Ballroom Matchmaker

Dan Fu and Kevin Fei
{dfu,kevinfei}@college.harvard.edu

Abstract—**Every year, the captain of the Harvard Ballroom Dance Team must assign newcomers to partnerships that will lead to better dancing and a better community. This is a tedious and time-intensive task that suggests an obvious computational approach. We attack this problem as a local search problem with a complex cost function. Our program successfully returns a matching that satisfies nearly all constraints and even proposes some prescient pairings.**

I. Introduction

Ballroom dancing is a popular collegiate activity with local competitions attracting hundreds of students. The Harvard Ballroom Dance Team [5] counts over 100 members and teaches its members 4 styles and a total of 19 dances. One peculiarity of this activity is that "it takes two to tango" so every dancer needs a partner for each style. In each partnership, there is a leader and a follower that defines the role of each dancer.

Finding partnerships is stressful and difficult, so as officers of the team, it is our job to partner the new dancers on the team. Our goal is to create prosperous and compatible partnerships that result in well-performing couples and strengthen the overall ballroom community.

Traditionally, we have assigned these partnerships manually; this is a difficult and time-intensive task that often does not result in completely equitable treatment. In this project, we seek to construct a program that will satisfy the hard constraints of partnering while also attempting to match softer constraints such as honoring individuals' dance preferences and matching height differences.

There are two ways of looking at this problem: as a constraint satisfaction problem or an optimization problem. We define a set of constraints that must be satisfied in a successful matching; for example, it is important to keep the height difference between leader and follower within a certain range for some dances. One way of solving this problem is by formulating a multi-level CSP and selectively relaxing constraints to find a valid solution. We will discuss this approach briefly, but we found that it was lacking because it could not distinguish between good and bad matchings within a single level of the CSP. The other approach is to use local search with some sort of cost function to judge matchings. We found that this approach was much more effective, so we will primarily be discussing this approach in this paper.

This paper is structured as follows: in section II, we discuss related work and other approaches to this problem. In section III, we discuss a formal specification of our problem. In section IV, we discuss our approaches to this problem. In section V, we discuss our experimental methodology and the results of our algorithm. We conclude and discuss future work in section VI.

II. Related Work

To the best of our knowledge, this is the first attempt to apply local search or CSP approaches to this problem. Traditionally, this problem has been solved manually; a search for "Ballroom Partner Search" on a major search engine returns a number of websites that exist to help people find partners online [1]–[3], [7]. Even these existing approaches do not offer solutions to the problem of matching up a large number of dancers with little previous ballroom experience. Based on informal interviews conducted with captains of other collegiate teams, the standard approach to the problem is to make the matchings manually, or have the rookie dancers find partners themselves.

A search for "ballroom" on GitHub revealed that there has been one attempt to solve this problem algorithmically, albeit in a relatively simple way [6]. This approach simply applied a greedy algorithm to the problem, matching up as many partners as possible before giving up. It does not attempt to switch individuals' roles to ensure that everyone gets a chance to dance.

In problem spaces adjacent to this one, there are the matching problem, and a specific subcase of it, the maximum-flow problem on bipartite graphs. The general matching problem can be expressed as follows: given a general graph G, find a set of edges such that no two edges share a vertex. The bipartite version of the matching problem partitions all the vertices into two groups such that there is no edge between any two vertices that are in the same group. At first glance, this appears to be a powerful framework for solving our problem. However, there are a few dynamics that a bipartite maximum flow problem does not cover. In particular, we allow people to switch between leader and follower roles if it helps create a better overall matching; this dynamic is not covered in standard bipartite maximum flow problems because it would require vertices to switch between groups (and change their edge memberships along the way). Additionally, we impose the constraint that two dancers cannot dance with each other for more than one dance. This introduces a new set of dynamically-changing edge constraints to the maximum flow problem. To solve the problem with these additional constraints, we will have to explore different algorithmic approaches.

III. Problem Specification

In this section, we describe a formal specification of our problem.

