Object Reachability via Swaps under Strict and Weak Preferences

Sen Huang · Mingyu Xiao

Received: date / Accepted: date

Abstract The Housing Market problem is a widely studied resource allocation problem. In this problem, each agent can only receive a single object and has preferences over all objects. Starting from an initial endowment, we want to reach a certain assignment via a sequence of rational trades. We first consider whether an object is reachable for a given agent under a social network, where a trade between two agents is allowed if they are neighbors in the network and no participant has a deficit from the trade. Assume that the preferences of the agents are strict (no tie among objects is allowed). This problem is polynomially solvable in a star-network and NP-complete in a tree-network. It is left as a challenging open problem whether the problem is polynomially solvable when the network is a path. We answer this open problem positively by giving a polynomial-time algorithm. Then we show that when the preferences of the agents are weak (ties among objects are allowed), the problem becomes NP-hard even when the network is a path. In addition, we consider the computational complexity of finding different optimal assignments for the problem under the network being a path or a star.

Keywords Resource Allocation · Social Choice Theory · Pareto Efficiency · Computational Complexity · Coordination and Cooperation

1 Introduction

Allocating indivisible objects to agents is an important problem in both computer science and economics. A widely studied setting is that each agent can only receive one single object and each agent has preferences over objects. This problem was previously called Assignment problem [1, 2, 3] and now we prefer to call it House Allocation problem [4, 5]. When each agent is initially endowed with an
object and we want to reallocate objects under some conditions without any monetary transfers, the problem is known as Housing Market problem [6]. Housing Market has several real-life applications such as allocation of housings [7], organ exchange [8] and so on. There are two different preference sets for agents. One is strict, which is a full ordinal list of all objects, and the other one is weak, where agents are allowed to be indifferent between objects. Both preference sets have been widely studied. Under strict preferences, the celebrated Top Trading Cycle rule [6] has several key desirable properties. Some modifications of Top Trading Cycle rule, called Top Trading Absorbing Sets rule and Top Cycles rule, were introduced for weak preferences [9, 10], which also hold some good properties. More studies of Housing Market under the two preference sets from different aspects can be found in the literature [10, 11, 12, 13, 14, 15].

Some rules allow a single exchange involving many agents. It is natural and fundamental to consider exchanges being bilateral deals (swaps), i.e., each exchange of objects happens only between two agents. A swap between two agents is allowed when they are mutually beneficial from the exchange. This natural rule for Housing Market has been studied in the literature [16, 17].

In some models, it is implicitly assumed that all agents have a tie with others. However, some agents often do not know each other and do not have the capacity to exchange even they can mutually get benefits. So Gourv et al. [17] studied Housing Market where the agents are embedded in a social network to denote the ability to exchange objects between them. In fact, recently it is a hot topic to study resources allocation problems over social networks and analyze the influences of networks. Abebe et al. [18] and Bei et al. [19] introduced social networks of agents into the Fair Division problem of cake cutting. They defined (local) fairness concepts based on social networks and then compared them to the classic fairness notions and designed new protocols to find envy-free allocations in cake cutting. Bredereck et al. [20] and Beynier et al. [21] also considered network-based Fair Division in allocating indivisible resources.

In this paper, we study Housing Market in a social network with simple trades between pairs of neighbors in the network. In this model, there are the same number of agents and objects and each agent is initially endowed with a single object. Each agent has a preference over all objects. The agents are embedded in a social network which determines their ability to exchange their objects. Two agents may swap their items under two conditions: they are neighbors in the social network, and they find it mutually profitable (or no one will become worse under weak preferences). Under this model, there are many problem can be studied, i.e, Object Reachability (OR), Assignment Reachability (AR) and Pareto Efficiency (PE), see [17]. Object Reachability is to determine whether an object is reachable for a given agent from the initial endowment via swaps. Assignment Reachability is to determine whether a certain assignment of objects to all agents is reachable. Pareto Efficiency is to find a Pareto optimal assignment within all the reachable assignments.

In this paper, we mainly consider Object Reachability and Pareto Efficiency under different preferences. Note that Assignment Reachability has been well studied and related results can be found in [17]. For Object Reachability, Damamme et al. [16] firstly proved that the problem is NP-hard even to decide whether any one of a subset of objects is reachable for each agent. Gourv et al. [17] further showed that Object Reachability is polynomially solvable un-
under star-structures and NP-complete under tree-structures. For the network being a path, they solved the special case where the given agent is an endpoint (a leaf) in the path and left it unsolved for the general case. All the above results are under strict preferences. In this paper, we will answer the open problem positively by giving a polynomial-time algorithm for Object Reachability in a path under strict preferences, and also prove that Object Reachability in a path under weak preferences is NP-hard and can be solvable in polynomial time in a star.

For Pareto Efficiency, Gourv et al. [17] proved that the problem under strict preferences is NP-hard in general graphs and polynomially solvable in a star or path. In this paper, we further consider Pareto Efficiency under weak preferences and show that it is NP-hard in a path. Our results and previous results are summarized in Table 1.

| Preferences | Problems | The network |
|-------------|----------|-------------|
| Strict      | OR       | P [17]      | P [17]      | NP-hard [17] | NP-hard [17] |
|             | PE       | P [17]      | P [17]      | ?            | NP-hard [17] |
| Weak        | OR       | NP-hard [17] | NP-hard [17] | NP-hard [17] | (Theorem 2)   |
|             | PE       | NP-hard [17] | NP-hard [17] | NP-hard [17] | (Theorem 3)   |

In Table 1, our results are marked as bold and the problems left unsolved are denoted by '?'. One of the most important results in this paper is the polynomial-time algorithm for Object Reachability under strict preferences in a path. Although paths are rather simple graph structures, Object Reachability in a path is not easy at all, as mentioned in [17] that “Despite its apparent simplicity, Reachable Object (Object Reachability) in a path is a challenging open problem when no restriction on the agent’s location is made. We believe that this case is at the frontier of tractability.” Our polynomial-time algorithm to solve it involves several techniques and needs to call solvers for the 2-Sat problem.

The following part of the paper is organized as follows. Section 2 provides some backgrounds. Section 3 tackles the reachability of an object for an agent in a path under strict preferences. Sections 4 studies Object Reachability and Pareto Efficiency (Object Reachability) in a path is a challenging open problem when no restriction on the agent’s location is made. We believe that this case is at the frontier of tractability.” Our polynomial-time algorithm to solve it involves several techniques and needs to call solvers for the 2-SAT problem.

The following part of the paper is organized as follows. Section 2 provides some backgrounds. Section 3 tackles the reachability of an object for an agent in a path under strict preferences. Sections 4 studies Object Reachability and Pareto Efficiency (Object Reachability) in a path, and Section 5 studies Object Reachability and Pareto Efficiency under weak preferences in a star. Section 6 makes some concluding remarks. Partial results of this paper were presented on the thirty-third AAAI conference on artificial intelligence (AAAI 2019) and appeared as [22].

2 Background

2.1 Models

There are a set $N = \{1, \ldots, n\}$ of $n$ agents and a set $O = \{o_1, \ldots, o_n\}$ of $n$ objects. An assignment $\sigma$ is a mapping from $N$ to $O$, where $\sigma(i)$ is the object held by
agent $i$ in $\sigma$. We also use $\sigma^t(o_i)$ to denote the agent who holds object $o_i$ in $\sigma$. Each agent holds exactly one object at all time. Initially, the agents are endowed with objects, and the initial endowment is denoted by $\sigma_0$. We assume without loss of generality that $\sigma_0(i) = o_i$ for every agent $i$.

