Exact and Approximation Algorithms for the Maximum Constraint Satisfaction Problem over the Point Algebra

Yoichi Iwata[1] **and Yuichi Yoshida**[2]

1 **University of Tokyo**
7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan
y.iwata@is.s.u-tokyo.ac.jp

2 **National Institute of Informatics and Preferred Infrastructure, Inc.**
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan
yyoshida@nii.ac.jp

Abstract

We study the constraint satisfaction problem over the point algebra. In this problem, an instance consists of a set of variables and a set of binary constraints of forms $(x < y), (x \leq y), (x \neq y)$ or $(x = y)$. Then, the objective is to assign integers to variables so as to satisfy as many constraints as possible. This problem contains many important problems such as Correlation Clustering, Maximum Acyclic Subgraph, and Feedback Arc Set.

We first give an exact algorithm that runs in $O^*(3^{\frac{\log 5}{\log 6}n})$ time, which improves the previous best $O^*(3^n)$ obtained by a standard dynamic programming. Our algorithm combines the dynamic programming with the split-and-list technique. The split-and-list technique involves matrix products and we make use of sparsity of matrices to speed up the computation.

As for approximation, we give a 0.4586-approximation algorithm when the objective is maximizing the number of satisfied constraints, and give an $O(\log n \log \log n)$-approximation algorithm when the objective is minimizing the number of unsatisfied constraints.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Constraint Satisfaction Problems, Point Algebra, Exact Algorithms, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.127

1 Introduction

Problems involving temporal constraints arise in various areas of computer science such as scheduling, program verification, and parallel computation. One of the most common frameworks to express temporal constraints is temporal constraint satisfaction problems (Temporal CSPs). In Temporal CSP, an instance consists of a set of variables and a set of constraints defined by first-order sentences with the predicate $(<)$, the strict total order of integers. Then, the objective is to assign integers to variables so as to satisfy all the constraints.[1]

One of most famous Temporal CSPs is the CSP over the point algebra, introduced by Vilain and Kautz [18]. In this problem, we have constraints of forms $(x < y), (x \leq y), (x \neq y)$

[1] We often choose the domain as the set of rational numbers and the predicate $(<)$ as the dense strict total order of rational numbers (e.g., [5]). However, we choose the domain as the set of integers to simplify our expositions.

and $(x = y)$. A considerably larger class of Temporal CSPs is the CSP over the Ord-Horn relations, introduced by Nebel and Bürkert [13]. CSPs over the point algebra and over Ord-Horn relations are known to be solvable in polynomial time [13, 18]. Also, Ordering CSPs introduced in [9] can be formulated as Temporal CSPs in which all variables must be assigned different integers.

Most existing works on Temporal CSPs are concerned with the problem of deciding whether all constraints are satisfied [1, 5, 12, 18]. However, it is natural to ask for an assignment that satisfies as many constraints as possible when all constraints are not satisfied simultaneously. In this paper, we are especially interested in the CSP over the point algebra as it is the most fundamental Temporal CSP. We can look at the problem in terms of maximizing the number of satisfied constraints (Max-PA), or in terms of minimizing the number of unsatisfied constraints (Min-PA). These two are equivalent at optimality but, as usual, differ from the point of view of approximation.

First, we give an exact algorithm for Max-PA (and hence for Min-PA) running in $O^*(3^{\frac{\log 5}{\log 6}n})$ time, where n is the number of variables.[2][3] This result improves the current best $O^*(3^n)$ obtained by a standard dynamic programming. Our algorithm is obtained by combining the dynamic programming with the split-and-list technique due to Ryan Williams [19]. That is, we reduce computation of the dynamic programming to computation of matrix products. The reduction is not trivial since the original split-and-list technique is only used to speed-up exhaustive search. Using the current fastest algorithm for multiplying general square matrices by Vassilevska Williams [20], we obtain an algorithm that runs in $O^*(3^{\frac{\omega}{\log 6}n})$ time, where $\omega < 2.3727 < \log 6$. However, one of the matrices generated by the reduction is sparse and has some recursive structure. To make use of this property, we modify the algorithm due to Bini et al. [3] and get an algorithm that runs in $O^*(3^{\frac{\log 5}{\log 6}n})$ time.

Next, we give a 0.4586-approximation algorithm for Max-PA. The idea of our algorithm is similar to [16]. We first solve a semidefinite relaxation and round the solution using three or four hyperplanes (we take the better one). If two variables are in the same side for every hyperplane, they will get the same value. Thus, we use at most 16 values. The ordering of values assigned to different clusters is chosen randomly.

If we only use constraints of the form $(x < y)$, then Max-PA coincides with Maximum Acyclic Subgraph, in which we want to find an ordering of vertices in a digraph so as to maximize the number of edges that go forward. It is NP-Hard to get a $(0.5+\epsilon)$-approximation for any $\epsilon > 0$ assuming Khot's unique games conjecture [10, 11]. The hardness suggests that it would be difficult to improve our approximation ratio significantly.