Let us have a set of dances D that the team is dancing. Each $d \in D$ has the attribute of being height sensitive or not.

Let us have the set $L = \{\text{lead}, \text{follow}, \text{both}\}$ as the possible set of roles a desired dancer can have. Define $^\circ : L \to L$ where $\text{lead}^\circ = \text{follow}$, $\text{follow}^\circ = \text{lead}$, and $\text{both}^\circ = \text{both}$. This operation finds the opposite role of the one given.

Let us have the set of people P. Each $p \in P$ has the following attributes

1) Email
2) Desired Role: $l \in L$, the role the dancer prefers.
3) Nonballroom Dance Experience: $n \in \mathbb{N}$ (years)
4) Ballroom Dance Experience: $\{0, 1\}$
5) Lessons attended: $n \in \mathbb{N}$
6) Height: $n \in \mathbb{N}$ (inches)
7) Dances: $E \subset D$, the subset of dances the dancer wishes to participate in.
8) Dance Preferences: A dictionary R with key being a dance d. Its value is a list of people that this person would prefer to dance with for this dance. There is also a special field *nogo* that indicates the person's preference not to dance with anyone on this list.

The set P may also include a special placeholder TBA. This special person is matched up with other dancers only in a worst case scenario where we are unable to pair certain dancers. TBA is willing to dance all dances in either role.

Let $Q \subset P \times D$ such that $(p, d) \in Q$ if $d \in p.E$.

Given this information, we must find a partnering of each dancer in each dance. We wish to match every $q \in Q$ with $(r, l) \in Q \times L$ where r is p's partner and l represents p's role in the partnership.

A. Constraints

We can't simply partner all our newcomers willy-nilly. There are many constraints we must try to follow, some of which are essential while others serve to create a better dancing experience for everyone.

1) Everyone has exactly one partner:
We want all our dancers to dance with each other (no solos or triumvirates). We want to make sure that dancers who aren't paired for one dance have partners for all their other dances.
Each (p, d) pair in Q must be assigned to exactly one other pair. The special value TBA is not used unless necessary (formally, we do not include TBA in P until we have no other choice).

2) Everyone has a different partner:
We want our dancers to meet a lot of people on the team. The more connections each dancer has, the stronger the team will be. This also helps people find people they enjoy dancing with.
If two people p and q are paired for some dance d, they cannot be paired for any other dance.
3) Leader/follower preference:
Dancers can either learn to lead or follow a dance. Generally males lead and females follow, but either gender may choose to do both. Each role has different steps so switching roles is often a very difficult transition. We also wish for dancers to have partners with their preferred role. It is much smoother dancing with someone who has learned and practiced their steps.
Each (p, d) pair must be assigned to (q, d, l) such that both p and q dance their preferred role. That is $l = p.\text{role}$ and $l^{\circ} = q.\text{role}$.
4) Height Similarity:
In general, dancers of similar height will look better and dance better. This matters more in some dances more than others. That is why they are designated height sensitive.
In a height sensitive dance, for some pair (p, d) assigned (q, d, l), we have if the leader in this partnership p has $p.\text{height} \in [q.\text{height} + 2, q.\text{height} + 7]$.
5) Partner preference:
During the lessons, we are sure to switch all the dancers around so that they have a chance to dance with everyone. Certain people may discover that they are compatible and wish to dance together. We try to honor these requests, but sometimes the need to pair all dancers may override a preference. There are also cases when we want to avoid uncomfortable situations by avoiding certain partnerships.
We examine each $q \in p.R$. If $p \in q.R$, then we preference the matching. If multiple such matches exist, we pick one randomly. We ignore one-sided partner preferences.
6) Attendance/past experience:
Not all our dancers have the same experience coming into the lessons. Some have had years of ballet, traditional, or hip-hop dance experience. Others have dabbled in some ballroom or social dancing. This form is mainly for newcomers to ballroom, so past ballroom experience is expected to be relatively little. Of course, none of this matters if they do not attend lessons! The most important dance to remember is atten-DANCE! Regular attendance ensures the partners are able to easily communicate and dance with each other.
For each (p, d) paired with (q, d), p and q should have similar past dance experiences. That is, $p.\text{past ballroom} = q.\text{past ballroom}$ and $p.\text{past dance} \in [q.\text{past dance} - 2, q.\text{past dance} + 2]$. They should also have similar lesson attendance rates where $p.\text{attendance} \in [q.\text{attendance} - 3, q.\text{attendance} + 3]$.

