{i=1}^N x_{ij}$.

The $\nu_j$ and $\alpha_{ij}$ updates now involve constrained maximization over $N$ binary variables. However, they can be computed efficiently in $O(N \log N)$ time for monotonically non-increasing $F_j(u_j)$, by sorting $(\eta_{kj} - c_{kj})$ in descending order for each facility $j$ and computing the cumulative sum. In general, computing messages for any potential based on the cardinality of its neighboring variables is tractable [32,75]. The CFL updates are summarized in Algorithm 2.

3.3 $k$-Facilities

In the $k$-facilities problem, we incorporate an additional potential on the number of open facilities in the solution. One way of accomplishing this is through the graphical model shown in Fig. 3.5, which we call $k$-UFL. It incorporates a hidden Markov model (HMM), where $z_j, j = 0, \ldots, M$ are hidden variables and $y_j, j = 1, \ldots, M$ can be thought of as noisy observations. Each hidden variable $z_j$ has $M + 1$ possible states, and in effect counts the number of open facilities in the $\{1, \ldots, j\}$ subset, i.e. $z_j = \sum_{k=1}^j y_k$. This is
Algorithm 2 CFL MAX-PRODUCT

Initialize $\eta \leftarrow 0, \alpha \leftarrow 0, \nu \leftarrow 0$

Compute messages:
repeat
  $\eta$-UPDATE
  $(\alpha, \nu)$-UPDATE
until convergence

Compute beliefs:

$b^x = \alpha + \eta - c$
$b^y = \nu - f$

Assign variables:
$x^* = I[b^x \geq 0]$
$y^* = I[b^y \geq 0]$

Procedure $\eta$-UPDATE:
for all $i, j$ do
  $\eta_{ij} \leftarrow \max_{k \neq j} (\alpha_{ik} - c_{ik})$
end for

Procedure $(\alpha, \nu)$-UPDATE:
for $j = 1 : M$ do
  $(\rho, Index_{\rho}) = $ SORT-DESCEND $([\eta_{ij} - c_{ij}])$
  $\nu_j \leftarrow CUMSUM(\rho)$
  for $i = 1 : N$ do
    $S = \nu_j - (\eta_{ij} - c_{ij}) CUMSUM(I[Index_{\rho} == i])$
    $m_1 = \arg \max_m (S_m + F_j(1 + m))$
    $m_0 = \arg \max_m (S_m + F_j(m))$
    $\alpha_{ij} \leftarrow \min(F_j(1 + m_1) + S_{m_1}, F_j(1 + m_1) + S_{m_1} - F_j(m_0) - S_{m_0})$
  end for
end for
enforced by setting $z_0 = 0$, and setting the factor potentials $G_j$, $j = 1, \ldots, M$ to:

$$G_j(y_j, z_{j-1}, z_j) = \begin{cases} 
0, & z_j = z_{j-1} + y_j \\
\infty, & \text{otherwise}. 
\end{cases} \quad (3.15)$$

The hidden variable $z_M = \sum_{k=1}^M y_k$ corresponds to the total number of open facilities and an arbitrary potential on $z_M$ is incorporated through the factor $G_{M+1}$. Computing max-product message updates for the HMM part of the graphical model is straightforward, and well-known in literature as the Viterbi algorithm. We list the factor-to-variable messages in Fig. 3.6 and their updates in Algorithm 3.
Algorithm 3 k-FACILITIES MAX-PRODUCT

Initialize $\eta \leftarrow 0, \alpha \leftarrow 0, \nu \leftarrow 0, \gamma \leftarrow 0, a \leftarrow 0, b \leftarrow 0$
Compute messages:
repeat
  $\eta$-UPDATE
  $\alpha$-UPDATE
  $\nu$-UPDATE
  $\gamma$-UPDATE
until convergence
Compute beliefs:
$b^x = \alpha + \eta - c$
$b^y = \nu - f$
Assign variables:
$x^* = I[b^x \geq 0]$
$y^* = I[b^y \geq 0]$

Procedure $\eta$-UPDATE:
for all $i, j$ do
  $\eta_{ij} \leftarrow - \max_{k \neq j} (\alpha_{ik} - c_{ik})$
end for

Procedure $\alpha$-UPDATE:
for all $i, j$ do
  $\alpha_{ij} \leftarrow \min[0, -f_j + \sum_{k \neq i} \max(0, \eta_{kj} - c_{kj})]$
end for

Procedure $\nu$-UPDATE:
for all $j$ do
  $\nu_j \leftarrow \sum_k \max(0, \nu_{kj} - c_{kj})$
end for

Procedure $(a, b)$-UPDATE:
Initialize: $z_0 = 0$, $b_M(z_M) = G_{M+1}(z_M)$
for $j = 1 : M$ do
  $a_j(z_j) \leftarrow \max[a_{j-1}(z_j), a_{j-1}(z_j - 1) + \nu_j - f_j]$
end for
for $j = M : 1$ do
  $b_{j-1}(z_{j-1}) \leftarrow \max[b_j(z_{j-1}), b_j(z_{j-1} + 1) + \nu_j - f_j]$
end for

Procedure $\gamma$-UPDATE:
for $j = 1 : M$ do
  $\gamma_j \leftarrow \max_z [a_{j-1}(z - 1) + b_j(z)] - \max_z [a_{j-1}(z) + b_j(z)]$
end for
3.4 Clustering Applications of FL Algorithms

3.4.1 \textit{k}-AP: Affinity Propagation With an Arbitrary Prior on the Number of Exemplars

One limitation of Affinity Propagation is that a prior belief or constraint on the number of cluster exemplars cannot be specified directly. Such a prior is intuitive and desirable in many applications. For example, if we want to segment a natural image by clustering pixels according to appearance and spatial coherence, we expect to see between two and ten clusters, corresponding to the objects present and the background, even if the image contains over 1M pixels. In other applications, we may have a range constraint on the number of clusters. In \textit{video abstraction} the goal is to summarize a video sequence via a set of representative keyframes, and a common approach is to do so by clustering frames. In this case, we need the set of selected frames to be small regardless of the amount of variation the video contains.

The number of clusters obtained by AP is primarily governed by the regularization term $\sum_j p_j y_j$, where $p_j$ are known as exemplar preferences and correspond to negative
facility costs, \( p_j = -f_j \). When all points are equally likely to be exemplars, this term is linear in the number of exemplars and equals \( p \sum_j y_j \). \( p \) can be thought of as a “control knob” for the number of clusters, as illustrated in Fig.3.7. However, there exists no principled way of setting the preference range for a given number of exemplars, and we may need to run the algorithm over several settings to get desired results.

![Figure 3.7: Illustrating the relationship between preferences and clustering granularity in AP on a toy data set, where similarities set to negative squared Euclidean distance and all preferences are equal. The top figure shows clusters obtained by AP for several preference settings, while the bottom figure plots the number of clusters vs. preference.](image)

We can apply ideas from Section 3.3 to incorporate an arbitrary prior belief and/or constraint on the number of exemplars. This can also be interpreted as a regularization term that is non-linear in the number of exemplars. We call the extended AP algorithm \( k \)-AP, and demonstrate its advantages over regular AP on synthetic data sets and on the task of video abstraction.

### 3.4.2 Synthetic Data

One scenario in which \( k \)-AP has an advantage over regular AP is when the number of underlying data clusters is known to be exactly \( k \). We illustrate this on toy data sets in Figures 3.8 and 3.9, where pairwise similarities are set to negative squared Euclidean distance and all preferences are equal. Obtaining the specified number of clusters with regular AP requires a search over the preference setting. On the other hand, in \( k \)-AP
the number of clusters and to a large extent the cluster membership are unaffected by varying the preferences.

### 3.4.3 Video Abstraction

We now apply $k$-AP to the problem of video abstraction, where the goal is to summarize a video sequence via a set of salient keyframes. Such abstracts are also known as static storyboards, and are designed mainly to enable efficient user browsing of video databases. They are especially useful when combined with video search engines or content-based retrieval, in a manner analogous to Internet search engines and textual webpage abstracts. Video abstracts also allow users easier access to semantically relevant frames in a single video sequence, and can greatly reduce the computational overhead in video content retrieval and analysis.

There has been much work in various types of video abstraction recently; a comprehensive summary is provided in [79]. However, keyframe selection is still largely in the research phase - most current video search engines such as Yahoo, Alta Vista, YouTube and Google Video currently represent videos using a single frame and text. One exception is the Open-Video Archive, where users can view a static storyboard of about 10-30 thumbnail images of each video.
Figure 3.9: Clustering synthetic data via regular AP and $k$-AP over different preference settings. Clusterings with 2 or 6 exemplars can be obtained by either varying the preference setting in AP, or enforcing $k = 2$ or $k = 6$ in the $k$-AP prior.

Keyframe selection involves finding data exemplars in a very high dimensional space, making it a particularly suitable problem for Affinity Propagation. However, as even short videos can contain many scene and shot changes, AP can potentially find a very large number of clusters, which is inefficient for user browsing. We limit the number of clusters using $k$-AP, and compare the results obtained by the two algorithms.

To initialize AP, we set pairwise similarities to the negative squared Euclidean distance between frame features. The frame feature we use is the gist descriptor [62] of R, G and B channels downsampled to $128 \times 96$ pixels, with overall dimensionality reduced to 40 using PCA. We arbitrarily set all preferences to be very low: 15 times the median similarity (for negative similarities). For $k$-AP, we use a discrete uniform prior on $\{1, \ldots, k\}$ with $k \in \{5, 7, 9\}$.

We demonstrate the storyboard results on two Open Video examples containing excerpts from the NASA 25th Anniversary Show in Figures 3.10 and 3.11. The figures contain the results obtained by $k$-AP, regular AP, and the storyboards currently available on Open Video. Both AP and $k$-AP discover frames similar to those on Open Video, but remove much of the redundancy. The $k$-AP exemplars are typically a less redundant
subset of the AP exemplars.

Figure 3.10: NASA 25th Anniversary Show, Segment 01. The figure shows video summaries obtained via $k$-AP with $k \in \{5, 7, 9\}$, regular AP, and the OpenVideo storyboard.

Figure 3.11: NASA 25th Anniversary Show, Segment 07. The figure shows video summaries obtained via $k$-AP with $k \in \{5, 7, 9\}$, regular AP, and the OpenVideo storyboard.

### 3.5 Discussion

In this chapter, we presented the graphical models and corresponding max-product algorithms for a number of facility location problem variants. For the problem of exemplar-based clustering, these algorithms correspond to a generalization of the Affinity Propagation algorithm that incorporates prior beliefs and/or constraints on the number of clusters and their size. As the graphical models for all problem variants contain loops, there are in general no guarantees on the optimality of their solutions, and they can be
seen as efficient heuristics. In the next chapter, we describe a related message passing approach which additionally has a $\rho$-approximation guarantee for metric UFL.
Chapter 4

Max-Product Linear Programming
Algorithm for UFL

Polynomial-time approximation algorithms are an important and well-researched class of algorithms for metric UFL. Typically, these algorithms are based on the standard LP relaxation of the problem where integrality constraints $x_{ij} \in \{0,1\}$ are replaced by non-negativity constraints $x_{ij} \geq 0$. In this chapter, we describe a novel approximation algorithm for metric UFL that is based on the MAP-LP relaxation and message passing.