Each agent has a preference regarding all objects. A strict preference is expressed as a full linear order of all objects. Agent $i$’s preference is denoted by $\succ_i$, and $o_a \succ_i o_b$ indicates the fact that agent $i$ prefers object $o_a$ than object $o_b$. The whole strict preference profile for all agents is represented by $\succ$. For weak preferences, agents are allowed to be indifferent between objects. For two disjoint subsets of objects $S_1$ and $S_2$, we use $S_1 \succ_i S_2$ to indicate that all objects in $S_1$ (resp., $S_2$) are indifferent for agent $i$ and agent $i$ prefers any object in $S_1$ than any object in $S_2$. We use $o_a \succeq_i o_b$ to denote that agent $i$ likes $o_a$ at least as much as the same as $o_b$. Two relations $o_a \succeq_i o_b$ and $o_b \succeq_i o_a$ together imply that $o_a$ and $o_b$ are indifferent for agent $i$. We may use $\succeq$ to denote the whole weak preference profile for all agents.

Let $G = (N,E)$ be an undirected graph as the social network among agents, where the edges capture the capability of communication and exchange between two agents. An instance of Housing Market is a tuple $(N,O,\succ,G,\sigma_0)$ or $(N,O,\succeq,G,\sigma_0)$ according to the preferences being strict or weak.

2.2 Dynamics

The approach we take in this paper is dynamic, and we focus on individually-rational trades between two agents. A trade is individual rational if each participant receives an object at least as good as the one currently held, i.e., for two agents $i$ and $j$ and an assignment $\sigma$, the trade between agents $i$ and $j$ on $\sigma$ is individual rational if $\sigma(j) \succeq_i \sigma(i)$ and $\sigma(i) \succeq_j \sigma(j)$.

We require that every trade is performed between neighbors in the social network $G$. Individual rational trades defined according to $G$ are called swaps. A swap is an exchange, where two participants have the capability to communicate.

A sequence of swaps can be represented as a sequence of assignments $(\sigma_0,\sigma_1,\sigma_2,\ldots,\sigma_t)$ such that for any $i \in \{1,\ldots,t\}$, $\sigma_i$ results from a swap from $\sigma_{i-1}$. An assignment $\sigma'$ is reachable if there exists a sequence of swaps $(\sigma_0,\sigma_1,\sigma_2,\ldots,\sigma_t)$ such that $\sigma_t = \sigma'$. An object $o \in O$ is reachable for an agent $i \in N$ if there is a sequence of swaps $(\sigma_0,\sigma_1,\sigma_2,\ldots,\sigma_t)$ such that $\sigma_t(i) = o$. An assignment $\sigma$ Pareto-dominate an assignment $\sigma'$ if for all $i \in N$, $\sigma(i) \succeq_i \sigma'(i)$ and there is an agent $j \in N$ such that $\sigma(j) \succ_j \sigma'(j)$. An assignment $\sigma$ is Pareto optimal if $\sigma$ is not Pareto-dominated by any other assignments.

2.3 Problems

We mainly consider two problems under our model. The first one is Object Reachability, which is to check whether an object is reachable for an agent from the initial endowment via swaps. Another one is Pareto Efficiency, which is to find a Pareto optimal assignment within all the reachable assignments. Object Reachability is formally defined below.
Object Reachability (OR)

**Instance:** \((N, O, \succ, G, \sigma_0)\), an agent \(k \in N\), and an object \(o_l \in O\).

**Question:** Whether is object \(o_l\) reachable for agent \(k\)?

When the preferences are strict, we call the problem **Strict Object Reachability**. When the preferences are weak, we call the problem **Weak Object Reachability in a path**. For Object Reachability in a path, an instance will be denoted by \(I = (N, O, \succ, P, \sigma_0, k \in N, o_l \in O)\), where we assume without loss of generality that \(l < k\). For path structures, we always assume without loss of generality that the agents are listed as 1, 2, \ldots, \(n\) on a line from left to right with an edge between any two consecutive agents. Below is an example for Object Reachability in a path.

**Example 1** There are four agents. The path structure, preference profile and a sequence of swaps are given in Figure 1.

![Fig. 1 Example 1](image)

In this figure, the initial endowments for agents are denoted by squares within the preferences. After a swap between agents 1 and 2 from \(\sigma_0\) we get \(\sigma_1\) and after a swap between agents 2 and 3 from \(\sigma_1\) we get \(\sigma_2\). The object \(o_1\) is reachable for agent 3.

**Pareto Efficiency** is formally defined below. When the preferences are strict, we call the problem **Strict Pareto Efficiency**. When the preferences are weak, we call the problem **Weak Pareto Efficiency**.

**Pareto Efficiency (PE)**

**Instance:** \((N, O, \succ, G, \sigma_0)\).

**Question:** To find a Pareto optimal assignment within all the reachable assignments.

### 3 Strict Object Reachability in a Path

**Strict Object Reachability** is known to be NP-complete when the network is a tree and polynomially solvable when the network is a star [17]. It is left unsolved whether the problem with the network being a path is NP-hard or not. We reveal some properties of **Strict Object Reachability** under the path structure and design a polynomial time algorithm for it. In the remaining part of this section, we assume that the preferences are strict and the network is a path.
Recall that the problem is to check whether an object \( o_l \) is reachable for an agent \( k \) with \( l < k \). The main idea of our algorithm is as follows. First, we show that the instance is equivalent to the instance after deleting all agents (and the corresponding endowed objects) on the left of agent \( l \). Thus, we can assume the problem is to check whether object \( o_l \) is reachable for an agent \( k \). Second, we prove that if \( o_l \) is reachable for agent \( k \) then there is an object \( o_{n'} \) with \( n' \geq k \) that should be moved to agent \( k - 1 \) in the final assignment and we can ignore all agents and objects on the right of agent \( n' \). We guess \( n' \) by letting it be each possible value between \( k \) and \( n \) and get at most \( n \) candidate instances. These instances are called \( \text{neat} \) \( (o_1, o_{n'}, k) \)-Constrained instances. A \( \text{neat} \) \( (o_1, o_{n'}, k) \)-Constrained instance contains only \( n' \) agents and it is to check whether there is a reachable assignment \( \sigma' \), called compatible assignment, such that \( \sigma'(k) = o_1 \) and \( \sigma'(k-1) = o_{n'} \). Third, we are going to find compatible assignments. In a neat \( (o_1, o_{n'}, k) \)-Constrained instance, in every compatible assignment each object \( o_i \) will be moved to either the left or the right of its original position in the path. We prove that for each direction, there is at most one possible position \( i_l \) (or \( i_r \) for the right direction) for each object \( o_i \). We can compute \( i_l \) and \( i_r \) directly in polynomial time. Since there are still two possible final positions for each object, we do not get a feasible assignment yet. Fourth, we reduce the subproblem to 2-Sat and determine which of \( \sigma'_i \) and \( \sigma'_o \) should be chose for each agent \( i \) by solving a 2-Sat instance. We show that each feasible assignment obtained in this step is corresponding to a reachable assignment for the neat \( (o_1, o_{n'}, k) \)-Constrained instance. Finally, we can solve the original problem in polynomial time, because the original instance is a yes-instance if and only if at least one of the candidate neat \( (o_1, o_{n'}, k) \)-Constrained instances is a yes-instance.

In fact, when the preferences are strict, we have the following observations and lemmas.