Finally, we give an $O(\log n \log\log n)$-approximation algorithm for Min-PA. If we only use constraints of the form $(x < y)$, then Min-PA coincides with Feedback Arc Set, in which we want to find an ordering of vertices in a digraph so as to minimize the number of edges that go backward. The best algorithm for Feedback Arc Set has an approximation ratio $O(\log n \log\log n)$ [8]. Thus, our algorithm can be seen as a generalization of the algorithm for Feedback Arc Set. The idea of our algorithm is reducing the problem to a variant of multicut problem, which we call the *symmetric multicut problem*. In this problem, we are given a digraph $G = (V, E)$, and a set of terminal pairs $T = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. Then, we want to find an edge set $F \subseteq E$ of minimum cardinality such that, for any terminal pair $(s, t) \in T$, $G - F$ contains no path from s to t or no path from t to s. Then, we obtain an $O(\log n \log\log n)$-approximation algorithm for the symmetric multicut problem.

[2] $O^*(\cdot)$ hides a factor polynomial in n.
[3] We denote by log the logarithm to the base 2.

1.1 Related Works

By restricting types of constraints further, the CSP over the point algebra coincides with many other problems. If we use constraints of the form $(x = y)$ and $(x \neq y)$ only, then we get Correlation Clustering [2]. As far as we know, exact algorithms for Correlation Clustering is not studied in the literature. However, we can easily obtain an $O^*(2^n)$-time algorithm by a simple application of fast subset convolution by Björklund et al. [4] (see Section 3 for detail). If the underlying graph is a complete graph, then the maximization version admits PTAS and the minimization version can be approximated within a factor of 4 [6]. For general underlying graphs, the maximization version can be approximated within a factor of 0.7666 [16] and the minimization version can be approximated within a factor of $O(\log n)$ [6].

If we use constraints of the form $(x < y)$ only, then Max-PA coincides with Maximum Acyclic Subgraph and Min-PA coincides with Feedback Arc Set. The current fastest exact algorithm for Maximum Acyclic Subgraph (and hence Feedback Arc Set) is a simple $O^*(2^n)$-time dynamic programming. The current best approximation algorithm for Maximum Acyclic Subgraph is just the random assignment and its approximation ratio is $1/2$. As we mentioned, Guruswami et al. [10] showed that it is NP-Hard to obtain $(1/2 + \epsilon)$-approximation for any $\epsilon > 0$ assuming Khot's unique games conjecture. As for Feedback Arc Set, there is an $O(\log n \log\log n)$-approximation algorithm [8]. It is known that obtaining 1.36-approximation is NP-Hard [7] and obtaining any constant approximation ratio is NP-Hard assuming Khot's unique games conjecture [10].

A Temporal CSP with a single predicate is called an Ordering CSP if all variables must be assigned different values. Guruswami et al. [9, 10] considered the maximization version of Ordering CSPs, and they showed that the random assignment always gives the best approximation ratio assuming Khot's unique games conjecture.

1.2 Organization

We give definitions used in this paper in Section 2. In Section 3, we show an $O^*(3^{\frac{\log 5}{\log 6}n})$-time exact algorithm for Max-PA. Sections 4 and 5 are devoted to show a 0.4586-approximation algorithm for Max-PA and an $O(\log n \log\log n)$-approximation algorithm for Min-PA, respectively.

2 Preliminaries

For an integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$. A *(d-ary) relation* over a domain $[k]$ is a subset of $[k]^d$, and a *(d-ary) constraint* is a pair of a tuple of d variables and a d-ary relation. A constraint $e = (\{x_1, \ldots, x_t\}, R)$ is called *satisfied* by an assignment f if $(f(x_1), \ldots, f(x_t)) \in R$. Now, we define two problems, Max-PA, and Min-PA.

Max-PA
Input: A set of n variables V and a set of m constraints C. Each variable $x \in V$ takes value from $[n]$, and each constraint is of the forms $(x < y), (x \leq y), (x \neq y)$ and $(x = y)$.
Output: An assignment $f : V \to [n]$ that maximizes the number of satisfied constraints.

Min-PA
Input: Same as Max-PA.
Output: An assignment $f : V \to [n]$ that minimizes the number of unsatisfied constraints.

Since the number of unsatisfied constraints is the number of constraints minus the number of satisfied constraints, the optimal assignments for the two problems coincide. Thus, for the exact algorithm, we deal with Max-PA only. In this paper, we only deal with unweighted instances, but our argument can be easily extended to weighted instances, for which we want to maximize (resp., minimize) the total weight of satisfied (resp., unsatisfied) constraints. If weights are integers between $-W$ and W, then the running time of our exact algorithm takes additional $O^*(W)$ factor. Running times of our approximation algorithms do not change.

3 Exact Algorithms

In this section, we give an exact algorithm for Max-PA and prove the following theorem.

▶ **Theorem 1.** *Max-PA can be solved in $O^*(3^{\frac{\min(\omega,\log 5)}{\log 6}n})$ time and $O^*(3^{\frac{2}{\log 6}n})$ space, where ω is the matrix product exponent.*

We cannot compare ω and $\log 5$ since we do not know the true value of ω. The current best bound on ω is 2.3727 by Vassilevska Williams [20], which is larger than $\log 5 < 2.3220$.