We first perform MAP inference using the max-product linear programming (MPLP) algorithm [29], one of many recent message passing algorithms based on the MAP-LP relaxation. At convergence, MPLP either finds the globally optimal solution, or leaves a subset of variables unassigned. For the later case, we describe a greedy variable “decoding” algorithm with a 3-approximation guarantee for metric UFL. We also demonstrate the empirical usefulness of the approach, comparing its solutions to a randomized variable assignment.
4.1 MAP-LP Relaxation and MPLP Updates

In this chapter, we use a slightly simplified UFL factor graph shown in Fig. 4.1, where we have removed the redundant $y_j$ variables and incorporated facility costs into factors $\theta_j^F$. The potentials are now defined as:

$$C_{ij}(x_{ij}) = c_{ij}x_{ij} \quad (4.1)$$

$$\theta_j^F(x_{ij}) = \begin{cases} f_j, & \sum_i x_{ij} > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (4.2)$$

$$\theta_i^C(x_i) = \begin{cases} 0, & \sum_j x_{ij} = 1 \\ \infty, & \text{otherwise.} \end{cases} \quad (4.3)$$

The MAP-LP relaxation for the UFL is:

$$\max_{\mu \in \mathcal{M}} \mu \cdot \theta = \sum_{i,j} \sum_{x_{ij}} \mu_{ij}(x_{ij})(c_{ij}x_{ij}) + \sum_j \sum_{x_{ij}} \mu_j^F(x_{ij})\theta_j^F(x_{ij}) + \sum_i \sum_{x_i} \mu_i^C(x_i)\theta_i^C(x_i) \quad (4.4)$$

$$\mathcal{M} = \left\{ \begin{array}{l}
\mu \geq 0 \\
\sum_{x_{ij}} \mu_{ij}(x_{ij}) = 1 \quad \forall i \in C, j \in F \\
\sum_{x_{ij}} \mu_j^F(x_{ij}) = \mu_{ij}(x_{ij}) \quad \forall i \in C, j \in F \\
\sum_{x_i} \mu_i^C(x_i) = \mu_{ij}(x_{ij}) \quad \forall i \in C, j \in F
\end{array} \right\}$$

Similarly to max-product belief propagation, MPLP can be described in terms of iteratively exchanged messages between neighboring variables in the graphical model. The sum of all messages a variable receives corresponds to its belief $b_{ij}(x_{ij})$ that it takes on a particular value. However, MPLP messages and beliefs also correspond to variables
Chapter 4. Max-Product Linear Programming Algorithm for UFL

Figure 4.1: UFL factor graph
in a particular formulation of MAP-LP dual problem, and their updates perform block coordinate ascent in this dual. Following [29], we express the dual LP in terms of messages and beliefs, providing the details in Section 4.4:

\[
\begin{align*}
\min & \quad g(\beta, \alpha, \eta) = \sum_{ij} \max_{x_{ij}} b_{ij}(x_{ij}) \\
\text{s.t.} & \quad b_{ij}(x_{ij}) = -c_{ij}x_{ij} + \alpha_{ij}(x_{ij}) + \eta_{ij}(x_{ij}) \\
& \quad \alpha_{ij}(x_{ij}) = \max_{x_{-ij}} \beta_{F}^{i}(x_{-j}) \\
& \quad \eta_{ij}(x_{ij}) = \max_{x_{i-}} \beta_{C}^{j}(x_{i-}) \quad (4.6) \\
& \quad \sum_{i} \beta_{F}^{i}(x_{-j}) = \theta_{F}^{j}(x_{-j}) \quad \forall j, x_{i} \\
& \quad \sum_{j} \beta_{C}^{j}(x_{i-}) = \theta_{C}^{i}(x_{i-}) \quad \forall i, x_{i}.
\end{align*}
\]

MPLP iterative updates correspond to performing block co-ordinate ascent in the dual variables $\beta$, obtained by optimizing over either $\beta_{C}^{j}(x_{i-})$ or $\beta_{F}^{i}(x_{-j})$, while holding all other variables constant. In practice, it suffices to only keep track of “message” variables $\eta_{ij}(x_{ij})$ and $\alpha_{ij}(x_{ij})$. In Section 4.4, we show that the differential updates $\eta_{ij} = \eta_{ij}(1) - \eta_{ij}(0)$ and $\alpha_{ij} = \alpha_{ij}(1) - \alpha_{ij}(0)$ of these messages are:

**Procedure $\eta$-UPDATE:**

\begin{algorithm}
\begin{algorithmic}
\FORALL {$i, j$}
\STATE $\eta_{ij} \leftarrow -\frac{1}{M} \max_{k \neq j} (\alpha_{ik} - c_{ik}) - \frac{M-1}{M} (\alpha_{ij} - c_{ij})$
\ENDFOR
\end{algorithmic}
\end{algorithm}

**Procedure $\alpha$-UPDATE:**

\begin{algorithm}
\begin{algorithmic}
\FORALL {$i, j$}
\STATE $\alpha_{ij} \leftarrow \frac{1}{N} \min \left[ 0, -f_{j} + \sum_{k \neq i} \max(0, \eta_{kj} - c_{kj}) \right] - \frac{N-1}{N} (\eta_{ij} - c_{ij})$
\ENDFOR
\end{algorithmic}
\end{algorithm}

At MPLP convergence, variables are assigned to values that maximize their beliefs, as in the standard max-product algorithm. If this assignment is unique, the MAP-LP relaxation is tight, and the solution is globally optimal [29]. However, it is also possible to
have a non-unique solution at convergence, with a subset of variables having equal beliefs
for different values, i.e. $b(0) = b(1)$. We will describe a greedy algorithm for assigning
these variables that guarantees to produce solutions within a factor 3 of optimal for
metric UFL instances.

4.2 Complementary Slackness and a 3-Approximation
Algorithm

4.2.1 MAP-LP Complementary Slackness

Our approach to decoding MPLP solutions is based on the MAP-LP complementary
slackness conditions. These conditions always hold for a pair of solutions $\mu, \beta$ that are
optimal for the primal and dual LP, and can be written as:

$$
\sum_{x_{ij}} \mu_{ij}(x_{ij}) \left[ b_{ij}(x_{ij}) - \max_{x_{ij}} b_{ij}(x_{ij}) \right] = 0 \quad (4.8)
$$

$$
\sum_{x_{i}} \mu_{i}^{Cj}(x_{i}) \left[ \beta_{i}^{Cj}(x_{i}) - \max_{x_{i-j}} \beta_{i}^{Cj}(x_{i}) \right] = 0 \quad (4.9)
$$

$$
\sum_{x_{j}} \mu_{j}^{Fi}(x_{j}) \left[ \beta_{j}^{Fi}(x_{j}) - \max_{x_{i-j}} \beta_{j}^{Fi}(x_{j}) \right] = 0 \quad (4.10)
$$

When the LP relaxation is tight, these conditions also hold for the integral solution
$x^* = \mu^*$, and can simply be expressed as:

(CS 1) Each customer $i$ is connected to exactly one facility $j$ for which $b_{ij} \geq 0$.

(CS 2) An open facility $j$ serves all customers $i$ for which $b_{ij} \geq 0$.

These conditions are illustrated in Fig 4.2. When the LP relaxation is not tight, any
feasible integral solution $x^*$ that maximizes beliefs will satisfy (CS 1), but not (CS 2).
Our decoding approach will be to greedily construct solutions that always satisfy (CS 2), but not necessarily (CS 1).

### 4.2.2 A 3-Approximation Algorithm for UFL

The pseudocode of our decoding algorithm is given in Alg. 4, and its steps are illustrated in Fig. 4.3. We start by constructing a bipartite support graph $G = (C, F, E)$ whose vertices $C$ and $F$ are customers and facilities, and edges $(i,j)$ connect each customer-facility pair for which $b_{ij} \geq 0$. We also associate a weight $\eta_{ij}$ with each edge $(i,j)$, where $\eta_{ij}$ are the values of a subset of MPLP messages (dual variables) at convergence.

We open facilities one by one, greedily choosing the facility with the minimum-weight edge. Whenever a facility is opened, all of its neighbor customers are assigned to it, ensuring that CS(2) conditions are satisfied. All facilities two edges away from the opened facility are then removed from the graph, as they can no longer be opened such that CS(2) holds. When no more facilities can be opened, each customer is either 1 or 3 edges away (in the original graph) from an open facility, to which it gets assigned.

We note that the greedy solution will be different from any MPLP solution when the LP relaxation is loose. An arbitrary belief-maximizing solution will always satisfy CS(1) but not CS(2), unless the LP relaxation is tight. On the other hand, a greedy solution will always satisfy CS(2) but not CS(1); beliefs will not be maximized for customers assigned to facilities 3 edges away.

**Algorithm 4 3-APPROXIMATION DECODING ALGORITHM**

```
initialize $G = (C, F, E)$
while $E$ is not empty do
  (A) $(i, j) \leftarrow$ edge with min weight $\eta_{ij}$
  (B) open facility $\hat{j}$ and connect all its neighbors in $G$
  (C) remove all facilities 2 edges from $\hat{j}$ from $G$
  remove $\hat{j}$ and its connected customers from $G$
end while
(D) assign remaining customers in $C_0$ to the closest open facility
```
Figure 4.2: Top: An MPLP fixed point with unresolved variables. Middle: an integral solution that satisfies (CS 1) but not (CS 2). Bottom: an integral solution that satisfies (CS 2), but not (CS 1).
Figure 4.3: Illustration of the greedy decoding algorithm. (A) Select the min-weight edge. (B) Open the corresponding facility and connect all customers. (C) Remove all facilities 2 edges away from the opened facility. (D) Once no facilities are available, connect remaining customers to the closest open facility.
The greedy Alg. 4 produces integral solutions whose cost $E(x^*)$ is 3 times that of the dual lower bound $-g(\beta, \alpha, \eta)$, and hence within a factor 3 of optimal, for metric UFL. The proof sketch is as follows. The integral cost of customers at path length 1 is equal to their cost in the dual, as the corresponding variables satisfy all complementary slackness conditions. The integral cost of customers at path length 3 is at most 3 times that of the dual. To show this, we need two properties of MPLP fixed points, which we show in detail in Appendix A:

- For each customer $i$ and all facilities $j$ such that $b_{ij} = 0$, $\eta_{ij} > c_{ij}$ and all $\eta_{ij}$ messages are equal. We will denote these messages by $\hat{\eta}_i$.

- The contribution of tied customers $C_0 = \{ i \in C | \exists j \in F \text{ s.t. } b_{ij} = 0 \}$ to the dual objective can be simplified to $-g(\beta, \alpha, \eta) = \sum_i \hat{\eta}_i$

When a customer $i \in C_0$ is assigned to a facility $j$ at path length 3, the cost contribution changes from $\hat{\eta}_i$ in the dual LP to $c_{ij}$ in the primal IP, and we can show that $c_{ij} < 3\hat{\eta}_i$. For example, for customer 2 and facility 1 in Fig. 4.3,

\[
\begin{align*}
c_{21} &\leq c_{11} + c_{22} + c_{12} \quad \text{(triangle inequality)} \\
&\leq \eta_{11} + \eta_{22} + \eta_{12} \quad \text{(} \eta_{ij} \geq c_{ij} \ \forall (i, j) \in E\text{)} \\
&\leq 3 \max(\hat{\eta}_1, \hat{\eta}_2) \quad \text{(} \eta_{ij} = \hat{\eta}_i \ \forall (i, j) \in E\text{)} \\
&= 3\hat{\eta}_2 \quad \text{(greedy order)}
\end{align*}
\]

To summarize, the integral cost of customers at path length 1 is equal to the dual, and the integral cost of customers at path length 3 is at most 3 times that of the dual lower bound. It follows that the solution cost $E(x^*)$ is at most 3 times that of the dual $g^0(\beta, \alpha, \eta)$, and hence within a factor 3 of the optimal solution.
4.3 Experiments

In this section, we empirically evaluate the 3-approximation decoding algorithm on metric UFL data, generated by randomly uniformly sampling $N$ points in a unit square, setting connection costs to Euclidean distances, and setting all facility costs to either $\sqrt{N}/10$, $\sqrt{N}/100$, or $\sqrt{N}/1000$, as proposed by [1].