IV. Approach

A. Multi-Tiered CSP

The constraints expressed in section III-A suggest a fairly natural CSP approach to this problem. Indeed, this was our initial idea, and the one that we pursued for a good deal of our project. However, we found that this algorithm was not able to distinguish between good and bad matchings without a cost function. Once we had a cost function, local search was a much better candidate. In this section, we will briefly discuss the multi-tiered CSP approach for the sake of completeness with respect to our project proposal. However, we will not discuss it in detail beyond this section.

In the multi-tiered CSP approach, we first try to solve the CSP problem with all constraints, and then we iteratively try again with one fewer constraint if we cannot find a solution. The total search space for a single iteration of the CSP problem is $\boldsymbol{O}(|P|!^{|}D|)$. With 60 dancers and four dances, this is more possibilities than there are atoms in the universe, so it is important to not try all possibilities. To make the search space smaller, we use arc consistency to reduce the number of states that we have to try. In this approach, we explicitly store the domain of possible partnerships for each dancer, and apply constraints one at a time. Each time a constraint is applied, the size of the domain of each dancer drastically decreases. If the domain size of any dancer ever reaches 1, we make the assignment. If the domain ever reaches 0, we return failure for this tier.

Once all the constraints have been applied, we start making assignments from each dancer's domain. When we make an assignment, we remove that dancer and the dancer's new partner from everyone else's domain. If the constraint that everyone has a different partner for each dance is active, we also remove the dancer and the partner from each other's domain for all the other dances. If any dancer's domain is ever empty, we go back and undo the last assignment and pick another assignment (and continue like this in a DFS fashion).

For real data sets, this algorithm usually cannot find solutions that satisfy constraints $3-6$. This is simply a numbers game; there are usually more followers than leaders, and it is difficult to match people who are either very tall or very short. This algorithm can usually quickly find a solution that satisfies constraints 1 and 2, given that there are more than $D+1$ dancers. However, these matchings are not useful; with constraints $3-6$ relaxed, the algorithm creates many partnerships where individuals are not dancing their preferred role, and many more where there is a poor height match. In other words, with constraints $3-6$ relaxed, the algorithm has no knowledge of which matchings are better than others.

B. *Local Search*

In response to the problems with the multi-tiered CSP approach, we have decided on a new approach to the problem: local search. We create a multi-tiered cost function where failing to satisfy certain constraints will cost orders of magnitude more than others. This allows us to replicate the structure of a multi-tiered CSP but with one uniform metric. One major advantage of using the local search approach is that there is a starting point for a matching, that is the preferences of the dancers. It is much more efficient to incorporate this into the initial matching than attempting to recreate them through random pairings.

We begin with assigning an initial pairing. Here we seek to match any mutual partner preferences and randomly assign all leftover leaders to followers. Often there are leftover followers and we will randomly switch some followers to lead in order to balance the matching. This initial matching will instantiate any TBA's we need.

After that we engage in a local search with simulated annealing in an attempt to find a minimum cost state. Pseudocode for this algorithm is given in algorithm 1. We repeatedly loop until we reach a specified number of steps. Every iteration, we choose a random neighbor in the matching state space. We do this by either switching two different partnership's leaders or switching the roles of leader and follower in a partnership. Evaluating our new state, we may move to the new state if it is better. If it is non-optimal, we move to that state with probability $e^{-(\text{new cost}-\text{old cost})/\text{temperature}/\text{initial cost}}$ where initial cost is the cost of the initial matching.