Before proving Theorem 1, we first show an $O^*(2^n)$-time algorithm for Correlation Clustering to see the difficulty of Max-PA. We formalize Correlation Clustering as a dynamic programming and solve it using fast subset convolution to achieve the time complexity $O^*(2^n)$. This can be done because of the simplicity of the recurrence in the dynamic programming.

Then, we introduce a standard $O^*(3^n)$-time dynamic programming algorithm for Max-PA. We will see that we cannot apply fast subset convolution to the recurrence. This is the point we become apart from Correlation Clustering, and we improve the running time of the algorithm to $O^*(3^{\frac{\omega}{\log 6}n})$ by applying the split-and-list technique to compute the recurrences. Finally, we further improve the running time by using structure of matrices involved when applying the split-and-list technique and give an $O^*(3^{\frac{\log 5}{\log 6}n})$-time algorithm.

3.1 Algorithm for Correlation Clustering

We explain an $O^*(2^n)$-time dynamic programming algorithm for Correlation Clustering. Recall that Correlation Clustering is a special case of Max-PA such that each constraint has the form $(x = y)$ and $(x \neq y)$ only. First, we reduce an instance of unweighted Correlation Clustering into an instance of weighted Correlation Clustering that has $(=)$-constraints only. This can be done by replacing each constraint of the form $(x \neq y)$ by a constraint $(x = y)$ with weight -1. If an assignment satisfies a removed constraint $(x \neq y)$, then it does not satisfy the added constraint $(x = y)$ and contributes to the objective value of the reduced instance by 0. Otherwise it satisfies the added constraint and contributes to the objective value by -1. Thus the difference of its contribution to the original instance and the reduced instance is always a fixed constant. Therefore, the optimal assignment does not change through the reduction.

Then, we solve the reduced instance by dynamic programming. For a subset $S \subseteq V$, we define $\mathrm{dp}_i(S)$ as the maximum total weight of satisfied constraints by assigning values from $[i]$ to S. When some variable in a constraint is not assigned any value, we simply regard that the constraint is not satisfied. We define $\mathrm{dp}_0(\emptyset) = 0$ and $\mathrm{dp}_0(S) = -\infty$ for any $S \neq \emptyset$. The optimal value can be obtained as $\mathrm{dp}_n(V)$. We can compute dp_{i+1} from dp_i by the following recurrence:

$$\mathrm{dp}_{i+1}(S) = \max_{T \subseteq S}\{\mathrm{dp}_i(T) + w(S \setminus T)\}, \tag{1}$$

where $w(S)$ is the total weight of the constraints of the form $(x = y)$ with $x \in S$ and $y \in S$. The running time of this dynamic programming is $O^*(\sum_{i=0}^{n} \binom{n}{i} 2^i) = O^*(3^n)$. Björklund et al. [4] showed that any recurrence of this form can be computed in $O^*(2^n)$ time by developing a technique called fast subset convolution. Thus, we can solve Correlation Clustering in $O^*(2^n)$ time.

3.2 Standard Dynamic Programming Algorithm

We explain an $O^*(3^n)$-time dynamic programming algorithm for Max-PA. First, we reduce an instance of unweighted Max-PA into an instance of weighted Max-PA that has $(<)$-constraints only. This can be done by the following reduction.

- For each constraint of the form $(x < y)$, we set its weight as 1.
- For each constraint of the form $(x \leq y)$, we replace it by a constraint $(x > y)$ with weight -1.
- For each constraint of the form $(x \neq y)$, we replace it by two constraints $(x < y)$ and $(x > y)$ with weight 1.
- For each constraint of the form $(x = y)$, we replace it by two constraints $(x < y)$ and $(x > y)$ with weight -1.

If an assignment satisfies a removed constraint $(x \leq y)$, then it does not satisfy the added constraint $(x > y)$ and contributes to the objective value of the reduced instance by 0. Otherwise it satisfies the added constraint and contributes to the objective value by -1. Thus the difference of its contribution to the original instance and the reduced instance is always a fixed constant. We have the same property for constraints $(x \neq y)$ and $(x = y)$. Therefore, the optimal assignment does not change through the reduction.

Then, we solve the reduced instance by dynamic programming. For a subset $S \subseteq V$, we define $\mathrm{dp}_i(S)$ as we did in the previous subsection. We can compute dp_{i+1} from dp_i by the following recurrence:

$$\mathrm{dp}_{i+1}(S) = \max_{T \subseteq S} \{\mathrm{dp}_i(T) + w(T, S \setminus T)\}, \tag{2}$$

where $w(A, B)$ is the total weight of constraints of the form $(x < y)$ with $x \in A$ and $y \in B$. The running time of this dynamic programming is $O^*(3^n)$ and it uses $O^*(2^n)$ space.

Compare to the recurrence (1) for Correlation Clustering, w in this recurrence does not have the form of $w(S \setminus T)$ but has the form of $w(T, S \setminus T)$ to which we cannot apply fast subset convolution. This is because, in order to determine whether a constraint is satisfied, we need to know not only how variables are partitioned into equal-valued sets, but also the ordering of those sets.