We perform inference using MPLP and resolve ties using (1) greedy Alg. 4, and (2) an arbitrarily belief-maximizing assignment. The results are shown in Fig. 4.4, where the error measures the percentage by which the solution cost exceeds the LP lower bound. In all cases but one, Alg. 4 results in equal or lower cost than belief maximization. An intuitive reason behind the improvement in performance is that satisfying (CS 2) requires that each facility serves enough customers to justify opening costs. Arbitrarily satisfying only (CS 1) may incur more cost due to too many facilities being open, as illustrated in Fig. 4.5.

4.4 Discussion

In this chapter, we described a new approximation algorithm for the UFL based on the max-product linear programming algorithm. In addition to the 3-approximation guarantee, our greedy algorithm also improves MPLP performance on a number of UFL instances. Overall, the approach offers more general insights into obtaining integral solutions from MPLP fixed points. Although in MPLP variables are typically assigned by maximizing beliefs (following the tradition of standard max-product), this simply corresponds to satisfying one particular subset of complementary slackness conditions for the MAP-LP. As we have demonstrated in this chapter, choosing to satisfy a different subset may prove empirically beneficial for some problems.
Figure 4.4: Experimental results comparing belief decoding and greedy decoding on synthetic metric clustering problems. Connection costs $c_{ij}$ are set to pairwise Euclidean distances between $N$ points randomly generated in a unit square. Facility costs are set to $\sqrt{N}/10$ (top), $\sqrt{N}/100$ (middle), and $\sqrt{N}/1000$ (bottom). Error is measured as the percentage by which the obtained cost exceeds the LP lower bound.
Figure 4.5: Top: a MPLP fixed point with unresolved variables. Middle: arbitrarily assigning variables by maximizing beliefs can lead to bad solutions, and potentially all facilities being open. Bottom: greedy Alg. 4 opens facilities conservatively, but may not maximize all customer beliefs.
Appendix A

We show details on constructing the dual of the MAP-LP and derive the MPLP message updates for UFL. We also describe some of the properties of MPLP fixed points for UFL, used to prove the 3-approximation guarantee.

MAP-LP dual

MPLP is based on the dual problem of MAP-LP augmented with redundant primal variables $\bar{\mu}$, which are simply replicas of $\mu$. For each factor $\theta_c$, we add $N_c$ copies of $\mu_c(x_c)$ to the primal LP, where $N_c$ is the number of factors that share variables with $\theta_c$. For UFL, this corresponds to introducing $M$ copies of $\mu^C_i(x_i)$ and $N$ copies of $\mu^F_j(x_{ij})$, which we denote by $\bar{\mu}^C_i(x_i)$, $j = 1, \ldots, M$, and $\bar{\mu}^F_j(x_{ij})$, $i = 1, \ldots, N$, respectively. We include additional constraints that $\bar{\mu}_c = \mu_c$ and list all of the the constraints and associated dual

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Dual variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{x_{ij}} \mu_{ij}(x_{ij}) = 1$</td>
<td>$\delta_{ij}$</td>
</tr>
<tr>
<td>$\bar{\mu}^F_i(x_{ij}) = \mu^F_i(x_{ij})$</td>
<td>$\beta^F_{ij}(x_{ij})$</td>
</tr>
<tr>
<td>$\bar{\mu}^C_j(x_i) = \mu^C_j(x_i)$</td>
<td>$\beta^C_{ij}(x_i)$</td>
</tr>
<tr>
<td>$\sum_{x_{i\rightarrow j}} \bar{\mu}^C_j(x_i) = \mu_{ij}(x_{ij})$</td>
<td>$\eta_{ij}(x_{ij})$</td>
</tr>
<tr>
<td>$\sum_{x_{\leftarrow ij}} \bar{\mu}^F_j(x_{ij}) = \mu_{ij}(x_{ij})$</td>
<td>$\alpha_{ij}(x_{ij})$</td>
</tr>
</tbody>
</table>

Table 4.1: MPLP constraints and corresponding dual variables
variables in Table 4.1. The Lagrangian is:

\[
L = \sum_{i, j} \sum_{x_{ij}} \mu_{ij}(x_{ij})(-c_{ij}x_{ij}) + \sum_{j} \sum_{x_j} \mu^F_j(x_j)\theta^F_j(x_j) + \sum_{i, x_i} \mu^C_i(x_i)\theta^C_i(x_i) \\
+ \sum_{i, j} \sum_{x_{ij}} \beta^{Fi}_j(x_{ij})[\hat{\mu}^{Fi}_j(x_{ij}) - \mu^F_j(x_{ij})] \\
+ \sum_{i, j} \sum_{x_{ij}} \beta^{Cj}_i(x_i)[\hat{\mu}^{Cj}_i(x_i) - \mu^C_i(x_i)] \\
+ \sum_{i, j} \sum_{x_{ij}} \alpha_{ij}(x_{ij})[\mu_{ij}(x_{ij}) - \sum_{x_{i-j}} \hat{\mu}^{Fi}_j(x_{ij})] \\
+ \sum_{i, j} \sum_{x_{ij}} \eta_{ij}(x_{ij})[\mu_{ij}(x_{ij}) - \sum_{x_{i-j}} \hat{\mu}^{Cj}_i(x_i)] \\
+ \sum_{i, j} \delta_{ij}(1 - \sum_{x_{ij}} \mu_{ij}(x_{ij})) \\
(4.11)
\]

The dual LP objective is \( g(\beta, \alpha, \eta, \delta) = \sup_{\hat{\mu}, \mu} L \). Collecting the terms corresponding to primal variables:

\[
g(\beta, \alpha, \eta, \delta) = \sum_{i, j} \delta_{ij} \\
+ \sup_{\hat{\mu}} \left\{ \sum_{i, j, x_{ij}} \hat{\mu}^{Fi}_j(x_{ij})[\beta^{Fi}_j(x_{ij}) - \alpha_{ij}(x_{ij})] \right\} \\
+ \sum_{i, j, x_{i}} \hat{\mu}^{Cj}_i(x_i)[\beta^{Cj}_i(x_i) - \eta_{ij}(x_{ij})] \\
+ \sup_{\mu} \left\{ \sum_{i, j, x_{ij}} \mu_{ij}(x_{ij})[-c_{ij}x_{ij} + \alpha_{ij}(x_{ij}) + \eta_{ij}(x_{ij}) - \delta_{ij}] \right\} \\
+ \sum_{j, x_j} \mu^F_j(x_j)[\theta^F_j(x_j) - \sum_{i} \beta^{Fi}_j(x_{ij})] \\
+ \sum_{i, x_i} \mu^C_i(x_i)[\theta^C_i(x_i) - \sum_{j} \beta^{Cj}_i(x_i)] \\
(4.18)
\]

Since the marginals and their copies are non-negative, the terms multiplying them
must be ≤ 0 for the dual to be feasible. Hence, the dual LP is:

\[
\begin{align*}
\text{minimize} & \quad \sum_{ij} \delta_{ij} \\
\text{subject to} & \quad \beta_j^{Fi}(x_{ij}) - \alpha_{ij}(x_{ij}) \leq 0 \quad \forall i, j, x_{ij} \\
& \quad \beta_i^{Cj}(x_{ij}) - \eta_{ij}(x_{ij}) \leq 0 \quad \forall i, j, x_{ij} \\
& \quad \delta_{ij} \geq -c_{ij}x_{ij} + \alpha_{ij}(x_{ij}) + \eta_{ij}(x_{ij}) \quad \forall i, j, x_{ij} \\
& \quad \theta_j^{Fi}(x_{ij}) - \sum_i \beta_j^{Fi}(x_{ij}) = 0 \quad \forall i, j, x_{ij} \\
& \quad \theta_i^{Cj}(x_{ij}) - \sum_j \beta_i^{Cj}(x_{ij}) = 0 \quad \forall i, j, x_{ij}.
\end{align*}
\]

Finally, writing \( b_{ij}(x_{ij}) = -c_{ij}x_{ij} + \alpha_{ij}(x_{ij}) + \eta_{ij}(x_{ij}) \) and \( \delta_{ij} = \max_{x_{ij}} b_{ij}(x_{ij}) \), we can express the dual objective as a sum of maximized variable beliefs:

\[
\begin{align*}
\text{min} & \quad g(\beta, \alpha, \eta) = \sum_{ij} \max_{x_{ij}} b_{ij}(x_{ij}) \\
\text{s.t.} & \quad b_{ij}(x_{ij}) = -c_{ij}x_{ij} + \alpha_{ij}(x_{ij}) + \eta_{ij}(x_{ij}) \\
& \quad \alpha_{ij}(x_{ij}) = \max_{x_{-ij}} \beta_j^{Fi}(x_{ij}) \\
& \quad \eta_{ij}(x_{ij}) = \max_{x_{-ij}} \beta_i^{Cj}(x_{ij}) \\
& \quad \sum_i \beta_j^{Fi}(x_{ij}) = \theta_j^{Fi}(x_{ij}) \quad \forall j, x_{ij} \\
& \quad \sum_j \beta_i^{Cj}(x_{ij}) = \theta_i^{Cj}(x_{ij}) \quad \forall i, x_{ij}.
\end{align*}
\]

**MPLP-UFL message updates**

MPLP message updates correspond to block coordinate steps in the dual variables \( \beta \), obtained by optimizing over either \( \beta_i^{Cj}(x_{ij}) \) or \( \beta_j^{Fi}(x_{ij}) \), while holding all other variables constant. In fact, the dual LP objective can be expressed solely in terms of variables \( \beta_i^{Cj}(x_{ij}) \) and \( \beta_j^{Fi}(x_{ij}) \) as

\[
g(\beta, \alpha, \eta) = \sum_{ij} \max_{x_{ij}} \left[ -c_{ij}x_{ij} + \max_{x_{-ij}} \beta_j^{Fi}(x_{ij}) + \max_{x_{-j}} \beta_i^{Cj}(x_{ij}) \right].
\]

The \( \beta \) updates are:
\begin{align*}
\beta_i^{Cj}(x_i) & \leftarrow \frac{1}{M} \theta_i^C(x_i) - \frac{M - 1}{M} (\alpha_{ij}(x_{ij}) - c_{ij}x_{ij}) + \frac{1}{M} \sum_{k \neq j} (\alpha_{ik}(x_{ik}) - c_{ik}x_{ik}) \quad (4.28) \\
\beta_j^{Fi}(x_j) & \leftarrow \frac{1}{N} \theta_j^F(x_j) - \frac{N - 1}{N} (\eta_{ij}(x_{ij}) - c_{ij}x_{ij}) + \frac{1}{N} \sum_{k \neq i} (\eta_{kj}(x_{kj}) - c_{kj}x_{kj}) \quad (4.29)
\end{align*}