An important step of increasing the efficiency of this algorithm is to store which states we have already visited. We do this with a cache implemented by a hash-table. We begin by turning each matching into a string by serializing all the partnerships. This allows a uniform and compact way to store all the information in a matching. We create a dictionary in which the serialized string is the key and the value is the associated cost. For a smaller problem, we may exhaust all possible neighbors and stop the execution before the full number of steps.

V. Experiments

A. *Implementation*

To make our implementation easy to host on GitHub Pages and easy for a non-technical population to use, we implemented our algorithm in JavaScript and made the interface a simple static HTML page. The user uploads a tab-separated values file with all the relevant information, and the page outputs a matching, as well as the score for that matching generated from the cost function.

Algorithm 1 Local Search with Simulated Annealing

```
procedure LocalSearchMatch(steps)
    $T \leftarrow 1000, stepnum \leftarrow 0$
    Current state ← Initialize matching
    while $stepnum < steps$ and a new neighbor exists do
        New state ← New random neighbor
        Cache the new state
        if New state has a lower cost OR new state has a higher cost and $p$ <Prob(New state, Current state) then
            Current state ← New state
        end if
        $stepnum \leftarrow stepnum + 1, T \leftarrow \alpha T$
        ▷ $\alpha$ is our temperature falloff constant
    end while
    return Current state
end procedure
```

When the file is uploaded, the JavaScript code processes it and creates an abstract representation of all the dancers. Next, we generate an initial matching that first matches up all partners that have listed each other in their preferences, then matches up available leaders and followers, and finally matches up the other dancers by switching roles if necessary.

To make random steps for our local search, we first randomly pick a dance to make a change in. With 50% chance, we then switch two leaders in the matching for that dance. With 50% chance, we switch a leader with a follower in another couple; the leader becomes the follower in the other couple, and the follower becomes the leader in the leader's couple. We do not count steps that do not generate new matchings in our step count; if our random step generates a state we have already visited, we simply generate a new random step from the original state.

Other details are listed in Appendix A.

B. Data

We test our implementation on two historical data sets, HBDT14 and HBDT16, from the Harvard Ballroom Dance Team. These data sets were curated from surveys given to first-year

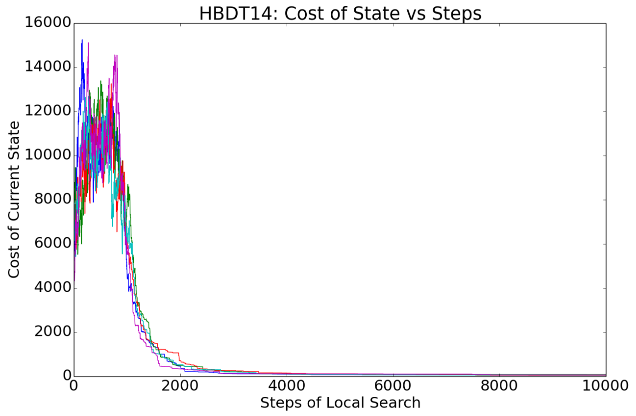

Fig. 1: Graph of the cost of the current state vs. number of steps for five runs of the local search algorithm on the HBDT14 data set.

dancers in preparation for their first competition, where the dancers dance four dances. The former data set was collected in October 2014 and contains 48 dancers. The latter was collected in October 2016 and contains 33 dancers. In 2015, the team captain elected to have the rookies find their own partnerships. Both of these data sets contain information about height, leader/follower preference, and attendance/past experience. However, only the HBDT16 data set contains partner preference information, since the captain in 2014 elected not to take it into account.

C. Experimental Setup

We ran our local search algorithm for $10,000$ steps on both data sets. We ran our experiments on a Linux machine with an intel dual-core processor clocked at 2.0 GHz.

D. Results

Figures 1 and 2 show graphs of the cost over time for multiple runs of the algorithm on the two data sets. Table I shows the average runtimes for these two data sets across the five runs.