3.3 Split-and-List Algorithm for Max-PA

In this subsection, we improve the standard dynamic programming algorithm in the previous subsection by applying split-and-list technique. The original technique was developed to speed-up the exhaustive search for Max 2-SAT by Ryan Williams [19], and here, we demonstrate that it can be used to speed-up the computation of the recurrence in the dynamic programming. In the original technique for Max 2-SAT, we split the variable set into three equal-sized parts A, B, and C. Then we create two matrices X and Y from the number of satisfied clauses by each assignment on $A \cup B$ and $B \cup C$, respectively, so that we can obtain the maximum number of satisfied clauses from the product XY. In our application, we create two matrices X and Y from values of dp_i so that we can obtain values of dp_{i+1} from the product XY.

▶ **Lemma 2.** *Max-PA can be solved in* $O^*(3^{\frac{\omega}{\log 6}n})$ *time and* $O^*(3^{\frac{2}{\log 6}n})$ *space.*

Proof. We divide the variables into two parts V_1 and V_2 so that $\alpha|V_1| = |V_2|$, where $\alpha \geq 1$ is a parameter. Then, for $S_1 \subseteq V_1$ and $S_2 \subseteq V_2$, we can rewrite the recurrence of the dynamic programming (2) as follows.

$$\begin{aligned}
\mathrm{dp}_{i+1}(S_1 \cup S_2) &= \max_{T \subseteq S_1 \cup S_2} \{\mathrm{dp}_i(T) + w(T, (S_1 \cup S_2) \setminus T)\} \\
&= \max_{T_1 \subseteq S_1} \max_{T_2 \subseteq S_2} \{\mathrm{dp}_i(T_1 \cup T_2) + w(T_1 \cup T_2, (S_1 \cup S_2) \setminus (T_1 \cup T_2))\} \\
&= \max_{T_1 \subseteq S_1} \max_{T_2 \subseteq S_2} \{\mathrm{dp}_i(T_1 \cup T_2) + w(T_1, S_1 \cup S_2) + w(T_2, S_1 \cup S_2) \\
&\quad - w(T_1 \cup T_2, T_1 \cup T_2)\} \\
&= \max_{T_1 \subseteq S_1} \{w(T_1, S_1 \cup S_2) + \\
&\quad \max_{T_2 \subseteq S_2} \{\mathrm{dp}_i(T_1 \cup T_2) + w(T_2, S_1) - w(T_1 \cup T_2, T_1 \cup T_2) + w(T_2, S_2)\}\}.
\end{aligned}$$

We can reduce the computation of this recurrence into a product of the following two matrices X and Y with an indeterminate t.

$$\begin{aligned}
X_{(S_1,T_1),T_2} &= [T_1 \subseteq S_1]\, t^{\mathrm{dp}_i(T_1 \cup T_2) + w(T_2, S_1) - w(T_1 \cup T_2, T_1 \cup T_2)}, \\
Y_{T_2,S_2} &= [T_2 \subseteq S_2]\ t^{w(T_2,S_2)},
\end{aligned}$$

where $T_1, S_1 \subseteq V_1$ and $T_2, S_2 \subseteq V_2$. Also, $[\cdot]$ has value 1 if the condition inside is true and has value 0 otherwise. Indeed, the product XY can be expressed as

$$(XY)_{(S_1,T_1),S_2} = [T_1 \subseteq S_1] \sum_{T_2 \subseteq S_2} t^{\mathrm{dp}_i(T_1 \cup T_2) + w(T_2,S_1) - w(T_1 \cup T_2, T_1 \cup T_2) + w(T_2, S_2)}.$$

Thus, the value of $\mathrm{dp}_{i+1}(S_1 \cup S_2)$ coincides with the maximum degree of a non-zero term in

$$\begin{aligned}
&\sum_{T_1 \subseteq S_1} t^{w(T_1, S_1 \cup S_2)} (XY)_{(S_1,T_1),S_2} \\
&= \sum_{T_1 \subseteq S_1} t^{w(T_1, S_1 \cup S_2)} \sum_{T_2 \subseteq S_2} t^{\mathrm{dp}_i(T_1 \cup T_2) + w(T_2,S_1) - w(T_1 \cup T_2, T_1 \cup T_2) + w(T_2,S_2)} \\
&= \sum_{T \subseteq S_1 \cup S_2} t^{\mathrm{dp}_i(T) + w(T, S_1 \cup S_2 \setminus T)}.
\end{aligned}$$

The number of choices for S_1 and T_1 with $T_1 \subseteq S_1 \subseteq V_1$ is $3^{\frac{n}{1+\alpha}}$, and the number of choices for $T_2 \subseteq V_2$ is $2^{\frac{\alpha n}{1+\alpha}}$. By setting $\alpha = \log 3$, sizes of matrices X and Y become $3^{\frac{n}{\log 6}} \times 3^{\frac{n}{\log 6}}$, and we can multiply them in $O^*(3^{\frac{\omega}{\log 6}n})$ time. After the multiplication, we can compute the sum over T in $O^*(3^{\frac{2}{\log 6}n})$ time. In total, we can compute dp_{i+1} from dp_i in $O^*(3^{\frac{\omega}{\log 6}n})$ time and $O^*(3^{\frac{2}{\log 6}n})$ space, and thus we can compute $\mathrm{dp}_n(V)$ in the same time (up to a polynomial factor) and the same space. ◀

3.4 Utilizing Sparsity

Since Y_{T_2,S_2} in the previous subsection has non-zero value only when $T_2 \subseteq S_2$, the matrix Y has only $3^{|V_2|}$ non-zero entries. Moreover, by aligning indices of Y properly, we can see that Y is a *recursively partial matrix*, defined as follows.