In practice, we only need to keep track of the message variables \( \eta_{ij}(x_{ij}) = \max_{x_{i-j}} \beta_i^{Cj}(x_i) \) and \( \alpha_{ij}(x_{ij}) = \max_{x_{-ij}} \beta_j^{Fi}(x_j) \). Substituting in the definitions of \( \theta_i^C(x_i) \) and \( \theta_j^F(x_j) \) and performing the maximizations yields the following message updates:

\begin{align*}
\eta_{ij}(1) & \leftarrow \frac{1}{M} \sum_{k \neq j} \alpha_{ik}(0) - \frac{M - 1}{M} (\alpha_{ij}(1) - c_{ij}) \\
\eta_{ij}(0) & \leftarrow \frac{1}{M} \sum_{k \neq j} \alpha_{ik}(0) + \frac{1}{M} \max_{k \neq j} (\alpha_{ik} - c_{ik}) - \frac{M - 1}{M} \alpha_{ij}(0)
\end{align*}

\begin{align*}
\alpha_{ij}(1) & \leftarrow \frac{1}{N} \sum_{k \neq i} \eta_{kj}(0) + \frac{1}{N} \left[ - f_j + \sum_{k \neq i} \max(0, \eta_{kj} - c_{kj}) \right] - \frac{N - 1}{N} (\eta_{ij}(1) - c_{ij}) \\
\alpha_{ij}(0) & \leftarrow \frac{1}{N} \sum_{k \neq i} \eta_{kj}(0) + \frac{1}{N} \max \left[ 0, - f_j + \sum_{k \neq i} \max(0, \eta_{kj} - c_{kj}) \right] - \frac{N - 1}{N} \eta_{ij}(0)
\end{align*}

As before, it suffices to only update the message differences \( \eta_{ij} = \eta_{ij}(1) - \eta_{ij}(0) \) and \( \alpha_{ij} = \alpha_{ij}(1) - \alpha_{ij}(0) \):

\begin{align*}
\eta_{ij} & \leftarrow -\frac{1}{M} \max_{k \neq j} (\alpha_{ik} - c_{ik}) - \frac{M - 1}{M} (\alpha_{ij} - c_{ij}) \quad (4.30) \\
\alpha_{ij} & \leftarrow \frac{1}{N} \min \left[ 0, - f_j + \sum_{k \neq i} \max(0, \eta_{kj} - c_{kj}) \right] - \frac{N - 1}{N} (\eta_{ij} - c_{ij}) \quad (4.31)
\end{align*}

**MPLP-UFL Fixed Point Properties**

Here, we decompose the MAP-LP dual objective \( g(\eta, \alpha, \beta) \) into components corresponding to uniquely and non-uniquely maximized beliefs.
The objective \( g(\eta, \alpha, \beta) \) is the sum of maximized beliefs. Let \( \mathbf{x}^* = \arg \max_{\mathbf{x}} \mathbf{b(x)} \) be any solution that maximizes beliefs. From the message update equations in 4.4,

\[
b_{ij}(x^*_{ij}) = \frac{1}{N} \sum_i \eta_{ij}(0) + \frac{1}{N} \max(0, -f_j + \sum_k \max(0, \eta_{kj} - c_{kj})) \tag{4.32}
\]

\[
= \frac{1}{M} \sum_j \alpha_{ij}(0) + \frac{1}{M} \max_k (\alpha_{ik} - c_{ik}) \tag{4.33}
\]

The dual objective can be computed as:

\[
\sum_{ij} b_{ij}(x^*_{ij}) = \sum_{ij} \eta_{ij}(0) + \sum_j \max(0, -f_j + \sum_i \max(0, \eta_{ij} - c_{ij})) \tag{4.34}
\]

\[
= \sum_{ij} \alpha_{ij}(0) + \sum_i \max_j (\alpha_{ij} - c_{ij}) \tag{4.35}
\]

Summing Eq. 4.34 and Eq. 4.35 and simplifying, we can express the dual objective as:

\[
\sum_{ij} b_{ij}(x^*_{ij}) = \sum_{ij} b_{ij}(0) + \sum_i \max_j (b_{ij} - \eta_{ij}) + \sum_j (0, -f_j + \sum_i \max(0, \eta_{ij} - c_{ij})) \tag{4.36}
\]

At a fixed point of MPLP, we can decompose the dual objective into components corresponding to variables with uniquely maximized beliefs, and variables for which \( b_{ij}(0) = b_{ij}(1) \). To do this, we first make note of some message properties. At convergence, the message updates evaluate to zero, and the differential beliefs all satisfy:

\[
b_{ij} = -c_{ij} + \alpha_{ij} + \eta_{ij}
\]

\[
= \frac{1}{M} (\alpha_{ij} - c_{ij}) - \frac{1}{M} \max_{k \neq j} (\alpha_{ik} - c_{ik}) \tag{4.37}
\]

\[
= \frac{1}{N} \min[\eta_{ij} - c_{ij}, -f_j + \sum_k \max(0, \eta_{kj} - c_{kj})] \tag{4.38}
\]

From the above equations, each facility \( j \) will be open, closed, or “tied” according to
whether \( b_j \equiv \max_i b_{ij} = -f_j + \sum_i \max(0, \eta_{ij} - c_{ij}) \) is greater than, less than, or equal to zero, respectively. A customer \( i \) will be connected or tied to a facility \( j \) (\( b_{ij} \geq 0 \)) only if \( \eta_{ij} \geq c_{ij} \). For each customer \( i \), the \( \eta_{ij} \) messages are equal for all \( j \) such that \( b_{ij} \geq 0 \) and we denote such messages by \( \hat{\eta}_i \).

Using these facts, we can decompose the dual objective into components \( g^1(\eta, \alpha, \beta) \) and \( g^0(\eta, \alpha, \beta) \) corresponding to connected and tied customers, respectively:

\[
-g^1(\eta, \alpha, \beta) = \sum_{j,b_{ij} \neq 0} f_j \max_i x_{ij}^* + \sum_{i,j,b_{ij} \neq 0} c_{ij} x_{ij}^* \tag{4.39}
\]

\[
-g^0(\eta, \alpha, \beta) = \sum_{i,b_{ij} = 0} \hat{\eta}_i \tag{4.40}
\]
Chapter 5

Benchmarking Message Passing Algorithms for UFL

So far, we have described graphical models and message passing algorithms for facility location problems. Our primary goal is to apply these algorithms to natural data, by formulating mixture modeling tasks as facility location instances. However, in this chapter evaluate our approach on synthetically generated UFL benchmarks, where the connection and facility costs are typically either randomly sampled from uniform or normal distributions, created using a set of rules, or both. The chosen data sets cover a wide variety of problem types: small and large instances, Euclidean, shortest-path, and random/non-metric costs.

We compare the performance of max-product and MPLP on UFL to two heuristic local search methods: Tabu Search of \cite{82} and Local Search of \cite{5}, as well as to two methods based on the LP relaxation dual: JMS \cite{39} and MYZ \cite{55}. 
5.1 Algorithms

5.1.1 JMS Algorithm

The JMS algorithm performs coordinate ascent in the dual of a LP relaxation of UFL; it has a 1.61-approximation guarantee and complexity $O(N^3)$. The algorithm pseudocode is given in Alg. 5.

**Algorithm 5 JMS ALGORITHM**

<table>
<thead>
<tr>
<th>Initialize customer budgets $B_i \leftarrow 0 \ \forall i \in C$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>while</strong> there exists an unconnected customer <strong>do</strong></td>
</tr>
<tr>
<td><strong>for all</strong> unconnected customers $i$ <strong>do</strong></td>
</tr>
<tr>
<td>Increase budget: $B_i \leftarrow B_i + \delta$</td>
</tr>
<tr>
<td><strong>end for</strong></td>
</tr>
<tr>
<td>Compute customer offers:</td>
</tr>
<tr>
<td><strong>for all</strong> unopened facilities $j$ <strong>do</strong></td>
</tr>
<tr>
<td>if customer $i$ is not connected <strong>then</strong></td>
</tr>
<tr>
<td>$O_{ij} \leftarrow \max(B_i - c_{ij}, 0)$</td>
</tr>
<tr>
<td>else if customer $i$ is connected to facility $k$ <strong>then</strong></td>
</tr>
<tr>
<td>$O_{ij} \leftarrow \max(c_{ik} - c_{ij}, 0)$</td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
<tr>
<td><strong>end for</strong></td>
</tr>
<tr>
<td>if facility $j$ not open, and $\sum_i O_{ij} &gt; f_j$ <strong>then</strong></td>
</tr>
<tr>
<td>open facility $j$ and connect all customers with $O_{ij} &gt; 0$</td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
<tr>
<td><strong>for all</strong> unconnected customers $i$, open facilities $j$ <strong>do</strong></td>
</tr>
<tr>
<td>if $B_i \geq c_{ij}$ <strong>then</strong></td>
</tr>
<tr>
<td>connect customer $i$ to facility $j$</td>
</tr>
<tr>
<td><strong>end if</strong></td>
</tr>
<tr>
<td><strong>end for</strong></td>
</tr>
<tr>
<td><strong>end while</strong></td>
</tr>
</tbody>
</table>

5.1.2 MYZ Algorithm

The MYZ algorithm [55] is an LP-based approximation algorithm for UFL with the best known approximation guarantee of 1.52. It uses the JMS algorithm as a subroutine, and applies scaling and greedy augmentation to it, as outlined in Alg. 6.
Algorithm 6 MYZ ALGORITHM

Scale up all facility costs $f_j$ by $\delta = 1.504$
Solve scaled instance by JMS
Scale down opening costs by $\delta$
repeat
    $E \leftarrow$ current solution cost
    for all unopened facilities $j$ do
        $E_j \leftarrow$ cost after additionally opening facility $j$
        $u_j \leftarrow (E - E_j - f_j) / f_j$
    end for
    open facility $j$ with maximum $u_j$
until max$_k u_k > 0$

5.1.3 Tabu Search for UFL

The simple Tabu search algorithm of [82] has been shown to work quite well and out-perform the genetic algorithm in [48] in terms of solution quality and execution time. The algorithm considers only variables $y$ that indicate which facilities are open. The pseudocode is given in Alg. 7, where the number of “tabu iterations” $K$ is adjusted using a standard scheme described in [82]. We run Tabu search 20 times with different random initializations and keep the best solution.

Algorithm 7 TABU SEARCH FOR UFL

Initialize:
$E^* \leftarrow \infty,$ $y$ arbitrary feasible solution
tabu-list $\leftarrow \emptyset$
repeat
    for all non-tabu facilities $j$ do
        $E_j \leftarrow$ cost saving by flipping $y_j$
    end for
    if max$_k E_k > 0$ then
        Flip variable $y_j$ with maximum $E_j$
        Put $j$ on tabu-list for $K$ iterations
    else
        Close random facility
    end if
Update connections, solution $E^*$, tabu-list
until change in $E^*$ in the last 500 iterations
5.1.4 Local Search for UFL

Similarly to Tabu, the Local search algorithm proposed by [5] also considers only the facility variables $y$, and is described in Alg. 8. Here, an operation $\text{op}(y)$ involves flipping a single variable $y_j$, or exchanging the status of an open and a closed facility. The algorithm parameters are set to $\epsilon = 0.1$ and $P(N, M) = N + M$. We run Local search 20 times with different random initializations and keep the best solution.