**Observation 1** Given a sequence of swaps \( (\sigma_0, \sigma_1, \ldots, \sigma_t) \) and an agent \( j \in N \).
For any \( i \in \{0, 1, \ldots, t - 1\} \), it holds either \( \sigma_{i+1}(j) = \sigma_i(j) \) or \( \sigma_{i+1}(j) \succ_j \sigma_i(j) \).

It implies the following lemma.

**Lemma 1** Given a sequence of swaps \( (\sigma_0, \sigma_1, \ldots, \sigma_t) \). For any two integers \( i < j \) in \( \{0, 1, \ldots, t\} \) and any agent \( q \in N \), if \( \sigma_i(q) = \sigma_j(q) \), then \( \sigma_d(q) = \sigma_i(q) \) for any integer \( i \leq d \leq j \).

Lemma 1 also says that an object will not ‘visit’ an agent twice. This property is widely used in similar allocation problems under strict preferences.

Next, we analyze properties under the constraint that the social network is a path. In a swap, the moving of an object is on the right direction if it is moved from agent \( i \) to agent \( i + 1 \), and the moving of an object is on the left direction if it is moved from agent \( i \) to agent \( i - 1 \). In each swap, one object is moved on the right direction and one object is moved on the left direction. We study the tracks of the objects in a feasible assignment sequence.

**Lemma 2** For a sequence of swaps \( (\sigma_0, \sigma_1, \ldots, \sigma_t) \), if \( \sigma_i^L(o_i) = j \) for an object \( o_i \), then there are exactly \( |j - i| \) swaps includes \( o_i \). Furthermore, all the \( |j - i| \) movements of \( o_i \) are on the right direction if \( i < j \), and all the \( |j - i| \) movements of \( o_i \) are on the left direction if \( i > j \).
Lemma 3. Let \((\sigma_0, \sigma_1, \ldots, \sigma_l)\) be a sequence of swaps, and \(o_a\) and \(o_b\) be two objects with \(a < b\). Let \(a' = \sigma^I_l(o_a)\) and \(b' = \sigma^I_l(o_b)\). If \(a' \leq a\) or \(b' \geq b\), then \(a' < b'\).

Proof. We consider three cases. If \(a' \leq a\) and \(b' \geq b\), then by \(a < b\) we get that \(a' \leq a < b \leq b'\). Next, we assume that \(a' \leq a\) and \(b' < b\). By Lemma 2, we know that both of the two objects \(o_a\) and \(o_b\) can only move on the left direction. If \(a' \geq b'\), by \(a < b\) we know that there must exist a swap including both \(o_a\) and \(o_b\). In this swap, two objects are moving on two opposite directions and one object is moving on the right direction, a contradiction to the fact that the two objects can only move on the left direction. The last case that \(a' > a\) and \(b' \geq b\) can be proved in a similar way. □

Lemma 3 shows that any object can only move on one direction. Lemma 3 shows that when an object moves on the right direction, all objects initially allocated on the left of it can not move to the right of it at any time; when an object moves on the left direction, all objects initially allocated on the right of it can not move to the left of it at any time.

In fact, if we want to move an object \(o_l\) to an agent \(k\) with \(k > l\), we may not need to move any object on the left of \(o_l\), i.e., \(o_{l'}\) with \(l' < l\). Equipped with Lemma 3, we can prove

Lemma 4. If object \(o_l\) is reachable for agent \(k\), then there is a feasible assignment sequence \((\sigma_0, \sigma_1, \ldots, \sigma_l)\) such that \(\sigma^I_l(o_l) = k\), and \(\sigma_i(i) = \sigma_0(i)\) for all \(i < l\) if \(l \leq k\) and for all \(i > l\) if \(l \geq k\).

Proof. The result can be derived from Proposition 2 in [17]. The authors give an algorithm to produce a sequence of swaps, which does not move any objects \(o_i\) for all \(i < l\) if \(l \leq k\) and for all \(i > l\) if \(l \geq k\). Proposition 2 says if object \(o_l\) is reachable for agent \(k\) then the sequence of swaps is feasible. □

For an instance \(I = (N, O, >, P, \sigma_0, k, o_l)\) of Strict Object Reachability in a path with \(l < k\), let \(I' = (N', O', >', P', \sigma'_0, k, o_l)\) denote the instance obtained from \(I\) by deleting agents \(\{1, 2, \ldots, l-1\}\) and objects \(\{o_1, o_2, \ldots, o_{l-1}\}\). In other words, we let \(N' = \{l, l+1, \ldots, n\}\), \(O' = \{o_l, o_{l+1}, \ldots, o_n\}\), and \(>\), \(P'\) and \(\sigma'_0\) be the corresponding subsets of \(>\), \(P\) and \(\sigma_0\).

Lemma 5. Object \(o_l\) is reachable for agent \(k\) in the instance \(I\) if and only if object \(o_l\) is reachable for agent \(k\) in the instance \(I'\).
By Lemma 5 we can always assume that the instance of Strict Object Reachability in a path is to check whether the object $o_1$ is reachable for an agent $k$.

Assume that object $o_1$ is reachable for agent $k$. For a sequence of swaps $(\sigma_0, \sigma_1, \ldots, \sigma_t)$ such that $\sigma_t(o_1) = k$, there are exactly $k - 1$ swaps including $o_1$ which are moving $o_1$ on the right direction according to Lemma 2. The last swap including $o_1$ will be happened between agent $k - 1$ and agent $k$. Let $o_{o'}$ denote the other object included in the last swap. In other words, after the last swap, agent $k - 1$ will get the object $o_{o'}$ and agent $k$ will get the object $o_1$. Note that the moving of $o_{o'}$ in this swap is in the left direction. By Lemma 2 we know that all movings of $o_{o'}$ in the sequence of swaps are in the left direction. Therefore, we have the following observation.

**Observation 2** It holds that $n' \geq k$.

Our idea is to transform our problem to the following constrained problem: to determine whether there is a reachable assignment $\sigma$ such that $\sigma(k - 1) = o_{o'}$ and $\sigma(k) = o_1$, where $n' \geq k$. We do not know the exact value of $n'$. So we search by letting $n'$ be each value in $\{k, k + 1, \ldots, n\}$. This will only increase the running time bound by a factor of $n$. We denote the above constrained problem by $(o_1, o_{o'}, k)$-Constrained problem.

**Lemma 6** An instance $I = (N, O, \succ, P, \sigma_0, k, o_1)$ is yes if and only if at least one of the $(o_1, o_{o'}, k)$-Constrained instances for $n' \in \{k, k + 1, \ldots, n\}$ is yes.

For an $(o_1, o_{o'}, k)$-Constrained instance $I$, we use $I_{n'}$ to denote the instance obtained from $I$ by deleting agents $\{n' + 1, n' + 2, \ldots, n\}$ and objects $\{o_{n' + 1}, o_{n' + 2}, \ldots, o_n\}$.