For both data sets, $10,000$ steps is more than enough to converge to a steady cost. The HBDT14 took longer to run and converged to a higher cost (non-negative). These results are expected given the differences between the two

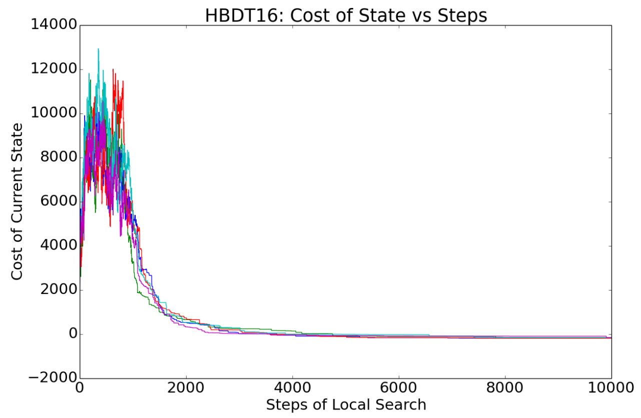

Fig. 2: Graph of the cost of the current state vs. number of steps for five runs of the local search algorithm on the `HBDT16` data set.

	Average Time (s)
`HBDT14`	14.17
`HBDT16`	6.30

TABLE I: Average runtimes for our local search algorithm for the two data sets across five runs.

data sets. `HBDT14` is larger than `HBDT16`, so we expect it to take longer to process computationally. Furthermore, since there are no preferences in `HBDT14`, there are no opportunities to reduce the cost by making matchings that match individuals' preferences.

Besides these metrics, we also looked at the final matchings that our algorithms generated from a qualitative standpoint. The overall matchings were quite impressive; even though there were many more followers than leaders in both data sets, no follower had to lead more than one dance. In addition, only one or two followers that led one dance had to follow someone who would have preferred to follow; in other words, every dancer had a majority of dances where both partners were dancing their preferred role.

In addition, the program consistently suggested some partnerships that were superior to the ones made by team captains, even without the proper preferences being included in the data, despite the inherent randomness of the local search algorithm. For example, one of the authors of this paper, `Dan Fu`, is included in the `HBDT14` data set. Team captains partnered him with `Jenni Haydek` for his first year of dancing. His second year, he and `Serena Wang` partnered with each other and did very well. The program consistently suggests than `Dan Fu` and `Serena Wang` be partnered together, even though the partner preferences do not express this information explicitly in the data set. Although events such as these matching are hard to quantify, they show that this approach has promise.

VI. Conclusion and Future Work

In this paper, we have presented a program to automatically create successful and long-lasting ballroom partnerships given a large population of dancers with little to no dance history. This is a version of the classical matching and bipartite maximum flow problems, but with additional constraints because of the nature of the problem. As a result, we have to explore new algorithmic approaches to make the problem tractable while still producing good results. We initially thought we could approach this as a multi-tiered CSP, where we start by searching the state space with all constraints, and then iteratively relax constraints if we do not find a solution. While this always allowed us to find a solution, it was rarely a very good one; with the weaker constraints relaxed, the solutions lose any way to judge which matchings are better than others.

Instead, we have formulated the problem as an optimization problem with a cost function based on the constraints we had previously identified. We solve this problem using local search with simulated annealing. We have tested our program on two data sets, `HBDT14` and `HBDT16` and found that the program successfully creates strong matchings, sometimes better than those created by hand, for both data sets.

In the immediate future, we would like to make our program more user-friendly by making it easier to input data and to view the outputted matchings. We would also like to make it possible for users to generate matchings that are meaningfully different from the final matching outputted (if the user does not like the final matching and wants a completely different one, for instance). Our ultimate goal is for the collegiate ballroom community to use our tool, so our short-term work will be aimed at making that goal a reality.

In the long-term, we would like to make our tool smarter by trying to predict how well couples will do, or how long they will stay together, given the limited information we can collect before the rookies' first competition. This will require curating new data sets or augmenting old ones with known information. A first step might be predicting how couples with existing competition histories will do against other couples in future competitions.