**Algorithm 8 LOCAL SEARCH FOR UFL**

```plaintext
Initialize:
y ← arbitrary feasible solution
$E^* ← E(y)$

while there is an operation $\text{op } j$ s.t. $E(\text{op } (y)) \leq (1 - \frac{\epsilon}{P(N,M)})E^*$ do
    $y ← \text{op}(y)$
    $E^* ← E(y)$
end while
```

5.1.5 Message-Passing Algorithms

The message passing algorithms we described include max-product belief propagation (BP), damped BP, and max-product linear programming (MPLP). As a reminder, BP and damped BP can be seen as efficient heuristic algorithms with no performance guarantees. MPLP corresponds to coordinate ascent in the dual of a LP relaxation, and has a 3-approximation guarantee when modified using the greedy decoding algorithm Alg. 4.

We show BP pseudocode for UFL in Alg. 9, noting that the message updates of BP, damped BP, and MPLP are related as:

- **BP:**
  \[ m^{(t+1)} ← m_{BP} \]

- **Damped BP:**
  \[ m^{(t+1)} ← (1 - \lambda_1)m_{BP} + \lambda_1 m^{(t)} \]

- **MPLP:**
  \[ m^{(t+1)} ← (1 - \lambda_2)m_{BP} + \lambda_2 (m^{(t)} - b^{(t)}) \]
In practice, we found that standard BP rarely converges, and damped BP typically converges for $\lambda_1 \geq 0.7$. The number of iterations to convergence of damped BP increases with $\lambda_1$. In our experiments, we use damped BP with $\lambda_1 = 0.8$.

MPLP is guaranteed to converge; however, it takes a very long time to do so. One possible explanation for this is that MPLP updates are similar in form to those of damped BP. The MPLP “damping” constant $\lambda_2$ in MPLP is not hand-tunable, but rather dependent on the graph, and for the UFL graphical model, it equals either $\lambda_2 = 1 - 1/N$ or $1 - 1/M$, depending on message type. In large problems with e.g. 1000 customers or facilities, this corresponds to messages changing by about 0.1% at each iteration. When the rate of change of messages decreases, it is also often unclear how to set the convergence threshold. In our experiments, we simply run MPLP to a maximum of 20,000 iterations. When the LP relaxation is not tight, we assign variables using the greedy 3-approximation algorithm on all but two data sets$^1$.

Algorithm 9 λ MAX-PRODUCT BELIEF PROPAGATION FOR UFL

1. Initialize $\eta \leftarrow 0$, $\alpha \leftarrow 0$
2. repeat
3.  for all customers $i$, facilities $j$ do
4.     Update: $\eta_{ij} \leftarrow (-\max_{k \neq j}(\alpha_{ik} - c_{ik}))$
5.  end for
6.  for all customers $i$, facilities $j$ do
7.     Update: $\alpha_{ij} \leftarrow \min[0, -f_j + \sum_{k \neq i} \max(0, \eta_{kj} - c_{kj})]$
8.  end for
9.  Compute beliefs: $b = \alpha + \eta - c$
10. until convergence
11. Assign customers to facilities that maximize their beliefs.

5.2 Data Sets

We evaluate our algorithms on a large number of benchmark problem instances, chosen to cover different types of facility location problems: small, medium, and large size, with

$^1$Perfect codes and Chessboard datasets
Table 5.1: ORLIB parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Size ((N \times M))</th>
</tr>
</thead>
<tbody>
<tr>
<td>71, 72, 73, 74</td>
<td>50 \times 16</td>
</tr>
<tr>
<td>101, 102, 103, 104</td>
<td>50 \times 25</td>
</tr>
<tr>
<td>131, 132, 133, 134</td>
<td>50 \times 50</td>
</tr>
<tr>
<td>(a, b, c)</td>
<td>1000 \times 100</td>
</tr>
</tbody>
</table>

Euclidean and shortest-path metric and non-metric costs, and randomly generated costs. All of the benchmarks are available on-line from the Max-Planck Institute\(^2\). In this section, we provide a brief description of each benchmark data set.

5.2.1 ORLIB instances

The ORLIB-cap instances [7] fall among the most widely used UFL benchmarks. They are non-metric, and their sizes are specified in Table 5.1.

5.2.2 Instances with strong local minima

Perfect codes

A perfect code is a binary code of is a nonempty subset of all possible binary words with length \(k\) whose pairwise Hamming distance is at least \(r\). A perfect code with distance \(r = 3\) produces a partition of the \(k\)-dimensional hypercube into disjoint spheres of radius 1.

In perfect code benchmarks [44], customer/facilities are the set of all binary vectors of length \(k\). All facility costs are set to 3,000. If the Hamming distance between two customers is less than or equal to 1, connection costs are sampled uniformly from \(\{0, 1, 2, 3, 4\}\); otherwise they are set to be very large. An arbitrary perfect code corresponds to a strong local minimum of the UFL. The number of codes and the minimum distance between two strong local minima grows exponentially with \(k\). The data set

\(^2\)Online at http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/
contains 32 instances of size $M = N = 128$, corresponding to $k = 7$.

**Chessboard**

Chessboard instances [44] are based on positions on a $3k \times 3k$ chessboard, which wraps around on both sides into a torus. There are $M = N = 9k^2$ customers/facilities, one per position. All facility costs are set to 3,000. If a chess king can reach a customer from a facility, connection costs are sampled uniformly from $\{0, 1, 2, 3, 4\}$; otherwise they are set to be very large. In the optimal solution, $k^2$ kings cover the board, and the number of sets of $k^2$ kings covering the board grows exponentially with $k$. The data set consists of 30 benchmarks with $k = 4$ and $M = N = 144$.

**5.2.3 Finite projective planes**

Finite projection plane benchmarks [44] are based on incidence matrices for finite projective planes of dimension $k$, where $M = N = k^2 + k + 1$. All facility costs are set to 3000. There are exactly $N + 1$ non-infinity connection costs, which are sampled from the set $\{0, 1, 2, 3, 4\}$. There are two data sets with $k = 11$ and $k = 17$, each containing 30 instances. These instances can be solved to optimality in polynomial time, but can present a challenge for local search algorithms.

**5.2.4 Random costs**

**Large duality gap**

In GapA, GapB and GapC benchmarks [44], customers and facilities are the same set, $N = M = 100$ and all facility costs $f_j$ are set to 3000. Connection costs are either cheap (sampled uniformly from $\{0, 1, 2, 3, 4\}$) or very expensive. In type A, each customer has 10 cheap connections. In type B, every facility has 10 cheap connections. In type C, there are 10 cheap connections for each customer and each facility. These instances are
known to have large duality gaps (typically between 20 and 30 percent) and increase in difficulty for mathematical programming and branch-and-bound algorithms from type A to type C. There are 30 instances of each type.

**Uniform**

In uniform benchmarks [44], customers and facilities are the same set, instance size is $M = N = 100$, all facility costs $f_j$ are set to 3000 and connection costs $c_{ij}$ are drawn from a discrete uniform distribution on $[0, 10000]$.

**Bilde-Krarup**

In the Bilde-Krarup instances [9], connection costs $c_{ij}$ are drawn from a discrete uniform distribution, and facility costs $f_j$ are either drawn from a discrete uniform distribution, or set to a constant. The instance sizes and parameters are given in Table 5.2. 10 instances were generated for each set of parameters.

**$M^*$ instances**

The $M^*$ instances [48] are known to contain a small number of “useless” facilities, and a very large number of suboptimal solutions, making them especially difficult for integer programming methods. The customer connection costs $c_{ij}$ are generated by multiplying an integer drawn uniformly on $[b_{\text{min}}, b_{\text{max}}]$ by a real number drawn uniformly on $[c_{\text{min}}, c_{\text{max}}]$. The facility costs $f_j$ are set according to

<table>
<thead>
<tr>
<th>Type</th>
<th>Size $(N \times M)$</th>
<th>Facility costs $f_j$</th>
<th>Connection costs $c_{ij}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>$50 \times 100$</td>
<td>Discrete Uniform (1000,10000)</td>
<td>Discrete Uniform (0,1000)</td>
</tr>
<tr>
<td>$C$</td>
<td>$50 \times 100$</td>
<td>Discrete Uniform (1000,10000)</td>
<td>Discrete Uniform (0,1000)</td>
</tr>
<tr>
<td>$Dq$</td>
<td>$30 \times 80$</td>
<td>Identical, 1000$q$</td>
<td>Discrete Uniform (0,1000)</td>
</tr>
<tr>
<td>$Eq$</td>
<td>$50 \times 100$</td>
<td>Identical, 1000$q$</td>
<td>Discrete Uniform (0,1000)</td>
</tr>
</tbody>
</table>

Table 5.2: Bilde-Krarup Sequences ($q = 1, \ldots, 10$)
Table 5.3: $M^*$ parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Size ($N \times M$)</th>
<th>$f$</th>
<th>$c$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MO$</td>
<td>$100 \times 100$</td>
<td>[50, 30]</td>
<td>[2, 10]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>$MP$</td>
<td>$200 \times 200$</td>
<td>[100, 600]</td>
<td>[2, 10]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>$MQ$</td>
<td>$300 \times 300$</td>
<td>[150, 900]</td>
<td>[2, 10]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>$MR$</td>
<td>$500 \times 500$</td>
<td>[100, 600]</td>
<td>[0.5, 5]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>$MS$</td>
<td>$1000 \times 1000$</td>
<td>[200, 1200]</td>
<td>[0.5, 5]</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>$MT$</td>
<td>$2000 \times 2000$</td>
<td>[400, 2400]</td>
<td>[0.5, 5]</td>
<td>[1, 5]</td>
</tr>
</tbody>
</table>

Table 5.4: Galvao-Raggi sequences

<table>
<thead>
<tr>
<th>Size ($N$)</th>
<th>Edge density $\delta$</th>
<th>Facility costs $f_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.061</td>
<td>$\mathcal{N}(25.1, 14.1)$</td>
</tr>
<tr>
<td>70</td>
<td>0.043</td>
<td>$\mathcal{N}(42.3, 20.7)$</td>
</tr>
<tr>
<td>100</td>
<td>0.025</td>
<td>$\mathcal{N}(51.7, 28.9)$</td>
</tr>
<tr>
<td>150</td>
<td>0.018</td>
<td>$\mathcal{N}(186.1, 101.5)$</td>
</tr>
<tr>
<td>200</td>
<td>0.015</td>
<td>$\mathcal{N}(149.5, 94.4)$</td>
</tr>
</tbody>
</table>

\[
f_j = f_{\text{max}} - \frac{(S_j - S_{\text{min}})(f_{\text{max}} - f_{\text{min}})}{S_{\text{max}} - S_{\text{min}}} \tag{5.1}\]

where $S_j = \sum_i c_{ij}$. There are 6 problem types; Table 5.3 specifies the instance sizes and parameters $f_{\text{min}}, f_{\text{max}}, b_{\text{min}}, b_{\text{max}}, c_{\text{min}}$ and $c_{\text{max}}$ for each type.