**Lemma 7** An $(o_1, o_{o'}, k)$-Constrained instance $I$ is yes if and only if the instance $I_{n'}$ is yes.

**Proof** We prove this lemma by induction on $n'$. It is clear that the lemma holds when $n' = 2$. Assume that the lemma holds for $n' = n_0 - 1$. We show that the lemma also holds for $n' = n_0$. Let $n' = n_0$. When $I_{n'}$ is yes, it is obvious that $I$ is also yes, since we can use the same sequence of swaps as the solution to them. Next we consider the converse direction and assume that $I$ is yes. Then there is a sequence of swaps $\{\sigma_0, \sigma_1, \ldots, \sigma_t\}$ such that $\sigma_t(k - 1) = o_{o_0}$ and $\sigma_t(k) = o_1$. We consider the first swap $(\sigma_{r-1}, \sigma_r)$ including object $o_{o_0}$. Since $o_{o_0}$ will be moved to agent $k - 1$ with $k - 1 < n_0$, by Lemma 2 we know that all swaps including $o_{o_0}$ will move $o_{o_0}$ on the left direction. So in the swap $(\sigma_{r-1}, \sigma_r)$, the object $o_{o_0}$ is also moving on the left direction. By Lemma 3, we know that no object $o_i$ with $i > n_0$ is moved to the left of $o_{o_0}$ before $\sigma_r$. Let $\sigma'_r$ be the assignments of the first $n_0$ agents in $\sigma_r$. So we do not need to move any objects $o_i$ with $i > n_0$ to get $\sigma'_r$. Now object $o_{o_0}$ is at the position $n_0 - 1$. Since the lemma holds for $n' = n_0 - 1$, we know that there is a solution (a sequence of swaps) that does not move any objects $o_i$ with $i > n_0$, which is also a solution to $I_{n'}$.

By Lemma 4 we can ignore all agents on the right of $n'$ in an $(o_1, o_{o'}, k)$-Constrained instance. An $(o_1, o_{o'}, k)$-Constrained instance is called neat if $n'$
is the last agent. We may simply consider neat \((σ_1, o_{n'}, k)\)-Constrained instances only. For any two integers \(a\) and \(b\), we use \([a, b]\) to denote the set of integers between \(a\) and \(b\) (including \(a\) and \(b\)).

**Lemma 8** Let \((σ_0, σ_1, \ldots, σ_t)\) be a sequence of swaps, and \(o_a\) and \(o_b\) be two objects with \(a < b\). Let \(a' = σ_t^1(o_a)\), \(b' = σ_t^1(o_b)\) and \(Q = [a, a'] \cap [b, b']\). Assume that \(Q \neq \emptyset\).

(a) If \(a' > a\) and \(b' > b\), it holds that \(o_a \succ_q o_b\) for all \(q \in Q\).

(b) If \(a' < a\) and \(b' < b\), it holds that \(o_b \succ_q o_a\) for all \(q \in Q\).

**Proof** Both objects \(o_a\) and \(o_b\) will visit each agent in \(Q\) during the sequence of swaps \((σ_0, σ_1, \ldots, σ_t)\).

(a) Since \(b' > b\), by Lemma 3, we know that for each agent \(q \in Q\) object \(o_b\) will visit agent \(q\) before object \(o_a\). By Observation 1, we know that \(o_a \succ_q o_b\).

(b) Since \(a' < a\), by Lemma 3, we know that for each agent \(q \in Q\) object \(o_a\) will visit agent \(q\) before object \(o_b\). By Observation 1, we know that \(o_b \succ_q o_a\). \(\square\)

See Figure 2 for an illustration for Lemma 8(a).

![Fig. 2 An illustration for Lemma 8(a)](image)

**Lemma 9** Let \((σ_0, σ_1, \ldots, σ_t)\) be a sequence of swaps for a neat \((σ_1, o_{n'}, k)\)-Constrained instance such that \(σ_t^1(o_1) = k\) and \(σ_t^1(o_{n'}) = k - 1\), and \(o_a\) and \(o_b\) be two objects with \(a < b\). Let \(a' = σ_t^1(o_a)\) and \(b' = σ_t^1(o_b)\). Assume that \(a' > a\), \(b' < b\) and \(Q = [a, a'] \cap [b, b'] \neq \emptyset\). Let \(Q' = [a + 1, a'] \cap [b, b']\).

(a) There is a swap including \(o_a\) and \(o_b\) which happens between agent \(c - 1\) and \(c\), where \(c = a' + b' - k + 1 \in Q'\).

(b) It holds that \(o_b \succ_q o_a\) for all \(\max(a, b') \leq q < c\), and \(o_a \succ_q o_b\) for all \(c \leq q \leq \min(a', b)\).

**Proof** Since \(Q \neq \emptyset\), we know that there exists a swap including \(o_a\) and \(o_b\) in the sequence of swaps. Next we determine the position of this swap.

By Lemma 2, we know that during the sequence of swaps each object will be moving on at most one direction. Let \(o_{r_i} = σ_i(i)\) for all \(1 \leq i \leq n'\). Since \(σ_t^1(o_1) = k\), by Lemma 3 we know that \(i < r_j\) for all \(1 \leq i < k\). Thus objects \(o_{r_i}\) for all \(1 \leq i < k\) are moving on the left direction. Since \(σ_t^1(o_{n'}) = k - 1\), by Lemma 3 we know that \(i > r_j\) for all \(k \leq i \leq n'\). Thus objects \(o_{r_i}\) for all \(k \leq i \leq n'\) are moving on the right direction.

Since \(σ_t^1(o_1) = k\) and \(o_{a'}\) moves on the right direction, by Lemma 8 again, we know that \(1 \leq σ_i(i) \leq a\) for each \(k \leq i \leq a'\). For each object in \(\{o_{a'+1}, \ldots, o_{n'}\}\), if it is moving on the right direction then it can only be assigned to agents in \(\{k, k + 1, \ldots, a'\}\) in \(σ_t\) by Lemma 8. So there are
exactly \( a' - k + 1 \) objects in \( \{o_{a+1}, \ldots, o_b\} \) are moving on the right direction, all of which are moved to right of \( k - 1 \) in \( \sigma_t \).

Since object \( o_a \) moves on the left direction from \( b \) to \( b' \), we know that there are exactly \( b - b' \) objects in \( \{o_1, \ldots, o_b\} \) that will be moved to the right of \( o_b \) in \( \sigma_t \), which are moving on the right direction. We further show that there is no object \( o_x \) in \( \{o_1, \ldots, o_b\} \) that are moving on the right direction, but it is moved to the right of \( b' \) in \( \sigma_t \). Since \( \sigma_t^Q(o_{a'}) = k - 1 \) and \( o_b \) moves on the left direction, by Lemma 3 we know that \( b' \leq k - 1 \), and all objects that move on the right direction are at the right of \( o_1 \) in \( \sigma_t \). If \( o_x \) exists, then it concludes that \( \sigma_t^Q(o_x) > k > b' \). It means that \( o_x \) is moved to the right of \( o_b \) in \( \sigma_t \), contradiction. So there are exactly \( b - b' \) objects in \( \{o_1, \ldots, o_b\} \) that are moving on the right direction.

By the above two results, we know that the number of objects in \( \{o_{a+1}, \ldots, o_b\} \) that are moving on the right direction is \( b - b' - a' + k - 1 \). \( o_a \) is swapped with \( o_a \), object \( o_b \) needs to be swapped with objects in \( \{o_{a+1}, \ldots, o_b\} \) that are moving on the right direction. So right before the swap including \( o_a \) and \( o_b \), object \( o_b \) is at agent \( b - (b-b' - a' + k - 1) = a' + b' - k + 1 \). Thus, \( c = a' + b' - k + 1 \). We know that (a) holds. Next, we consider (b).

For any agent \( q \) such that \( \max(a, b') \leq q < c \), both of \( o_a \) and \( o_b \) will visit it. By Lemma 3 we know that \( o_a \) will visit \( q \) before \( o_b \). By Observation 2, we know that \( o_a \succ q o_b \). For any agent \( q \) such that \( c \leq q \leq \min(a', b) \), both of \( o_a \) and \( o_b \) will visit it. By Lemma 3 we know that \( o_a \) will visit \( q \) before \( o_b \). By Observation 2, we know that \( o_a \succ q o_b \). Thus (b) holds.

