A decision diagram-based solver for the independent set reconfiguration problem

Jun Kawahara[1] and Hiroki Yamazaki[1]

[1]Kyoto University

1 Engine: Core Solver or Algorithm

Our solver is based on the algorithm [2] that uses zero-suppressed binary decision diagrams [4] and the graphillion library [1].

2 ZDD

A ZDD is a data structure for efficiently representing a family of sets. The definition of a ZDD is given below. Let $X = \{x_1, \dots, x_n\}$ be an underlying set of a family represented by a ZDD, and let $x_1 < x_2 < \cdots < x_n$. A ZDD is a directed acyclic graph (DAG) $D = (N, A)$ with the following property. D has at most two nodes with outdegree zero, which are called 0-terminal and 1-terminal. Nodes other than the terminals are called non-terminal nodes. A non-terminal node has one element of X and is called a label. A non-terminal node has two arcs, called a 0-arc and a 1-arc. If a non-terminal node $\nu \in N$ has label $x \in X$ and its 0-arc and 1-arc point at $\nu_0, \nu_1 \in N$, respectively, we write $\nu = (x, \nu_0, \nu_1)$. If $\nu_i = (x^i, \nu_{i0}, \nu_{i1})$ holds, $x < x^i$ must be satisfied $(i = 0, 1)$. D has exactly one node with indegree zero, called the root.

A node ν in a ZDD recursively represents a family of sets as follows. If ν is the 0- and 1-terminal, it represents $\emptyset$ and $\{\emptyset\}$, respectively. If ν is a non-terminal node, we write $\nu = (x, \nu_0, \nu_1)$. The family $\mathcal{S}(\nu)$ of sets is the union of the family of sets represented by ν_0 and the family of sets obtained by adding x to each element of the family of sets represented by ν_1. That is, $\mathcal{S}(\nu) = \mathcal{S}(\nu_0) \cup (\{\{x\}\} \bowtie \mathcal{S}(\nu_1))$, where $\mathcal{F} \bowtie \mathcal{G} = \{F \cup G \mid F \in \mathcal{F}, G \in \mathcal{G}\}$. We interpret the family of sets represented by its ZDD as the family of sets represented by the root node. For more information on ZDDs, please refer to [3].

3 Algorithm for solving the independent set reconfiguration problem

Given a graph $G = (V, E)$, an independent set of G can be represented as a subset of V. The collection of all independent sets of G can be represented by a family of sets with underlying set V. For a family $\mathcal{F}$ of sets with underlying set V and a set $A \in V$, let $\mathsf{swap}(\mathcal{F}, A)$ be the family of all the sets obtained by removing an element from every set in $\mathcal{F}$ and adding an element in A to the set. That is,

$$\mathsf{swap}(\mathcal{F}, A) = \{F \cup \{x\} \setminus \{x'\} \mid F \in \mathcal{F}, x \notin F, x \in A, x' \in F\}.$$

Our algorithm for computing $\mathsf{swap}(\mathcal{F}, A)$ is described in the next section. Using swap, the token-jumping model of the independent set reconfiguration problem can be solved as follows. In what follows, the underlying set of all families is V. Let $\mathcal{F}_{\mathrm{ind}}$ be the family of all the independent sets of G. Let $\mathcal{F}_0 = \{S\}$. We recursively define $\mathcal{F}_{i+1} = \mathsf{swap}(\mathcal{F}_i, V) \cap \mathcal{F}_{\mathrm{ind}}$. $\mathsf{swap}(\mathcal{F}_i, V)$ is the family of sets obtained by adding and removing one element of each set of $\mathcal{F}_i$. By taking the intersection of $\mathsf{swap}(\mathcal{F}_i, V)$ and $\mathcal{F}_{\mathrm{ind}}$, we extract independent sets from $\mathsf{swap}(\mathcal{F}_i, V)$. $\mathcal{F}_i$ is the family of independent sets obtained by reconfiguring S using at most i steps (satisfying that the intermediate sets are also independent sets). The above computation is performed for $i = 0, 1, \ldots$ in this order. If $T \in \mathcal{F}_i$ holds for some i, we know that there exists a reconfiguration sequence from S to T with length i, and the algorithm stops. If $\mathcal{F}_i = \mathcal{F}_{i+1}$ for some i, no new independent set is obtained, and we know that there is no reconfiguration sequence from S to T. From the construction of $\mathcal{F}_i$, the smallest i such that $T \in \mathcal{F}_i$ holds is the minimum number of steps among all the reconfiguration sequences from S to T.

4 Reconfiguration operation for families of sets using ZDDs

In this section, we describe how to realize the algorithm described in the previous section using ZDDs. A method to construct a ZDD representing $\mathcal{F}_{\mathrm{ind}}$ is described in the book [3]. The construction of a ZDD for $\mathcal{F}_0 = \{S\}$ is obvious. For any set A, there exists a method to determine whether A belongs to $\mathcal{F}$. Given two families $\mathcal{F}, \mathcal{F}'$ of sets as ZDDs, a method for constructing a ZDD representing $\mathcal{F} \cap \mathcal{F}'$ has been proposed [4]. Therefore, if there is a method for constructing a ZDD representing $\mathsf{swap}(\mathcal{F}, A)$, all the above procedures can be performed as ZDD operations.