### 5.2.5 Metric instances

**Galvao-Raggi**

In Galvao-Raggi [27] benchmarks, customers/facilities are vertices in a weighted graph, with edge density $\delta$ and weights drawn uniformly in $[1, N]$. Facility costs $f_j$ are sampled from a normal distribution. Connection costs $c_{ij}$ are set to the length of the shortest paths between $i$ and $j$ in the graph. The instance sizes and parameters are given in Table 5.4, and there are 10 instances of each type.
Euclidean plane instances

In Euclidean plane instances [44], customers/facilities are a set of $N = 100$ points drawn randomly on a square of size $7000 \times 7000$. Facility costs $f_j$ are set to 3000 and connection costs $c_{ij}$ are set to pairwise Euclidean distances.

5.3 Experimental results

We compared algorithms JMS, MYZ, Tabu Search, Local Search, damped BP and MPLP on the described data sets in terms of both solution quality and efficiency. The randomized algorithms Tabu Search and Local Search were run 20 times with different random initializations; we report both the average and best run performance.

Table 5.5 shows the number of instances solved to optimality in each data set, while Table 5.6 shows the average solution error per data set. Damped BP has the best performance overall, in terms of both number of global optima found and the cost of suboptimal solutions. Its performance is especially impressive on instances with strong local optima (Perfect codes, Chessboard), and large duality gap instances (GapA, GapB, and GapC). Randomized local search algorithms also find good solutions on small instances, but perform poorly on instances with many local optima.

Approximation algorithms based on LP relaxations (JMS, MYZ, MPLP) perform slightly better than other algorithms on ORLIB and metric problems (Galvao-Raggi, Euclid), but are inferior overall. MYZ uses JMS as a subroutine, and applies greedy scaling and augmentation to it. Although this procedure leads to a better approximation guarantee, it does not necessarily yield better solutions in practice. MPLP performance is comparable to that of JMS and MYZ.

When it comes to speed of convergence, Local and Tabu search algorithms are the fastest. However, obtaining good results often requires a number of random restarts with different initializations; we report the total number of iterations for 20 such restarts.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Total</th>
<th>Tabu</th>
<th>Local</th>
<th>dBP</th>
<th>MPLP</th>
<th>JMS</th>
<th>MYZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORLIB</td>
<td>32</td>
<td>12</td>
<td>10</td>
<td>9</td>
<td>13</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Perfect Codes</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Chessboard</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fpp, $k = 11$</td>
<td>30</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Fpp, $k = 17$</td>
<td>30</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GapA</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GapB</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GapC</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Uniform</td>
<td>30</td>
<td>1</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Bilde-Krarup</td>
<td>220</td>
<td>138</td>
<td>146</td>
<td>80</td>
<td>21</td>
<td>63</td>
<td>99</td>
</tr>
<tr>
<td>M</td>
<td>22</td>
<td>14</td>
<td>14</td>
<td>10</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Galvao-Raggi</td>
<td>50</td>
<td>19</td>
<td>12</td>
<td>37</td>
<td>40</td>
<td>41</td>
<td>38</td>
</tr>
<tr>
<td>Euclidean</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.5: Number of instances solved to optimality for each data set.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Tabu</th>
<th>Local</th>
<th>dBP</th>
<th>MPLP</th>
<th>JMS</th>
<th>MYZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORLIB</td>
<td>0.51</td>
<td>0.51</td>
<td>0.47</td>
<td>0.15</td>
<td>0.77</td>
<td>0.72</td>
</tr>
<tr>
<td>Perfect Codes</td>
<td>38.84</td>
<td>38.26</td>
<td>0.015</td>
<td>30.7</td>
<td>157.62</td>
<td>157.62</td>
</tr>
<tr>
<td>Chessboard</td>
<td>27.94</td>
<td>27.94</td>
<td>7.67</td>
<td>23.80</td>
<td>115.39</td>
<td>115.81</td>
</tr>
<tr>
<td>Fpp, $k = 11$</td>
<td>49.91</td>
<td>47.06</td>
<td>0.03</td>
<td>78.37</td>
<td>179.95</td>
<td>259.76</td>
</tr>
<tr>
<td>Fpp, $k = 17$</td>
<td>0.10</td>
<td>0.10</td>
<td>0.02</td>
<td>133.15</td>
<td>44.39</td>
<td>55.34</td>
</tr>
<tr>
<td>GapA</td>
<td>21.74</td>
<td>21.19</td>
<td>0.57</td>
<td>26.35</td>
<td>14.82</td>
<td>15.10</td>
</tr>
<tr>
<td>GapB</td>
<td>16.86</td>
<td>16.86</td>
<td>4.35</td>
<td>31.82</td>
<td>88.08</td>
<td>77.35</td>
</tr>
<tr>
<td>GapC</td>
<td>18.34</td>
<td>18.10</td>
<td>1.32</td>
<td>32.39</td>
<td>101.72</td>
<td>79.36</td>
</tr>
<tr>
<td>Uniform</td>
<td>1.32</td>
<td>1.32</td>
<td>0.77</td>
<td>9.82</td>
<td>2.49</td>
<td>2.73</td>
</tr>
<tr>
<td>Bilde-Krarup</td>
<td>0.48</td>
<td>0.45</td>
<td>1.13</td>
<td>13.31</td>
<td>1.53</td>
<td>0.77</td>
</tr>
<tr>
<td>M</td>
<td>0.06</td>
<td>0.05</td>
<td>0.21</td>
<td>11.53</td>
<td>28.86</td>
<td>2.86</td>
</tr>
<tr>
<td>Galvao-Raggi</td>
<td>0.02</td>
<td>0.02</td>
<td>0.092</td>
<td>0.09</td>
<td>0.036</td>
<td>0.068</td>
</tr>
<tr>
<td>Euclidean</td>
<td>1.77</td>
<td>1.71</td>
<td>0.83</td>
<td>0.58</td>
<td>1.12</td>
<td>1.35</td>
</tr>
</tbody>
</table>

Table 5.6: Percentage error, measured as the amount by which the obtained cost exceeds the optimal cost or its lower bound, averaged over all instances in each data set.
Table 5.7: Average number of iterations required to convergence for each data set.

<table>
<thead>
<tr>
<th>Problem</th>
<th>$(M \times N)$</th>
<th>Tabu</th>
<th>Local</th>
<th>dBp</th>
<th>MPLP</th>
<th>JMS</th>
<th>MYZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORLIB</td>
<td>Table 5.1</td>
<td>10,180</td>
<td>200</td>
<td>1670</td>
<td>9,967</td>
<td>&gt;20K</td>
<td>&gt;20K</td>
</tr>
<tr>
<td>Perfect Codes</td>
<td>128 × 128</td>
<td>10,440</td>
<td>4,620</td>
<td>170</td>
<td>&gt;20K</td>
<td>2,611</td>
<td>4,036</td>
</tr>
<tr>
<td>Chessboard</td>
<td>144 × 144</td>
<td>10,420</td>
<td>440</td>
<td>2,636</td>
<td>&gt;20K</td>
<td>2,556</td>
<td>3,854</td>
</tr>
<tr>
<td>Fpp, $k = 11$</td>
<td>133 × 133</td>
<td>10,220</td>
<td>240</td>
<td>177</td>
<td>19,641</td>
<td>675</td>
<td>2,357</td>
</tr>
<tr>
<td>Fpp, $k = 17$</td>
<td>307 × 307</td>
<td>10,340</td>
<td>360</td>
<td>179</td>
<td>&gt;20K</td>
<td>680</td>
<td>3,076</td>
</tr>
<tr>
<td>GapA</td>
<td>100 × 100</td>
<td>10,260</td>
<td>280</td>
<td>7,445</td>
<td>&gt;20K</td>
<td>2,363</td>
<td>3,877</td>
</tr>
<tr>
<td>GapB</td>
<td>100 × 100</td>
<td>10,320</td>
<td>360</td>
<td>10,730</td>
<td>19,426</td>
<td>2,258</td>
<td>4,204</td>
</tr>
<tr>
<td>GapC</td>
<td>100 × 100</td>
<td>10,300</td>
<td>320</td>
<td>8,279</td>
<td>18,232</td>
<td>1,756</td>
<td>3,579</td>
</tr>
<tr>
<td>Uniform</td>
<td>100 × 100</td>
<td>10,300</td>
<td>320</td>
<td>3,444</td>
<td>19,867</td>
<td>1,885</td>
<td>2,342</td>
</tr>
<tr>
<td>Bilde-Krarup</td>
<td>50 × 100</td>
<td>10,100</td>
<td>100</td>
<td>535</td>
<td>13,739</td>
<td>698</td>
<td>756</td>
</tr>
<tr>
<td>M</td>
<td>Table 5.3</td>
<td>10,140</td>
<td>140</td>
<td>151</td>
<td>19,677</td>
<td>27</td>
<td>33</td>
</tr>
<tr>
<td>Galvao-Raggi</td>
<td>Table 5.4</td>
<td>10,880</td>
<td>900</td>
<td>154</td>
<td>10,099</td>
<td>&gt;20K</td>
<td>&gt;20K</td>
</tr>
<tr>
<td>Euclidean</td>
<td>100 × 100</td>
<td>10,300</td>
<td>320</td>
<td>170</td>
<td>&gt;20K</td>
<td>1,777</td>
<td>1,935</td>
</tr>
</tbody>
</table>

The speed of convergence for JMS and MYZ depends on the rate at which the customer “budgets” $B_i$ are increased. As all costs are integers, we increased $B_i$’s by 1 at each iteration; the total number of iterations could conceivably be decreased by dynamically adjusting this rate.

As discussed, in most problems we simply ran MPLP to a maximum of 20,000 iterations. MPLP variables (messages) typically change at a low rate, and it is often unclear how to set a convergence threshold. As this rate relates to the structure of the graphical model, this opens questions about possibly speeding up the algorithm by a more clever decomposition of the problem into factors.

Damped BP tends to converge fairly quickly, except in a small number of cases where messages oscillate, resulting in a higher number of iterations on average. As it also performs well in terms of solution quality, it should be the algorithm of choice for practitioners.
Chapter 6

FLoSS: Facility Location for Subspace Segmentation

In this chapter, we describe an algorithm called FLoSS: Facility Location for Subspace Segmentation, which discovers multiple low-dimensional linear subspaces in high-dimensional data by posing the problem as an instance of facility location. We apply FLoSS to synthetic data sets, as well as to the problem of motion segmentation in video sequences, obtaining results comparable to state-of-the-art methods.

6.1 Subspace Segmentation

Many statistical models used for data analysis in vision assume that high-dimensional input data has an intrinsic low-dimensional representation. Furthermore, many such models assume the data can be well approximated as lying on a linear subspace; these include principal component analysis (PCA) [40], independent component analysis [36], factor analysis [30], and nonnegative matrix factorization [52]. Although the linearity assumption is often inaccurate, it nevertheless turns out to be a reasonable and useful approximation in many cases [80, 81, 89]. Even non-linear dimensionality reduction methods typically assume that data is locally linear and can be represented as some
In *subspace segmentation*, the underlying assumption is that the data is composed of points lying on several distinct linear subspaces, not necessarily of the same intrinsic dimension, as illustrated in Fig. 6.1\(^1\). The goal of subspace segmentation is to recover the underlying subspaces and to assign the data points to one subspace each. Thus, it is a more flexible model compared to the single linear subspace representation, but it still retains some of the computationally favorable properties of linear subspace models.