See Figure 3 for an illustration for Lemma 3.

![Fig. 3 An illustration for Lemma 3](image)

Given a neat \( (o_1, o_n, k) \)-CONSTRAINED instance and an assignment \( \sigma_t \) such that \( \sigma_t^Q(o_1) = k \) and \( \sigma_t^Q(o_n) = k - 1 \). We show some conditions for \( \sigma_t \) to be a feasible assignment. For any two objects \( o_a \) and \( o_b \), we let \( a' = \sigma_t^Q(o_a) \) and \( b' = \sigma_t^Q(o_b) \). We say \( o_a \) and \( o_b \) are intersected if \( Q = [a, a'] \cap [b, b'] \) is not an empty set. There are three kinds of intersections, which are corresponding to Lemma 3(a), Lemma 3(b) and Lemma 3. We say a pair of objects \( o_a \) and \( o_b \) are compatible in assignment \( \sigma_t \) if there are either not intersected or intersected and satisfying one of the following:

1. when \( a' > a \) and \( b' > b \), it holds that \( a' < b' \) (corresponding to Lemma 3) and \( o_a \succ q o_b \) for all agents \( q \in Q \) (corresponding to Lemma 3(a));
2. when \( a' < a \) and \( b' < b \), it holds that \( a' < b' \) (corresponding to Lemma 3) and \( o_b \succ q o_a \) for all agents \( q \in Q \) (corresponding to Lemma 3(b));
3. when \( a' > a \) and \( b' < b \), it holds that \( c = a' + b' - k + 1 \in Q' = \{a\}, o_b \succ q o_a \) for all \( \max(a, b') \leq q < c \), and \( o_a \succ q o_b \) for all \( c \leq q \leq \min(a', b) \) (corresponding to Lemma 3).
An assignment $\sigma$ is compatible if it holds $\sigma(i) \neq o_i$ for any agent $i$ and any pair of objects in it are compatible.

Lemma 10 and Lemma 11 directly imply that

**Lemma 10** If $\sigma_t$ is a reachable assignment for a neat $(o_1, o_{n'}, k)$-constrained instance such that $\sigma_t^T(o_1) = k$ and $\sigma_t^T(o_{n'}) = k - 1$, then $\sigma_t$ is compatible.

**Lemma 11** Let $\sigma_t$ be a compatible assignment for a neat $(o_1, o_{n'}, k)$-constrained instance such that $\sigma_t^T(o_1) = k$ and $\sigma_t^T(o_{n'}) = k - 1$. For any two objects $o_{x-1}$ and $o_x$, if $\sigma_t^T(o_{x-1}) > x - 1$ and $\sigma_t^T(o_x) < x$, then the swap between agents $x - 1$ and $x$ in $\sigma_0$ is feasible. Let $\sigma_1$ denote the assignment after the swap between $x - 1$ and $x$ in $\sigma_0$. Then $\sigma_1$ is still compatible by taking $\sigma_1$ as the initial endowment.

**Proof** Consider the two objects $o_{x-1}$ and $o_x$, where $o_{x-1}$ is moving on the right direction and $o_x$ is moving on the left direction. They are compatible. By item 3 in the definition of the compatibility, we know that $c = x$, which implies the swap between agents $x - 1$ and $x$ is feasible in $\sigma_0$.

Next we show that if we take $\sigma_1$ as the initial endowment, the assignment $\sigma_1$ is still compatible. Compared with $\sigma_0$, the endowment positions of two objects are changed. To check whether $\sigma_1$ is compatible with $\sigma_1$ (being the initial endowment), we only need to check the compatibility of object pairs involving at least one of the two objects $o_{x-1}$ and $o_x$.

Now $\sigma_1(x - 1) = o_x$, $\sigma_1(x) = o_{x-1}$ and $\sigma_1(i) = o_i$ for all other $i \neq x - 1, x$.

First, we consider object pair $o_x$ and an object $o_i$ with $i \geq x$. If $o_x$ and $o_i$ are intersected, then the intersection can only be the case in Lemma 11(b). Item 2 in the definition of compatibility will hold because only the value of $a$ changes to a smaller value from $x$ to $x - 1$ and the domain of $Q$ will not increase.

Second, we consider object pair $o_x$ and an object $o_i$ with $1 \leq i \leq x - 2$. If $o_x$ and $o_{x-1}$ are intersected, there are two possible cases. When $\sigma_1(o_i) < i$, the intersection will be the case in Lemma 11(b). Item 2 in the definition of compatibility will still hold because only the value of $a$ changes to a smaller value from $x$ to $x - 1$ and the domain of $Q$ will not increase. When $\sigma_1(o_i) \geq i$, the intersection will be the case in Lemma 11(b). We show that item 3 in the definition of compatibility will hold. The value of $c$ for $o_x = o_i$ and $o_x = o_x$ is the same no matter taking $\sigma_0$ or $\sigma_1$ as the initial endowment, since none of $a'$, $b'$ and $k$ is changed. Note that when taking $\sigma_0$ as the initial endowment the value of $c$ is $x$ for $o_x = o_{x-1}$ and $o_x = o_x$. If the value of $c$ is also $x$ for $o_x = o_i$ and $o_x = o_x$ ($i \neq x - 1$), then we will get a contradiction that both of $o_{x-1}$ and $o_i$ will be assigned to the same agent in $\sigma_1$ by the computation formula of $c$. So we know that $c \neq x$ for $o_x = o_i$ and $o_x = o_x$. Thus, we get $c \in Q = [i + 1, \sigma_t^T(o_i)] \cap [\sigma_t^T(o_x), x - 1]$ because $c \in Q = [i + 1, \sigma_t^T(o_i)] \cap [\sigma_t^T(o_x), x]$ by $\sigma_t$ being compatible with $\sigma_0$ and $c \neq x$. After taking $\sigma_1$ as the initial endowment, the values of $a$, $a'$, $c$ and $b'$ will not change, and the value of $b$ changes from $x$ to a smaller value $x - 1$. We can see that the follows still hold: $o_b \geq q$ for all $a, a' \leq q < c$, and $a_b \geq q$ for all $c \leq q \leq \min(a', b)$.

Third, we consider $o_{x-1}$ and an object $o_i$ with $1 \leq i \leq x - 1$. If $o_{x-1}$ and $o_i$ are intersected, then the intersection can only be the case in Lemma 11(a). Item 1 in the definition of compatibility will still hold because only the value of $b$ changes to a bigger value from $x - 1$ to $x$ and the domain of $Q$ will not increase.
Fourth, we consider \( o_{x-1} \) and an object \( o_i \) with \( i \geq x \). If \( o_{x-1} \) and \( o_i \) are intersected, there are two possible cases. When \( \sigma_{t}(o_i) > i \), the intersection will be the case in Lemma 8(a). Item 1 in the definition of compatibility will still hold because only the value of \( a \) changes to a bigger value from \( x - 1 \) to \( x \) and the domain of \( Q \) will not increase. When \( \sigma_{t}(o_i) < i \), the intersection will be the case in Lemma 9. Analogously, we use similar arguments for the second case, we can prove that item 3 in the definition of compatibility holds.

So if \( \sigma_{t} \) is compatible by taking \( \sigma_{0} \) as the initial endowment, then it is compatible by taking \( \sigma_{1} \) as the initial endowment.

Based on Lemma 11 we will prove the following lemma.