▶ **Definition 3** (Recursively Partial Matrix)**.** Any matrix X with size 1×1 is a recursively partial matrix. A square matrix X with size 2^k is called recursively partial if when dividing it into four submatrices of size 2^{k-1}, the bottom left submatrix is a zero matrix and the other three submatrices are recursively partial matrices.

To exploit this structure when computing matrix products, we use the following theorem due to Bini et al. [3].

▶ **Theorem 4** ([3])**.** *We can compute an approximate product of a 2×2 matrix of the form $\begin{bmatrix} a & b \\ 0 & c \end{bmatrix}$ and any 2×2 matrix with 5 multiplications.*

Here, approximate product means that for any $\epsilon > 0$, we can compute the product with relative error depending on ϵ and it convergences to the exact value when $\epsilon \to 0$. Schönhage [14] observed that we can compute the exact matrix product by regarding ϵ as an indeterminate. This modification increases the running time by $O(\log n)$ factor. This type of matrix product, which computes the product of matrices containing zeros in a structured way, is called *partial matrix product*. The partial matrix product has been well studied for obtaining a faster algorithm for square matrix product, however, in our case, we can use the algorithm recursively to multiply a matrix X and a recursively partial matrix Y.

We give a detailed explanation of the algorithm. Let A be a $2^k \times 2^k$ matrix, B be a $2^k \times 2^k$ recursively partial matrix, and C be a product of A and B. We say that C' is a *d-approximate product* of C with an indeterminate ϵ if for each index (i, j), $C'_{i,j}$ can be written as $C'_{i,j} = C_{i,j} + \epsilon P_{i,j}(\epsilon)$ for some polynomial P of degree at most $d - 1$. Now, we compute the k-approximate product C' of C. We divide each matrix into four submatrices of size $2^{k-1} \times 2^{k-1}$. Then, the product of these two matrices is:

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1}, & C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2}, \\ C_{2,1} &= A_{2,1}B_{1,1}, & C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}. \end{aligned}$$

We compute the $(k-1)$-approximate products of the followings by applying the algorithm recursively.

$$\begin{aligned} Z_1 &= (A_{2,1} + \epsilon A_{2,2})(B_{2,2} + \epsilon B_{1,2}), \\ Z_2 &= A_{1,1}(B_{1,1} + \epsilon B_{1,2}), \\ Z_3 &= A_{2,1}B_{2,2}, \\ Z_4 &= (A_{1,1} + A_{2,1} + \epsilon A_{1,2})B_{1,1}, \\ Z_5 &= (A_{2,1} + \epsilon A_{1,2})(B_{2,2} + B_{1,1}). \end{aligned}$$

Then, we can obtain k-approximate product C' from these products as follows:

$$\begin{aligned} C'_{1,1} &= Z_2, & C'_{1,2} &= \epsilon^{-1}(Z_2 - Z_3 - Z_4 + Z_5), \\ C'_{2,1} &= -Z_3 + Z_5, & C'_{2,2} &= \epsilon^{-1}(Z_1 - Z_3). \end{aligned}$$

Therefore, we can compute k-approximate product of two matrices in $O^*(5^k)$ time. Because the degree of each polynomial $P_{i,j}$ is at most $k - 1$, we can compute the exact product C by running the algorithm k times and using the interpolation.

By using this partial matrix product algorithm, we can compute the product XY in $O^*(5^{|V_2|}) = O^*(3^{\frac{\log 5}{\log 6}n})$ time. Recall that $|V_2| = \frac{\alpha n}{1+\alpha} = \frac{\log 3}{\log 6}n$ by our choice of $\alpha = \log 3$. As a result, we can solve Max-PA in $O^*(3^{\frac{\log 5}{\log 6}n})$ time and $O^*(3^{\frac{2}{\log 6}n})$ space.

4 Approximation Algorithms for Max-PA

In this section, we give a 0.4586-approximation algorithm for Max-PA. First, we reduce an instance of general Max-PA into an instance of Max-PA that has $(<)$-constraints and $(=)$-constraints only. This can be done by the following reduction.

- For each constraint of the form $(x \leq y)$, we replace it by two constraints $(x < y)$ and $(x = y)$.
- For each constraint of the form $(x \neq y)$, we replace it by two constraints $(x < y)$ and $(y < x)$.

The optimal value does not change through the reduction. This is because, for any assignment f, if the removed constraint is satisfied by f, then exactly one of the added two constraints is satisfied by f, and if the removed constraint is not satisfied by f, then none of the added two constraints is satisfied by f.