Subspace segmentation arises in a number of computer vision applications. One example is clustering images of different objects under varying illumination. It has been shown in [34] that a set of images of a Lambertian object under varying lighting conditions forms a convex polyhedral cone in the image space, which is well-approximated by a low dimensional subspace. As images of different objects lie on different subspaces, subspace segmentation can be used for clustering such images. Another application is to 3D multi-body video motion segmentation from point correspondences. Given the image coordinates of several keypoints lying on a rigid object, it can be shown that vectors of point trajectories lie on a linear subspace of dimension 2, 3 or 4 [78, 88]. When the tracked keypoints lie on several moving objects, the motion segmentation task of clustering points according to object is another instance of subspace segmentation.

---

\(^1\)In general, we will be modeling *affine* subspaces, i.e. those that do not necessarily pass through the origin, as in the first example of Fig. 6.1
In Facility Location for Subspace Segmentation, or FLoSS, we formulate subspace segmentation as an instance facility location by constructing a large initial set of candidate subspaces of dimension $D_j - 1$ from randomly sampled $D$-tuples of linearly independent data points. These subspaces serve as facilities, whose opening costs $f_j$ increase with intrinsic subspace dimensionality. The assignment cost $c_{ij}$ of a customer $i$ to facility $j$ is set to the squared normal distance from the point to the subspace. Once the facilities and costs are initialized, solution are obtained using the damped max-sum algorithm.

### 6.1.1 Previous Work

There exist numerous notable subspace segmentation algorithms, having different underlying approaches to the problem. When the number of subspaces is unknown, a sensible approach is to search for them one at a time, and select the one that represents a large number of points well at each pass. One such algorithm is random sample consensus (RANSAC) [23,77,87], a generic algorithm for outlier detection. RANSAC fits a $(D-1)$-dimensional subspace by iteratively (1) constructing a basis from $D$ randomly sampled points, (2) computing the normal distance from all points to this subspace, and (3) labeling those above some distance threshold as outliers. This is repeated until a specified number of inliers is reached, or a sufficient number of points have been sampled. Multiple subspaces are found iteratively, by removing the inliers from the previous step and repeating. A similar idea - that of iteratively searching for a subspace with the most inliers - is used by Da Silva et.al. [17]. They formulate this task as an unconstrained, but non-convex optimization problem, with improved efficiency over RANSAC. Neither method provides a direct way of estimating subspace dimensionalities. One proposed solution is to start with the highest-dimensional model, and recursively check each found solution for lower-dimensional models [14]. An alternative is to simultaneously apply the algorithm on multiple hypotheses and use model selection [24,67].

When the number of subspaces and their dimensionalities are specified, it is more
intuitive to determine all subspaces at once. One approach is to iterate between assigning points to their nearest subspaces, and re-estimating the subspace bases from the assigned points. \(k\)-subspaces \([34]\), an extension of the \(k\)-means algorithm, iterates between making hard assignments of points to subspaces based on minimal point-subspace normal distance, and re-computing the subspace bases using PCA. Mixture of pPCA (mpPCA) \([76]\) makes this process probabilistic by using latent variables to indicate the assignment of each point to one of \(k\) probabilistic PCA models. The model parameters and the probability distribution over the latent variables are estimated iteratively, using the Expectation Maximization (EM) algorithm \([18]\). Both methods can be sensitive to initialization and local optima.

Another possible approach, when the subspace number and dimensionality are available, is to construct the solution algebraically. Generalized PCA (GPCA) \([84]\) represents a union of \(k\) subspaces embedded in \(\mathbb{R}^D\) by a set of homogeneous polynomials of degree \(k\) in \(D\) variables. The polynomial coefficients can be estimated linearly from the data. The complexity of GPCA scales as \(k^D\), and the number of data points needed to estimate polynomials is exponential in \(k\); hence, it is only practical for a small number of low-dimensional subspaces. When the number of subspaces is unavailable, the authors determine it by estimating the rank of a matrix. A recursive approach similar to \([14]\) can be used when subspace dimensionalities are unknown.

Subspace separation (SS) \([41]\) is also an algebraic approach. It relies on the observation that when the subspaces are linearly independent and noise-free, it is possible to compute a binary data interaction matrix, indicating whether two points lie on the same subspace or not. Additive noise is addressed by using model selection to decide whether to merge subspaces.

Overall, none of the methods provide an effective way of estimating the number of subspaces and their dimensionalities. However, there exist applications in which subspace structures are known beforehand, the most notable being motion segmentation where
underlying dimensionality is 2, 3 or 4. Indeed, many subspace segmentation methods were actually designed as motion segmentation algorithms [41, 72, 88].

The multi-stage learning (MSL) algorithm of [72] for motion segmentation refines the subspace segmentation results of SS using three stages of mpPCA of increasing complexity, each corresponding to a different type of motion. The simplest mpPCA model is initialized using SS, and the results at each stage are used to initialize the next stage. In this way, MSL accounts for the cases where SS fails, namely, when the subspaces are co-dependent. This can occur frequently in motion data, especially when the motion of the points is in part due to a moving camera.

Another multi-body motion segmentation method is local subspace affinity (LSA) [88]. This is an algebraic method that first projects points onto the first $R$ principal components and then onto a hyper-sphere $S^{R-1}$. A local subspace is fit around each point and its $k$ nearest neighbors. The points are then clustered using spectral clustering [68] with pairwise similarities computed using angles between the local subspaces. Misclassification can occur near the intersection of two subspaces (as the nearest neighbors lie on different subspaces), or when the nearest neighbors do not span the selected subspace. Model selection is used to select appropriate subspace dimensionality.

The algorithms of [53] and [51] (i.e. FLoSS) frame the problem as an instance of uncapacitated facility location. Here, subspaces are constructed from randomly sampled $D$-tuples of data, as in RANSAC. However, all constructed subspaces (of possibly different dimensionality) are considered simultaneously, and all $k$ subspaces are selected at once.

The subspace dimensionalities or their range are required as inputs, and the number of subspaces is determined automatically. We note a few differences between FLoSS and [53]: the method of [53] solves UFL by simply rounding the solutions of the natural LP relaxation, which can be extremely inefficient for large problem instances. Furthermore, the facility costs are constant and do not reflect the dimensionality of the underlying subspaces.
6.1.2 Experiments on Synthetic Data

We compare the subspace segmentation results obtained FLoSS to those of RANSAC, mpPCA and GPCA on illustrative synthetic data sets. We first investigate the case of subspaces of the same dimensionality. We generate several synthetic data sets by sampling data points from planes in $\mathbb{R}^3$, and adding orthogonal Gaussian noise with variance at 5% of data variance. For all algorithms, we specify the number of subspaces $k$ and their dimensionality. The segmentation results are shown in Fig. 6.2, where the colors indicate subspace membership.

In general, RANSAC does not give very good results, and its performance mainly depends on the number of iterations. mpPCA performs well on most data sets. However, as it is geared towards modeling mixtures of linear segments rather than infinite subspaces, it may assign two disjoint pieces of the same subspace to different mixture components, as illustrated in the top row of Fig. 6.2. In addition, mpPCA can have difficulties distinguishing linear segments that overlap close to their means, as is the case for the data shown in the middle row of Fig. 6.2. Although GPCA gives good results on a variety of subspace configurations, its performance degrades as the number of subspaces $k$ increases since the number of data points needed to estimate subspaces is exponential in $k$. This explains its poor performance on the 4-plane data set in the bottom row of Fig. 6.2. GPCA can also be susceptible to noise; in fact, as noted in [84], it is suboptimal compared to the other algorithms in the Gaussian noise case when $k > 1$.

FLoSS gives very good results on the example configurations. It treats subspaces as infinite, and its performance is not affected by disconnected segments of the same subspace or the point of intersection of several subspaces. Increasing the number of subspaces $k$ does not degrade its performance either, although higher values of $k$ may require using more facilities at initialization.

We illustrate a case where FLoSS may fail using a more challenging data set, shown in Fig. 6.3. The data set contains a plane and two co-planar lines, at two levels of noise:
Figure 6.2: Comparison of different algorithms on data sets consisting of planes, (a) RANSAC, (b) mpPCA, (c) GPCA, and (d) FLoSS

1% and 5% of data variance. We use a fixed dimensionality of 2 for mpPCA, RANSAC and GPCA, and initialize FLoSS with both 1D and 2D subspaces. Although it is possible to specify different dimensionalities for GPCA, we found that fixed dimensionality gives better results using the code available at http://perception.csl.uiuc.edu/gpca/.

On this data set, only mpPCA and GPCA correctly identify the subspaces, and only in the low-noise case. FLoSS, on the other hand, groups the two lines into one plane at both noise levels. In general, FLoSS prefers lower-dimensional subspaces through lower costs. However, having several densely sampled $D$-dimensional subspaces embedded in a $(D + 1)$-dimensional subspace may offset the cost difference, causing FL to choose the $(D + 1)$ dimensional subspace. As the structure of the subspaces is unknown in general, it is difficult to set facility costs so as to prevent this; a possible remedy could be the recursive approach of [14].
Figure 6.3: Mixed dimensionality subspaces, two noise levels: $\sigma^2 = 0.01$ (top row) and $\sigma^2 = 0.05$ (bottom row). (a) RANSAC, (b) mpPCA, (c) GPCA, and (d) FLoSS

6.2 Multibody Motion Segmentation

Motion segmentation is the task of identifying different motions in a video containing multiple moving objects, with numerous computer vision applications including surveillance, tracking and action recognition [78]. Clustering tracked points lying on rigidly moving objects has been shown to correspond to identifying low-dimensional linear subspaces of a high-dimensional space [41]. In this section, we review the geometry of 3D rigid body motion and apply FLoSS to a benchmark motion segmentation database.

6.2.1 3D Motion Geometry

Let $\{w_{f,p} \in \mathbb{R}^2\}_{p=1,...,P}^{f=1,...,F}$ be the image projections of $P$ 3D points $\{X_p \in \mathbb{P}^3\}_{p=1,...,P}$, lying on a rigidly moving object, over $F$ frames of a rigidly moving camera. Under the affine projection model, keypoint coordinates satisfy $w_{f,p} = A_f X_p$. Here, $A_f \in \mathbb{R}^{2\times4}$ is the affine camera matrix at frame $f$, which depends on the camera calibration parameters $K_f \in \mathbb{R}^{2\times3}$ and the object pose $(R_f t_f) \in SE(3)$ as:
\[ A_f = K_f \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} G \begin{bmatrix} R_f & t_f \\ o^T & 1 \end{bmatrix} \] (6.1)

Let \( W \in \mathbb{R}^{2F \times P} \) be a matrix whose columns are the 2D point trajectories over \( F \) frames. \( W \) can be decomposed into a structure matrix \( S \in \mathbb{R}^{P \times 4} \) and a motion matrix \( M \in \mathbb{R}^{2F \times 4} \)

\[
W_{2F \times P} = MS^T = \begin{bmatrix} A_1 \\ \vdots \\ A_F \end{bmatrix}_{2F \times 4} \begin{bmatrix} X_1 & \cdots & X_P \end{bmatrix}_{2F \times 4} \tag{6.2}
\]

Therefore, the 2D trajectories of a set of 3D points captured by a rigidly moving camera live in a subspace of dimension \( 2 \leq \text{rank}(W) \leq 4 \). When the tracked points lie on \( n \) moving objects, the trajectories lie on multiple linear subspaces of \( \mathbb{R}^{2F} \), and the matrix of 2D point trajectories can be decomposed as:

\[
W = [W_1, W_2, \ldots, W_n] \Gamma \tag{6.3}
\]

\[
= [M_1, M_2, \ldots, M_n] \begin{bmatrix} S_1^T \\ S_2^T \\ \vdots \\ S_n^T \end{bmatrix} \Gamma \tag{6.4}
\]

\[
= MS^T \Gamma \tag{6.5}
\]

where \( \Gamma \) is a permutation matrix. It follows that one approach is to find \( \Gamma \) so that \( W \) factors into a motion matrix \( M \) and a block-diagonal structure matrix \( S \). However,
in order for such factorization to hold, the motion subspaces must be independent, i.e. \( \dim(W_i \cap W_j) = 0 \). Unfortunately, most practical video sequences contain partially dependent motions, due to articulated motion or a moving camera, which remains one of the main challenges in multibody motion segmentation.