Acknowledgments

We would like to thank the Harvard Ballroom Dance Team for providing us with the data for our analysis. We would also like to thank the team captains and advanced dancers we interviewed for providing their insights into constructing this tool. Finally, we would like to thank Professor Scott Kuindersma and the course staff of CS 182 for giving us the tools to solve this problem.

Appendix
A: System Description

A. Cost Function

1) Everyone has a partner: For every dancer, cost c = # of TBA dances $*1000/$(# of non-TBA dances).
2) Everyone has a different partner: For every dancer, cost is $c = 50*$(# of dances they dance with a partner $-1)^2$ for each partner they dance with.
3) Leader/follow preference: For every dancer, cost is $c = 50 *$ (# of dances they dance in the opp role$)^2$.
4) Partner Lead/follow preference: cost is $c = 10 *$ (# of dances with partner opposite preference) $*$ (D/# of their dances).
5) Height difference: For height sensitive dances; calculate the height difference as the leader's height minus the follower's height. Then $c = -10$ if the difference is 4-5 inches, $c = 0$ if 2-7 inches, $c = +10$ if is outside that range.
6) Partner Preferences: $c = -30$ if matched correctly (two ways)! $c = +50$ if someone is matched with someone they dislike.
7) Past experience: $c = -2$ if both have ballroom experience or within 2 years of nonballroom experience.
8) Lesson attendance: $c = -2$ if both have attended the same number of lessons plus or minus 2.

B. Running Ballroom Matchmaker

1) Creating the tsv: The input file that the program takes is a tsv (tab separated value) with the following format.

1) Line 1: $|D|, D$
   Tab separated: the number of dances $|D|$, followed by a list of dance name, followed by its height sensitivity (1/0).
2) Line 2: $|P|$ The number of people P in the system
3) Lines 3 to $3+|P|$: Tab separated list of
   a) Name
   b) e-mail
   c) height (in inches)
   d) comma-separated list of dances they want to dance
   e) lead/follow/both preference (one for all dances)
   f) time spent dancing (integer in hours/week)
   g) lessons attended
   h) ballroom experience (years)
   i) non-ballroom experience (years)
   j) comma-separated list of people they don't want to dance with
   k) comma-separated lists of people they do want to dance with for each dance (in the order given in line one)

There are examples included in the examples/ directory.

2) Creating the matching: Click on the button in the middle of the screen to upload your .tsv. The program will run client-side; currently, there is no visual indication that the program is running, but the program does print out intermediate cost values to the JavaScript console. After running for a few seconds, a link will allowing you to download an indented list enumerating each dancer followed by each of their partnerships.

3) Miscellaneous: The code is available at https://github.com/DanFu09/ballroom-matchmaker. It is being hosted with GitHub Pages at https://danfu09.github.io/ballroom-matchmaker.

Appendix
B: Group Makeup

1) Dan Fu
    a) Conducted background research
    b) Curated data sets
    c) Determined the method to solve this problem.
    d) Implemented initial matching, random neighbor function, display of data, and cache
2) Kevin Fei
    a) Formalized problem description
    b) Implemented file parsing, cost function, local search with simulated annealing
    c) Ran tests on data
    d) Implemented the frontend and website

References

[1] Ballroom & Latin Dance Partner Search. https://www.facebook.com/BallroomLatinDancePartnerSearch/.

[2] DanceSportInfo Partner Search. http://www.dancesportinfo.net/PartnerWantedSearchY.aspx.

[3] D. Fu. Ballroom-Partner-Finder. https://github.com/DanFu09/Ballroom-Partner-Finder.

[4] GitHub Pages. https://pages.github.com/.

[5] Harvard Ballroom Dance Team. http://www.harvardballroom.org/.

[6] G. Larson. Partner-Search. https://github.com/gnarph/Partner-Search.

[7] Partner Search Classifieds. http://www.ballroomdancers.com/Classifieds/Partners/.