**Lemma 12** Let \( \sigma_{t} \) be an assignment for a neat \((o_1, o_{n'}, k)\)-Constrained instance such that \( \sigma^T_{t}(o_1) = k \) and \( \sigma^T_{t}(o_{n'}) = k-1 \). If \( \sigma_{t} \) is compatible, then \( \sigma_{t} \) is a reachable assignment.

**Proof** When \( \sigma_{t} \) is compatible, we use the following algorithm to find a sequence of swaps from \( \sigma_{0} \) to \( \sigma_{t} \). For \( i \) from 1 to \( k - 1 \), we do that: let \( \sigma_{t}(i) = o \) and move object \( o \) from its current position to agent \( i \) by a sequence of swaps including it (if the current position of \( o \) is agent \( j \), then it is a sequence of \( |j - i| \) swaps). We prove the correctness of this algorithm: each swap is feasible and finally we will get the assignment \( \sigma_{t} \).

We first show that the first loop of the algorithm can be executed, i.e., \( \sigma_{t}(1) \) can be moved to agent 1 by a sequence of swaps. The first swap between agents \( x - 1 \) and \( x \) in \( \sigma_{0} \) is feasible by Lemma 11. We use \( \sigma_{t} \) to denote the assignment after this swap in \( \sigma_{0} \). Then \( \sigma_{t} \) is compatible by taking \( \sigma_{1} \) as the initial endowment by Lemma 11. By applying Lemma 11 again and again, we know that the swap between agents \( x - 1 - i \) and \( x - i \) in \( \sigma_{t} \) is feasible (for \( i > 0 \)). We use \( \sigma_{t+1} \) to denote the assignment after the swap in \( \sigma_{t} \) and assume that object \( o_{x} \) is assigned to agent 1 in \( \sigma_{t+1} \). By Lemma 11 we know that the sequence of swaps from \( \sigma_{0} \) to \( \sigma_{t+1} \) are feasible and \( \sigma_{t+1} \) is compatible by taking \( \sigma_{t+1} \) as the initial endowment.

Next, we consider \( \sigma_{t+1} \) as the initial endowment. Agent 1 has already gotten object \( o_{x} \) and we can ignore it. After deleting agent 1 and object \( o_{x} \) from the instance, we get a new \((o_1, o_{n'}, k)\)-Constrained instance. The second loop of the algorithm is corresponding to the sequence of swaps in the new instance to move an object to the first agent, the correctness of which can be proved by the above argument for the first loop.

By iteratively applying the above arguments, we can prove that each loop of the algorithm can be executed legally. Therefore, all swaps in the algorithm are feasible.

Let \( \sigma_{t'} \) be the assignment returned by the algorithm. It holds that \( \sigma_{t'}(i) = \sigma_{t}(i) \) for \( i \leq k - 1 \). For \( i \geq k \), we show that \( \sigma_{t'}(i) = \sigma_{t}(i) \) still holds. For any object \( o_j = \sigma_{t'}(i) \) with \( i \geq k \), objects \( o_j \) and \( o_{n'} \) are intersected and the intersection can only be the case of Lemma 9. By item 3 in the definition of compatibility, we know the value \( c \) will not change no matter what the endowment position of \( o_j \) is. Thus, \( o_j \) and \( o_{n'} \) will be swapped between two fixed agents and this is the last swap including \( o_j \). The object \( o_j \) will arrive at the position \( a' = \sigma_{t'}^T(o_j) \). Therefore, \( \sigma_{t'} = \sigma_{t} \).

By Lemma 12 to solve a neat \((o_1, o_{n'}, k)\)-Constrained instance, we only need to find a compatible assignment.
3.1 Computing Compatible Assignments

In a compatible assignment, object $o_1$ will be assigned to agent $k$ and object $o_{n'}$ will be assigned to agent $k-1$. We consider other objects $o_i$ for $i \in \{2,3,\ldots,n'-1\}$. In a compatible assignment, object $o_i$ will not be assigned to agent $i$ since each agent will attend in at least one swap including object $o_1$ or $o_{n'}$. There are two possible cases: $o_i$ is assigned to agent $i'$ such that $i' < i$; $o_i$ is assigned to agent $i''$ such that $i'' > i$. We say that $o_i$ is moved to the left side for the former case and moved to the right side for the latter case. We will show that for each direction, there is only one possible position for each object $o_i$ in a compatible assignment.

First, we consider $i \in \{2,3,\ldots,k-1\}$. Assume that object $o_i$ is moved to the left side in a compatible assignment. Thus, $o_1$ and $o_i$ are intersected and the intersection is of the case in Lemma 9. We find the index $i'$ such that $i' < i$. $o_1$ is the only possible agent for object $o_i$ to make $o_1$ and $o_i$ compatible if $o_i$ is moved to the left side. We use $i_l$ to denote this agent if it exists for $i$. Assume that object $o_i$ is moved to the right side in a compatible assignment. Since $o_1$ will be moved to agent $k$ and $o_i$ will be moved to the right side, by Lemma 3 we know that $o_i$ will be moved to the right of $o_1$, i.e., an agent $i''$ with $i'' > k$. Thus, $o_1$ and $o_{n'}$ are intersected and the intersection is of the case in Lemma 9. We find the index $i'$ such that $i'' > k$, $o_i$ in $o_{n'}$ and $o_{n'}$ in $o_i$ for each $j \in \{k-1,k,\ldots,i''-1\}$. We can see that $i'$ is the only possible agent for object $o_i$ to make $o_{n'}$ and $o_i$ compatible if $o_i$ is moved to the right side. We use $i_r$ to denote this agent if it exists for $i$.

Second, we consider $i \in \{k,k+1,\ldots,n'-1\}$. In fact, the structure of neat $(o_1,o_{n'}),k$-CONSTRAINED instances is symmetrical. We can rename the agents on the path from left to right as $\{n',n'-1,\ldots,1\}$ instead of $\{1,2,\ldots,n'\}$ and then this case becomes the above case. We can compute $i_l$ and $i_r$ for each $i \in \{k,k+1,\ldots,n'-1\}$ if they exist.

If none of $i_l$ and $i_r$ exists for some $i$, then this instance is a no-instance. If only one of $i_l$ and $i_r$ exists, then object $o_i$ must be assigned to this agent in any compatible assignment. The hardest case is that both of $i_l$ and $i_r$ exist, where we may not know which agent the object will be assigned to in the compatible assignment. For this case, we will rely on algorithms for 2-SAT to find possible solutions.

For each agent $j$, we will use $R_j$ to store all possible objects that may be assigned to agent $j$ in a compatible assignment. We use the following procedure to maintain $R_j$. Initially, we let $R_{k-1} = \{o_{n'}\}$, $R_k = \{o_1\}$, and $R_{l} = \emptyset$ for all other agent $i$. Then for each $i \in \{2,3,\ldots,n'-1\}$, we compute $i_l$ and $i_r$ and add $o_i$ into $R_{i_l}$ and $R_{i_r}$. After this, we can do the following to make the size of each $R_j$ at most 2. While there is a set $R_{j_0}$ becoming an empty set, stop and report the instance is a no-instance; while there is a set $R_{j_0}$ containing only one object $o_{n}$ and the object $o_{n}$ appears in two sets $R_{j_0}$ and $R_{j_0'}$, delete $o_{n}$ from $R_{j_0'}$.