After applying the reduction, consider the following SDP:

$$\begin{aligned} \text{maximize} \quad & \sum_{(u=v)\in C} \langle x_u, x_v\rangle + \sum_{(u<v)\in C} (1-\langle x_u, x_v\rangle), \\ \text{subject to} \quad & \|x_v\|^2 = 1 \quad (\forall v \in V), \\ & \langle x_u, x_v\rangle \geq 0 \quad (\forall u, v \in V), \end{aligned}$$

where x_v $(v \in V)$ is a real vector. To see that the SDP above is indeed a relaxation, let $f^* : V \to [n]$ be the optimal solution. Then, we set x_v be the vector whose i-th coordinate is 1 if $f^*(v) = i$ and 0 otherwise. Note that $\langle x_u, x_v\rangle = 1$ iff $f^*(u) = f^*(v)$. In particular, $1 - \langle x_u, x_v\rangle$ has value 1 when the constraint $(u < v)$ is satisfied. Therefore, the optimal SDP value gives the upper bound on the optimal value of the original Max-PA.

We can solve the SDP in polynomial time and let x^* be the optimal SDP solution. We create an assignment f by rounding x^* as follows. First, we generate k random n-dimensional unit vectors $y_1, \ldots, y_k$, where k is a parameter. Then, we partition the variable set V into 2^k groups according to the signs of k inner products $\langle x_v^*, y_i\rangle$. For each group, we assign a unique value to all variables in the group. Thus, we use at most 2^k different values in total. Finally, we introduce a random ordering among the values assigned to groups.

Now, we analyze the approximation ratio of this algorithm. A constraint $(u = v)$ is satisfied if u and v are in the same group. This event happens when they are in the same side of every hyperplane $\langle x, y_i\rangle = 0$. Thus, the probability that $(u = v)$ is satisfied is

$$\Pr[f(u) = f(v)] = \left(1 - \frac{\theta}{\pi}\right)^k,$$

where θ is the angle between two vectors x_u^* and x_v^*. In contrast, the constraint $(u = v)$ contributes to the SDP value by $\langle x_u^*, x_v^*\rangle = \cos\theta$.

A constraint $(u < v)$ is satisfied with probability $\frac{1}{2}$ if they are in different groups. Thus, the probability that the constraint is satisfied is

$$\frac{1}{2}\Pr[f(u) \neq f(v)] = \frac{1}{2}\left(1 - \left(1 - \frac{\theta}{\pi}\right)^k\right).$$

In contrast, the constraint $(u < v)$ contributes to the SDP value by $1 - \langle x_u^*, x_v^*\rangle = 1 - \cos\theta$.

The approximation ratio is a convex function of k and takes the maximum between $k = 3$ and $k = 4$. In order to take the balance, we run the algorithm by choosing $k = 3$ with probability α and $k = 4$ with probability $1-\alpha$. Then, the approximation ratio is at least

$$\min\left\{\min_\theta \frac{\alpha(1-\frac{\theta}{\pi})^3+(1-\alpha)(1-\frac{\theta}{\pi})^4}{\cos\theta}, \min_\theta \frac{\alpha\left(1-(1-\frac{\theta}{\pi})^3\right)+(1-\alpha)\left(1-(1-\frac{\theta}{\pi})^4\right)}{2(1-\cos\theta)}\right\}.$$

By setting $\alpha = \frac{191}{593}$, the above value becomes $\frac{272}{593} > 0.4586$. The worst case is achieved when $\theta = \frac{\pi}{3}$ for $(=)$-constraints and $\theta = \frac{\pi}{2}$ for $(<)$-constraints.

5 Approximation Algorithms for Min-PA

In this section, we give an $O(\log n \log\log n)$-approximation algorithm for Min-PA. First, we reduce an instance of general Min-PA into an instance of Min-PA that has $(\leq)$-constraints and $(\neq)$-constraints only. This can be done by the following reduction.

- For each constraint of the form $(x = y)$, we replace it by two constraints $(x \leq y)$ and $(x \geq y)$.
- For each constraint of the form $(x < y)$, we replace it by two constraints $(x \leq y)$ and $(x \neq y)$.

The optimal value does not change through the reduction. This is because, for any assignment f, if the removed constraint is unsatisfied by f, then exactly one of the added two constraints is unsatisfied by f, and if the removed constraints is not unsatisfied, then none of the added two constraints is unsatisfied by f.

A sequence of variables $(v_1, \ldots, v_k)$ is called a *$(\leq)$-path* from v_1 to v_k if a constraint $(v_i \leq v_{i+1})$ exists for every $i \in [k-1]$. Using the notion of $(\leq)$-path, we can obtain the following necessary and sufficient condition under which an instance is satisfiable.

▶ **Lemma 5** ([17]). *A CSP instance with $(\leq)$-constraints and $(\neq)$-constraints only is satisfiable if and only if, for any constraint $(x \neq y)$, we do not have a $(\leq)$-path from x to y and a $(\leq)$-path from y to x simultaneously.*

Due to Lemma 5, Min-PA with $(\leq)$-constraints and $(\neq)$-constraints only can be formulated as the following integer programming.