Given a family $\mathcal{F}$ of sets as a ZDD, we describe how to construct a ZDD representing $\mathsf{swap}(\mathcal{F}, A)$. The swap operation uses the following operations:

$$\mathsf{rem}(\mathcal{F}) = \{F \setminus \{x\} \mid F \in \mathcal{F}, x \in F\},$$
$$\mathsf{add}(\mathcal{F}, A) = \{F \cup \{x\} \mid F \in \mathcal{F}, x \notin F, x \in A\}.$$

For every $x \in X$, we can write $\mathcal{F} = \mathcal{F}_0 \cup (\{\{x\}\} \bowtie \mathcal{F}_1)$. We let $\mathcal{F}^{\mathrm{rem}} = \mathsf{rem}(\mathcal{F})$ and $\mathcal{F}^{\mathrm{rem}} = \mathcal{F}_0^{\mathrm{rem}} \cup (\{\{x\}\} \bowtie \mathcal{F}_1^{\mathrm{rem}})$, where any set in $\mathcal{F}_0^{\mathrm{rem}}$ and $\mathcal{F}_1^{\mathrm{rem}}$ does not contain x.

Each element of $\{\{x\}\} \bowtie \mathcal{F}_1^{\mathrm{rem}}$ is obtained by removing one element other than x from each set in $\{\{x\}\} \bowtie \mathcal{F}_1$. Thus, $\mathcal{F}_1^{\mathrm{rem}} = \mathsf{rem}(\mathcal{F}_1)$ holds. Also, each element of $\mathcal{F}_0^{\mathrm{rem}}$ is obtained by removing one element from each set in $\mathcal{F}_0$, or by removing x from each set in $\{\{x\}\} \bowtie \mathcal{F}_1$. Thus, $\mathcal{F}_0^{\mathrm{rem}} = \mathsf{rem}(\mathcal{F}_0) \cup \mathcal{F}_1$ holds.

We describe our algorithm $\mathsf{rem}(\nu)$ that constructs a ZDD representing the family $\mathsf{rem}(\mathcal{S}(\nu))$ of sets for a ZDD $\nu = (x, \nu_0, \nu_1)$. If ν is the 0- or 1-terminal, the 0-terminal is returned. Otherwise, we perform the following. First, recursively compute $\mathsf{rem}(\nu_0)$ and take the union of it and ν_1 (as a ZDD operation) [3]. Next, recursively compute $\mathsf{rem}(\nu_1)$. Finally, return $(x, \mathsf{rem}(\nu_0) \cup \nu_1, \mathsf{rem}(\nu_1))$ as the root node of $\mathsf{rem}(\nu)$.

For a ZDD $\nu = (x, \nu_0, \nu_1)$, the algorithm $\mathsf{add}(\nu, A)$, which constructs a ZDD representing $\mathsf{add}(\mathcal{S}(\nu), A)$, can be written as well as rem in a recursive mannar. If ν is the 0-terminal, it returns the 0-terminal. If ν is the 1-terminal, it returns the ZDD representing $\{\{x\} \mid x \in A\}$ (the construction of the ZDD is clear). If $x < \min(A)$, return $(x, \mathsf{add}(\nu_0, A), \mathsf{add}(\nu_1, A))$ because $x < \min(A)$ means $x \notin A$, where $\min(A)$ is the element y in A such that for all $y' (\neq y) \in A$, $y < y'$ holds. If $x = \min(A)$, return (x, ν_0', ν_1'), where $\nu_0' = \mathsf{add}(\nu_0, A \setminus \{x\})$, and $\nu_1' = \mathsf{add}(\nu_1, A \setminus \{x\}) \cup \nu_0$. If $x > \min(A)$, return $(\min(A), \mathsf{add}(\nu, A \setminus \{x\}), \nu)$.

For a ZDD $\nu = (x, \nu_0, \nu_1)$, the algorithm $\mathsf{swap}(\nu, A)$, which constructs a ZDD representing $\mathsf{swap}(\mathcal{S}(\nu), A)$, can be written as well as rem and add in a recursive mannar. If ν is the 0- or 1-terminal, it returns the 0-terminal. If $x < \min(A)$, return $(x, \mathsf{swap}(\nu_0, A), \mathsf{swap}(\nu_1, A))$. If $x = \min(A)$, return (x, ν_0', ν_1'), where $\nu_0' = \mathsf{swap}(\nu_0, A \setminus \{x\}) \cup \mathsf{add}(\nu_1, A \setminus \{x\})$, and $\nu_1' = \mathsf{swap}(\nu_1, A \setminus \{x\}) \cup \mathsf{rem}(\nu_0)$. If $x > \min(A)$, return $(\min(A), \mathsf{swap}(\nu, A \setminus \{x\}), \mathsf{rem}(\nu))$.

References

[1] Inoue, T., Iwashita, H., Kawahara, J., Minato, S., Graphillion: software library for very large sets of labeled graphs, International Journal on Software Tools for Technology Transfer 18 (1) 57–66, 2016. `https://doi.org/10.1007/s10009-014-0352-z`

[2] Ito, T., Kawahara, J., Nakahata, Y., Soh, T., Suzuki, A., Teruyama, J., Toda, T., ZDD-based algorithmic framework for solving shortest reconfiguration problems, arXiv:2207.13959v1, 2022.

[3] Knuth, D. E., The Art of Computer Programming, Volume 4A, Combinatorial Algorithms, Part 1, 1st Edition, Addison-Wesley Professional, 2011.

[4] Minato, S., Zero-suppressed BDDs for set manipulation in combinatorial problems. In *the 30th ACM/IEEE design automation conference*, pages 272–277, 1993. `https://doi.org/10.1145/157485.164890`