The correctness of the third step is based on the fact that agent $j_0$ should get one object. If there is only one candidate object $o_{j_0}$ for agent $j_0$, then $o_{j_0}$ can only be assigned to agent $j_0$ no possible to agent $j_0'$. We also analyze the running time of the above procedure to compute $R_j$. For each object $o_i$, we can compute $i_l$ and $i_r$ in $O(n)$. Therefore, we use $O(n^2)$ time to
set the values for all sets \( R_j \) in the first two steps. To update \( R_i \), we may execute at most \( n \) iterations in the third step and each iteration can be executed in \( O(n) \).

Therefore, the procedure running times in \( O(n^2) \) time.

\textbf{Lemma 13} After the above procedure, either the instance is a no-instance or it holds that \( 1 \leq |R_j| \leq 2 \) for each \( j \in \{1, 2, \ldots, n'\} \).

\textit{Proof} We only need to consider the latter case where each set \( R_j \) containing at least one object after the procedure. Note that for each set \( R_j \) containing only one object, the object will not appear in any other set. We ignore these singletons \( R_j \) and consider the remaining sets. Each remaining set containing at least two objects. On the other hand, each other object \( o_i \) (not appearing in a singleton) can be in at most two sets \( R_i \) and \( R_i' \). On average each remaining set \( R_j \) contains at most two objects. Therefore, each remaining set \( R_j \) contains exactly two objects.

\[ \square \]

For a set \( R_j \) containing only one object \( o_i \), we know that the object \( o_i \) should be assigned to agent \( j \) in any compatible assignment. For sets \( R_j \) containing two objects, we still need to design which object is assigned to this agent such that we can get a compatible assignment.

We will reduce the remaining problem to 2-Sat. The instance contains \( n' \) variables \( \{x_1, x_2, \ldots, x_{n'}\} \) corresponding to the \( n' \) objects. When \( x_i = 1 \), it means that the object \( o_i \) moves on the right direction, i.e., we will assign it to the agent \( i_r \) in the compatible assignment. When \( x_i = 0 \), it means that the object \( o_i \) moves on the left direction and we will assign it to the agent \( i_l \) in the compatible assignment.

We have two kinds of clauses, called agent clauses and compatible clauses.

For each set \( R_j \), we associate \( |R_j| \) literals with it. If there is an object \( o_i \) such that \( i_l = j \), we associate literal \( \overline{x} \) with \( R_j \); if there is an object \( o_i \) such that \( i_r = j \), we associate literal \( \overline{x} \) with \( R_j \). For each set \( R_j \) of size 1 (let the associated literal be \( \ell_j \)), we construct one clause containing only one literal \( c_j : \ell_j \). For each set \( R_j \) of size 2 (let the associated literals be \( \ell_j^r \) and \( \ell_j^l \)), we construct two clauses \( c_{j1} : \ell_j^r \lor \ell_j^l \) and \( c_{j2} : \overline{\ell_j^r} \lor \overline{\ell_j^l} \). These clauses are called \textit{agent clauses}. The agent clauses are to guarantee that exactly one object is assigned to each agent.

For each pair of sets \( R_j \) and \( R_i \), we construct several clauses according to the definition of compatibility. For any two objects \( o_j \in R_j \) (the corresponding literal associated to \( R_j \) is \( \ell_j \)) and \( o_i' \in R_i \) (the corresponding literal associated to \( R_i \) is \( \ell_i \)), we say that \( \ell_j \) and \( \ell_i \) are \textit{compatible} if \( o_j \) and \( o_i' \) are compatible when \( o_j \) is assigned to agent \( j \) and \( o_i' \) is assigned to agent \( i \) in the assignment. If \( \ell_j \) and \( \ell_i \) are not compatible, then either \( o_j \) cannot be assigned to agent \( j \) or \( o_i' \) cannot be assigned to agent \( i \) in any compatible assignment. So we construct one \textit{compatible clause}: \( \overline{\ell_j} \lor \overline{\ell_i} \) for each pair of incompatible \( \ell_j \) and \( \ell_i \). Since each set contains at most two objects, for each pair of sets \( R_j \) and \( R_i \), we will create at most \( 2 \times 2 = 4 \) clauses. In fact, when there are four clauses for a pair, the instance will become a no-instance, since no matter what objects assigned to agents \( j \) and \( i \), there is no compatible assignment. In the following example, we will illustrate the construction of agent clauses and compatible clauses.

We can see that each clause contains at most two literals and then the instance is a 2-Sat instance. The construction of the 2-Sat instance implies that

\textbf{Lemma 14} The 2-Sat instance has a feasible assignment if and only if the corresponding neat \((o_1, o_{i'}, k)\)-Constrained instance has a compatible assignment.
The main steps of the whole algorithm to solve **Strict Object Reachability** in a path are listed in Algorithm 1. The correctness of the algorithm follows from Lemma 5, Lemma 6, Lemma 7 and Lemma 14. Next, we analyze the running time bound of it. The dominating part of the running time is taken by the computation for neat \((o_1, o_{n'}, k)\)-Constrained instances. Next, we consider the running time used for solving each neat \((o_1, o_{n'}, k)\)-Constrained instance. By the above analysis, we use \(O(n^4)\) time to compute the final values for all sets \(R_j\). To construct 2-Sat instance, we construct at most 2\(n\) agent clauses in \(O(n^2)\) time and construct at most \(4\binom{n}{2}\) compatible clauses, each of which will take \(O(n^2)\) time to check the compatibility. So the 2-Sat instance can be constructed in \(O(n^3)\) time. We use the \(O(n^2+m)\)-time algorithm for 2-Sat \([23]\) to solve our instance, where \(m = O(n^2)\). There are at most \(n\) neat \((o_1, o_{n'}, k)\)-Constrained instances. In total, we use \(O(n^4)\) time.

**Algorithm 1:** Main steps to solve **Strict Object Reachability** in a path

**Input:** An instance \((N, O, \succ, P, \sigma, k \in N, o_1 \in O)\)

**Output:** To determine whether \(o_1\) is reachable for \(k\)

1. for \(k \leq n' \leq n\) do
2. Construct the neat \((o_1, o_{n'}, k)\)-Constrained instance by deleting agent \(i\) and object \(o_i\) for all \(n' < i \leq n\);
3. Compute \(i_r\) and \(i_l\) for all \(1 \leq i \leq n'\) if it exist according to Lemma 9;
4. Construct the set \(R_j\) of all possible objects that may be assigned to each agent \(j\) according to the result in the above step, where \(R_{k-1} = \{o_{n'}\}\) and \(R_k = \{o_1\}\);
5. Iteratively update \(R_j\) according to the procedure before Lemma 13;
6. Construct a 2-Sat instance as follows: construct a variable for each object, construct agent clauses according to \(R_j\), and construct compatible clauses for all incompatible pairs;
7. Determine whether the 2-Sat instance is satisfiable;
8. if the 2-Sat instance is yes then
   return yes;
9. return no.

**Theorem 1** **Strict Object Reachability** in a path can be solved in \(O(n^4)\) time.

We give an example to show the steps to compute a compatible assignment for a neat \((o_1, o_{n'}, k)\)-Constrained instance.