### 6.2.2 Hopkins155 Motion Segmentation Dataset

A benchmark database for multi-body motion segmentation from point correspondences is the Hopkins155 database [78]. The database contains 50 video sequences of indoor and outdoor scenes, each containing two or three motions. Additionally, the 35 three-motion videos are split into \( \binom{3}{2} \) groups containing only two out of three motions, resulting in a total of 155 sequences. The data contains subspaces of different dimensionalities. The three video types that make up the database are:

- **Checkerboard**: 104 video sequences with 2 checkerboard-pattern objects. The camera undergoes rotation, translation, or both.
- **Traffic**: 38 sequences of outdoor traffic scenes, taken by a moving hand-held camera.
- **Articulated and non-rigid sequences**: 13 video sequences of motions constrained by joints and non-rigid motions.

Example frames from the three types of video sequences are shown in Fig. 6.4.

### 6.2.3 Hopkins155 Experiments

We evaluate the performance on FLoSS on the benchmark Hopkins155 motion segmentation database, comparing it to subspace segmentation using RANSAC, GPCA and mpPCA, as well as the LSA and MSL motion segmentation algorithms. Except for FLoSS and mpPCA, the reported results are obtained from [78], where the following settings were used: GPCA was run on the first 5 principal components of the data matrix.
Figure 6.4: Example frames with keypoints (left) and trajectories (right) of checkerboard, traffic, and articulated motion sequences from the Hopkins155 database. The keypoints colors denote hand labeled objects.
W, and LSA was run on the first $k$ principal components, where $k$ was the number of objects present. For RANSAC, the dimension of all subspaces was assumed to be 4; the algorithm was run 1000 times on each sequence, and the average results were recorded. mpPCA was run on the first 12 principal components of $W$, and the subspace dimensionality was set to 4. FLoSS was also run on the first 12 principal components of $W$, and initialized with random subsets of 3, 4 and 5 points (corresponding to subspaces of dimension 2, 3, and 4).

The segmentation errors, calculated as the percentage of misclassified points, are summarized in Tables 6.1 and 6.2. We note that no single method outperforms all others for all data sets. While GPCA achieves very good results for the 2 objects data, it performs poorly for the 3 objects data. As for the motion segmentation algorithms, LSA performs well, although inconsistently; while it is one of the best methods for the checkerboard sequences, it has the worst performance on traffic. MSL also performs well overall, notably better than mpPCA. Recall that MSL consists of three stages of mpPCA, initialized using the subspace separation algorithm, and adapted to different types of motion including degenerate. The large gap in the performance of the two methods is an indication of the sensitivity of mpPCA to initialization and variable subspace dimensionality.

FLoSS outperforms all other methods on the traffic sequences, and achieves comparable results on the checkerboard and articulated motion sequences. The FLoSS error median is typically low; however, some large errors do occur, most frequently as a consequence of choosing the wrong subspace dimensionality. This is illustrated in Fig. 6.5, which shows the first 3 principal components of data corresponding to the checkerboard sequence shown in Fig. 6.4. Here, instead of a higher-dimensional subspace, FLoSS chooses two lower-dimensional subspaces embedded in it. GPCA and LSA correctly group the two embedded subspaces. On the other hand, FLoSS outperforms other methods on data that contains two disjoint parts of the same subspace, such as the data shown in Fig. 6.6, corresponding to the traffic sequence shown in Fig. 6.4. In this case, LSA
fails due to the non-local structure, and GPCA fails because very few points lie on two of the three groups. Such cases occur more frequently in traffic data when a large number of keypoints are detected on disjoint pieces of the background (due to, for example, trees and grass), in contrast to only a few keypoints per car. In comparison to the other non-motion segmentation specific methods (RANSAC, mpPCA, and GPCA) FLoSS is either better (the traffic and articulated motion data for 3 objects), or performs very closely to the best method (GPCA for 2 objects checkerboard and articulated motion, mpPCA for 3 objects checkerboard).

With respect to run time, the algebraic methods GPCA and LSA are much faster than the iterative methods RANSAC, mpPCA, MSL, and FLoSS. Among iterative methods FLoSS is the slowest, and the number of iterations it requires to converge typically depends on the number of facilities it is initialized with. Its run time can potentially be improved through simple steps like pruning the initial set of facilities prior to passing messages, or selecting the initial set strategically, e.g. using RANSAC.

![Figure 6.5: Checkerboard sequence, first 3 principal components. (a) Ground truth, (b) FLoSS, (c) GPCA, and (d) LSA](image)

**6.3 Discussion**

In this chapter, we described a new subspace segmentation method that discovers linear subspaces in data using a message passing algorithm. We demonstrated its advantages over other methods on synthetic geometrical data, and evaluated its performance on
Figure 6.6: Traffic sequence, first 3 principal components. (a) Ground truth, (b) FLoSS, (c) GPCA, and (d) LSA

<table>
<thead>
<tr>
<th></th>
<th>Error</th>
<th>RANSAC</th>
<th>mpPCA</th>
<th>GPCA</th>
<th>FLoSS</th>
<th>LSA</th>
<th>MSL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Checkerboard</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>6.52</td>
<td>9.89</td>
<td>6.09</td>
<td>7.70</td>
<td><strong>2.57</strong></td>
<td>0.27</td>
<td>4.46</td>
</tr>
<tr>
<td>Median</td>
<td>1.75</td>
<td>2.49</td>
<td>1.03</td>
<td>1.23</td>
<td>0.27</td>
<td><strong>0.00</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Traffic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>2.55</td>
<td>21.41</td>
<td>1.41</td>
<td>0.14</td>
<td>5.43</td>
<td>2.23</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>0.21</td>
<td>17.61</td>
<td><strong>0.00</strong></td>
<td><strong>0.00</strong></td>
<td>1.48</td>
<td><strong>0.00</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Articulated</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>7.25</td>
<td>25.13</td>
<td><strong>2.88</strong></td>
<td>4.69</td>
<td>4.10</td>
<td>7.23</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>2.64</td>
<td>19.44</td>
<td><strong>0.00</strong></td>
<td>1.30</td>
<td>1.22</td>
<td><strong>0.00</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Motion segmentation percent error, 2 objects

multi-body motion segmentation from video. There are several possible directions for extensions and improvements of the developed FLoSS algorithm, such as adopting a more strategic approach for choosing the candidate subspaces, and refining the choices using, for instance, PCA on points assigned to each of the subspaces.

<table>
<thead>
<tr>
<th></th>
<th>Error</th>
<th>RANSAC</th>
<th>mpPCA</th>
<th>GPCA</th>
<th>FLoSS</th>
<th>LSA</th>
<th>MSL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Checkerboard</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>25.7</td>
<td>15.44</td>
<td>31.95</td>
<td>16.45</td>
<td><strong>5.80</strong></td>
<td>10.38</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>26.01</td>
<td>12.71</td>
<td>32.93</td>
<td>16.79</td>
<td><strong>1.77</strong></td>
<td>4.61</td>
<td></td>
</tr>
<tr>
<td><strong>Traffic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>12.83</td>
<td>37.02</td>
<td>19.83</td>
<td><strong>0.29</strong></td>
<td>25.07</td>
<td>1.80</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>11.45</td>
<td>30.89</td>
<td>19.55</td>
<td><strong>0.00</strong></td>
<td>23.79</td>
<td><strong>0.00</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Articulated</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>21.38</td>
<td>53.12</td>
<td>16.85</td>
<td>8.51</td>
<td>7.25</td>
<td><strong>2.71</strong></td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>21.38</td>
<td>53.12</td>
<td>16.85</td>
<td>8.51</td>
<td>7.25</td>
<td><strong>2.71</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Motion segmentation percent error, 3 objects
Chapter 7

Conclusions and Future Directions

In this thesis, we described a new approach to solving discrete facility location problems, namely that of using message passing inference in probabilistic graphical models. In Chapter 2, we listed the most common facility location problems and interpreted several important machine learning problems as facility location instances. For the uncapacitated family of problems, we also provided a brief overview of known complexity and approximability results, and approaches for constructing approximation algorithms for metric UFL. Chapter 2 also provided background on message-passing algorithms for MAP inference in graphical models. We described both the widely used max-product algorithm and the more recent max-product linear programming.

In Chapter 3, we showed the graphical models and max-product inference algorithms for different variants of the FL problems. Using these algorithms, we were able to generalize the Affinity Propagation algorithm for exemplar-based clustering to include prior beliefs and constraints on the number of clusters and their granularity. All graphical models in Chapter 3 contain loops and there are in general no guarantees on the optimality of their solution; however, they can be used as efficient heuristics.

In Chapter 4, we used the MPLP algorithm to find UFL solutions. Augmenting MPLP with a greedy heuristic allowed us to give sufficient conditions for solution optimality
and worst-case performance guarantees. Our method also provided some insights into constructing integral solutions from MPLP fixed points. We showed that the traditional variable assignment based on maximizing beliefs simply corresponds to satisfying one particular subset of complementary slackness conditions, and that a strategic choice of a different subset may prove empirically beneficial or provide optimality guarantees.

In Chapter 5, we performed an experimental evaluation of message passing algorithms on a number of UFL benchmarks, and compared the results to those obtained by LP-based and local search algorithms.

In Chapter 6 we applied the developed algorithms in the context of discrete multiple model selection. We described an algorithm called FLoSS, which discovers multiple low-dimensional linear subspaces in high-dimensional data by posing the problem as an instance of facility location. We evaluated FLoSS on synthetic data sets and applied it to the problem of motion segmentation in video sequences.

This work opens interesting directions for future research in both optimization and applications. Although there has been much work in developing LP-based message passing algorithms, there are no principled methods for assigning variables when LP solutions are fractional, except in certain special families of graphs. Our approach provides one possible direction - that of examining all of the complementary slackness conditions an optimal integral solution would satisfy. In terms of applications, an interesting direction is in constructing discrete formulations of mixture modeling problems and tackling them using combinatorial optimization.
Bibliography


[54] C. Lund and M. Yannakakis. On the hardness of approximating minimization prob-

[55] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric
facility location problems. In *Proc. 5th International Workshop on Approximation

[56] M. Mahdian, Y. Ye, and J. Zhang. Approximation algorithms for metric facility

[57] A. Manne. Plant location and economy of scale decentralization and computation.


Sons, Inc., 1990.