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in C} x_e, \\ \text{subject to} \quad & x_e + \sum_{f \in P_{u,v}} x_f + \sum_{f \in P_{v,u}} x_f \geq 1 \quad \begin{pmatrix} \forall e = (u \neq v) \in C \\ \forall P_{u,v} : \text{a } (\leq)\text{-path from } u \text{ to } v \\ \forall P_{v,u} : \text{a } (\leq)\text{-path from } v \text{ to } u \end{pmatrix}, \qquad (3) \\ & x_e \in \{0,1\} \ (\forall e \in C). \end{aligned}$$

Here, $x_e = 1$ means that the constraint e is unsatisfied. Now we relax it to a linear programming by changing the last constraint to $x_e \geq 0$. The LP contains an exponential number of constraints. However, we can solve it in polynomial time by using the ellipsoid method since there is a polynomial-time separation oracle. More specifically, we construct a digraph as follows to check constraints (3). That is, For each constraint of the form $e = (u \leq v)$, we make an edge (u, v) of length x_e. Then, we check the corresponding constraints (3) is satisfied for each constraint of the form $e = (u = v)$. Here, $\sum_{f \in P_{u,v}} x_f$ (resp., $\sum_{f \in P_{u,v}} x_f$) can be computed as the length of the shortest path from u to v (resp., v to u) in the digraph.

Let x^* be the optimal solution to the LP. We create a new LP by using x^*. Let $e = (u \neq v)$ be a constraint, $P_{u,v}$ be a $(\leq)$-path from u to v and $P_{v,u}$ be a $(\leq)$-path from v to u. We first remove the corresponding constraint (3). If $x_e^* \geq \frac{1}{2}$, then we add a constraint $x_e \geq 1$ to the new LP. Otherwise and hence if $\sum_{f \in P_{u,v}} x_f^* + \sum_{f \in P_{v,u}} x_f^* \geq \frac{1}{2}$, then we add a constraint $\sum_{f \in P_{u,v}} x_f + \sum_{f \in P_{v,u}} x_f \geq 1$ to the new LP. The optimal value of the new LP is at most twice the optimal value of the original LP since $2x^*$ is a feasible solution to the new LP, and any feasible solution to the new LP is also a feasible solution to the original LP. Therefore, an integer solution to the new LP whose value is at most k times the optimal value of the new LP is a $2k$-approximation solution to the original problem.

The obtained problem can be considered as a variant of the directed multicut problem: given a digraph $G = (V, E)$ and a set of terminal pairs T, find an edge set $F \subseteq E$ of minimum cardinality such that for any terminal pair $(u, v) \in T$, $G - F$ contains no path from u to v or no path from v to u. We call this problem as the *symmetric multicut problem*. Also, we call a terminal pair (u, v) *separated* if there is no path from u to v or there is no path from v to u. We note that, in the standard multicut problem, a terminal pair (u, v) is also directed and we only care about deleting paths from u to v. We can obtain an instance of the symmetric multicut problem by setting $E = \{(u, v) \mid (u \leq v) \in C\}$, and $T = \{(u, v) \mid e = (u \neq v) \in C, x_e^* < \frac{1}{2}\}$.

To get approximation to the symmetric multicut problem, we consider the following LP relaxation:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{e \in E} x_e, \\ \text{subject to} & \displaystyle\sum_{f \in P_{u,v}} x_f + \sum_{f \in P_{v,u}} x_f \geq 1 \quad \begin{pmatrix} \forall (u, v) \in T \\ \forall P_{u,v} : \text{a path from } u \text{ to } v \\ \forall P_{v,u} : \text{a path from } v \text{ to } u \end{pmatrix}, \\ & x_e \geq 0 \ (\forall e \in E). \end{array} \tag{4}$$

Even et al. [8] showed an $O(\log n \log \log n)$-approximation algorithm for the (standard) multicut problem in some special kind of graphs, called *circular networks*. We use the same approach to solve the symmetric multicut problem. The following theorem immediately gives that the symmetric multicut problem and hence Min-PA can be approximated within a factor of $O(\log n \log \log n)$.

▶ **Theorem 6.** *The solution to the symmetric multicut problem whose value is at most $O(\log n \log \log n)$ times the optimal value of the LP (4) can be obtained in polynomial time.*

Proof. Let l be twice the optimal value of the LP (4). For an edge set $F \subseteq E$, we define $l(F) = \sum_{e \in F} l(e)$. We define the *length* of a path P as $l(P)$, and the *distance from u to v* as the length of the shortest path from u to v. For any terminal pair (s, t), the sum of distances from s to t and from t to s is at least 2.

We fix some terminal pair (s, t) and we assume that the distance from s to t is at least 1. Let $d(v)$ be the distance from s to v. For any $0 \leq x \leq 1$, we define edge sets $A(x)$, $L(x)$, and $B(x)$ as follows:

$$\begin{aligned} A(x) &= \{(u, v) \in E \mid d(u), d(v) \leq x\}, \\ L(x) &= \{(u, v) \in E \mid d(u) \leq x < d(v)\}, \\ B(x) &= \{(u, v) \in E \mid x < d(u), d(v)\}. \end{aligned}$$

Now we claim that there exists $0 \leq x \leq 1$ such that

$$|L(x)| \leq \mu(l(E)) - \mu(l(A(x))) - \mu(l(B(x))), \tag{5}$$

where $\mu(x) = 4x \ln(4x) \ln\log(4x)$. To show the claim, we use the following lemma by Seymour [15].

▶ **Lemma 7** ([15]). *Let $k > 0$ be a real number, let y be a real-valued monotone increasing function on $[0, 1]$ such that $y(0) \geq 0, y(1) \leq 1$ and for all $h \in [0, 1] - I$, where $I \subseteq [0, 1]$ is some finite subset of $[0, 1]$, y is differentiable and $\frac{dy}{dx}\Big|_{x=h} \geq \frac{1}{k}$. Then there exists h with $\frac{1}{4} < h < \frac{3}{4}, h \notin I$, such that*

$$k \left.\frac{dy}{dx}\right|_{x=h} \leq \mu(k) - \mu(ky(h)) - \mu(k(1 - y(h))).$$

This lemma is slightly different from the original one. In the original lemma, y is required to be contiguous, and this implies that y is monotone increasing because $\frac{dy}{dx}\Big|_{x=h} \geq \frac{1}{k}$. Actually, the same proof can be applied if y is not contiguous but monotone increasing.

We instantiate Lemma 7 with the following function y:

$$y(x) = \frac{1}{l(E)} \left(l(A(x)) + \sum_{e=(u,v)\in L(x)} (x - d(u)) \right).$$

Note that the function $y(x)$ is not contiguous at $x = d(v)$ for $v \in V$. An edge $e = (u, v)$ contributes to y by $\frac{1}{l(E)}(x-d(u))$ if $e \in L(x)$, and by $\frac{1}{l(E)}l(e)$ if $e \in A(x)$. Thus its contribution changes from $\frac{1}{l(E)}(d(v) - d(u))$ to $\frac{1}{l(E)}l(e)$ at $x = d(v)$. Because $d(v) - d(u) \leq l(e)$ holds for any edge $e = (u, v)$, its contribution to y never decreases. Therefore, the function y is monotone increasing. For any differentiable point $0 < x < 1$, the value of $\frac{dy}{dx}$ is $\frac{|L(x)|}{l(E)}$, which is at least $\frac{1}{l(E)}$ because $L(x) \neq \emptyset$. Therefore, there exists h such that:

$$\begin{aligned} |L(h)| &= l(E) \left.\frac{dy}{dx}\right|_{x=h} \\ &\leq \mu(l(E)) - \mu(l(E)y(h)) - \mu(l(E)(1 - y(h))) \\ &\leq \mu(l(E)) - \mu(l(A(h))) - \mu(l(B(h))), \end{aligned}$$

and the claim holds. Because there are essentially at most n choices for h, we can find it in polynomial time.

To obtain a good cut, we just find a real number x satisfying the condition (5) and then we remove the edge set $L(x)$. Let V_1 be the set of vertices whose distances from s are at most x and V_2 be the set of vertices whose distances from s are greater than x. Then, $L(x)$ contains only edges from V_1 to V_2 and does not contain any edge from V_2 to V_1. Nonetheless, since all paths from V_1 to V_2 are cut, every terminal pair (u, v) with $u \in V_1$ and $v \in V_2$ become separated. In particular, the terminal pair (s, t) is separated.

Thus, we can consider two graphs $G_1 = (V_1, A(x))$ and $G_2 = (V_2, B(x))$ separately, and we can recursively solve the symmetric multicut problem on G_1 and G_2 independently. We can show that the total number of removed edges in recursive steps is at most $\mu(l(E))$ by induction on $|E|$ because $|L(x)| + \mu(l(A(x))) + \mu(l(B(x))) \leq \mu(l(E))$. On the other hand, since l is twice the optimal LP solution, the optimal LP value is $\frac{1}{2}l(E)$. Thus we can obtain a solution whose value is at most $O(\log n \log\log n)$ times the optimal value of the LP (4). ◀

Acknowledgments

The authors are grateful to François Le Gall for valuable discussions about the usage of sparse matrix product for the exact algorithm.

References

1 J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1):89–113, 2004.

3 D. Bini, M. Capovani, F. Romani, and G. Lotti. $O(n^{2.7799})$ complexity for $n*n$ approximate matrix multiplication. *Inf. Process. Lett.*, 8(5):234–235, 1979.

4 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–74, 2007.

5 M. Bodirsky and J. Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):1–41, 2010.

6 M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.

7 I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, pages 439–485, 2005.

8 G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

9 V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing*, 40(3):878–914, 2011.

10 V. Guruswami, R. Manokaran, and P. Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 573–582, 2008.

11 S. Khot. On the power of unique 2-prover 1-round games. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 767–775, 2002.

12 J. Malik and T. Binford. Reasoning in time and space. In *Proc. 8th International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 343–345, 1983.

13 B. Nebel and H. Bürckert. Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

14 A. Schönhage. Partial and total matrix multiplication. *SIAM J. Comput.*, 10(3):434–455, 1981.

15 P. D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

16 C. Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 526–527, 2004.

17 P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

18 M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. 5th National Conference on Artificial Intelligence*, pages 377–382, 1986.

19 R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

20 V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.