**Example 2** Consider a neat \((o_1, o_{n'}, k)\)-Constrained instance with \(n' = 8\) and \(k = 5\) as Figure 4.

We compute \(i_r\) and \(i_l\) for all \(1 \leq i \leq n'\) by the above procedure, the values of which are listed Table 2.

| agent \(i\) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|------------|---|---|---|---|---|---|---|---|
| \(i_l\)   | - | 1 | 2 | 3 | 2 | 3 | - | 4 |
| \(i_r\)   | 5 | 6 | 6 | 7 | 6 | 7 | 8 | - |

**Table 2** The values of \(i_l\) and \(i_r\)
We construct $R_j$ for each agent $j$ according to this table, and then update them as doing in the procedure. After the update, it holds that $1 \leq |R_j| \leq 2$ for all $1 \leq j \leq n'$. We reduce the remaining problem to 2-Sat. The agent clauses for each set $R_j$ are given in the last column of Table 3.

| Set | Initial | Updated | Agent clauses |
|-----|---------|---------|---------------|
| $R_1$ | $\{o_3\}$ | $\{o_2\}$ | $x_2$ |
| $R_2$ | $\{o_2, o_5\}$ | $\{o_3, o_5\}$ | $x_3 \lor x_5, x_4 \lor x_5$ |
| $R_3$ | $\{o_4, o_6\}$ | $\{o_4, o_6\}$ | $x_4 \lor x_6, x_4 \lor x_6$ |
| $R_4$ | $\{o_9\}$ | $\{o_3\}$ | $x_3$ |
| $R_5$ | $\{o_1\}$ | $\{o_1\}$ | $x_1$ |
| $R_6$ | $\{o_2, o_3, o_5\}$ | $\{o_3, o_5\}$ | $x_3 \lor x_5, x_4 \lor x_5$ |
| $R_7$ | $\{o_4, o_6\}$ | $\{o_4, o_6\}$ | $x_4 \lor x_6, x_4 \lor x_6$ |
| $R_8$ | $\{o_7\}$ | $\{o_7\}$ | $x_7$ |

Table 3 $R_j$ and compatible clauses

Next, we construct compatible clauses for all incompatible pairs. We check all pairs of objects and find that there are only two incompatible cases: $o_4$ and $o_5$ are incompatible if $o_4$ and $o_5$ are moved to agent 3 and agent 2, respectively; $o_4$ and $o_5$ are incompatible if $o_4$ and $o_5$ are moved to agent 7 and agent 6, respectively. The compatible clauses are

$$x_4 \lor x_5 \quad \text{and} \quad \overline{x_4} \lor \overline{x_5}.$$  

By using the $O(n+m)$ time algorithm for 2-Sat \cite{23}, we get a feasible variables assignment $\langle 1,0,0,0,1,1,1,0 \rangle$ for the 2-Sat instance. The corresponding swap sequence constructed via variables assignment above is given in Figure 5.

**Remark:** Although Step 4 of our algorithm will compute possible values $i_l$ and $i_r$ for each $i$, it does not mean that object $o_i$ must be reachable for $i_l$ or $i_r$. In the above example, object $o_2$ is not reachable for agent $2_\ast = 6$ since agent 3 prefers its initial object $o_3$ to $o_2$ and then $o_2$ cannot go to the right direction. In our algorithm, the compatible clauses can avoid assigning an object to an unreachable value $i_l$ or $i_r$. In the above example, if object $o_2$ goes to agent 6, then some object $o_i$ with $i > 2$ will go to agent 1 or 2 and we will get an incompatible pair $o_2$ and $o_1$.  

Fig. 4 The instance of Example 2
4 Weak Preference Version in a Path

We have proved that Strict Object Reachability in a path is polynomially solvable. Next, we show that Weak Object Reachability in a path is NP-hard. One of the most important properties is that Lemma 1 will not hold for Weak Object Reachability and an object may ‘visit’ an agent more than one time. Our proof uses the same idea of the reduction in [17] to prove the NP-hardness of Strict Object Reachability in a tree.

Theorem 2 Weak Object Reachability is NP-hard even when the network is a path.

We give a reduction from the known NP-complete problem 2P1N-SAT [24]. In a 2P1N-SAT instance, we are given a set \( V = \{v_1, v_2, \ldots, v_n\} \) of variables and a set \( C = \{C_1, C_2, \ldots, C_m\} \) of clauses over \( V \) such that every variable occurs 3 times in \( C \) with 2 positive literals and 1 negative literal. The question is to check whether there is an variable assignment satisfying \( C \). For a 2P1N-SAT instance \( I_{SAT} \), we construct an instance \( I_{WOR} \) of Weak Object Reachability on a path such that \( I_{SAT} \) is a yes-instance if and only if \( I_{WOR} \) is a yes-instance.

The instance \( I_{WOR} \) contains \( 6n + m + 1 \) agents and objects, which are constructed as follows. There is an agent named \( T \). For each clause \( C_i \) (\( i \in \{1, \ldots, m\} \)), we introduce an agent also named \( C_i \). For each variable \( v_i \), we add six agents, named as \( X^n_i \), \( X^p_i \), \( X^q_i \), \( A^3_i \), \( A^2_i \) and \( A^1_i \). They form a path of length 5 in the order as showed below. We call the path a block and denote it by \( B_i \) as Figure 6. The names of the six agents have certain meaning: \( X^n_i \) means that the negative literal of \( v_i \) appears in the clause \( C_n_i \); \( X^p_i \) and \( X^q_i \) mean that the positive literals of \( v_i \) appear in the two clauses \( C_p_i \) and \( C_q_i \); \( A^3_i \), \( A^2_i \) and \( A^1_i \) are three auxiliary agents.

![Block B_i](image)

Fig. 6 Block \( B_i \)

The whole path is connected in the order showed as Figure 7.

In the initial assignment \( \sigma_0 \), object \( t \) is assigned to agent \( T \), object \( c_i \) is assigned to agent \( C_i \) for \( i \in \{1, 2, \ldots, m\} \), object \( a^j_i \) is assigned to agent \( A^j_i \) for each \( i \in \{1, 2, \ldots, m\} \).
The whole path

\[ B_n \rightarrow \cdots \rightarrow B_1 \rightarrow \underbrace{C_m \cdots C_1} \rightarrow T \]

**Fig. 7** The whole path

\{1, 2, \ldots, n\} and \( j \in \{1, 2, 3\} \), and \( \overline{c}_i^p \) (resp., \( o_i^p \) and \( o_i^n \)) is assigned to agent \( X_i^p \) (resp., \( X_i^p \) and \( X_i^n \)) for each \( i \in \{1, 2, \ldots, n\} \).

Next, we define the preference profile \( \succeq \). We only show the objects that each agent prefers at least as its initial one and all other objects can be put behind its initial endowment in any order. The initial endowment is denoted by a square in the preference. Let \( L_i \) be the set of the objects associated with the literals in clause \( C_i \), i.e., \( L_i \) is the set of objects \( \overline{c}_i^a \), \( o_i^b \) and \( o_i^n \) with \( n_a = i \), \( p_b = i \) or \( q_c = i \). For each variable \( v_i \), we define a set of objects \( W_i = \{c_1, \ldots, c_m\} \cup \{\overline{c}_i^p : j > i\} \cup \{o_i^p : j > i\} \cup \{a_j : j < i, l = 1, 2, 3\} \). We are ready to give the preferences for the agents.

First, we consider the preferences for \( T \) and \( C_i \). The following preferences ensure that when \( C_i \) holds an object in \( L_i \) for each \( i \in \{1, \ldots, m\} \), object \( t \) is reachable for agent \( C_m \) via a sequence of \( m \) swaps between \( C_i \) and \( C_{i-1} \) for \( i = 1, \ldots, m \), where \( C_0 = T \). Note that the squares in the preferences indicate the initial endowments for agents.

\[
T : L_1 \succ \square ; \\
C_i : L_{i+1} \succ t \succ L_i \succ c_1 \succ L_{i-1} \succ \cdots \succ c_{i-1} \succ L_1 \succ c_i , \text{ for all } 1 \leq i \leq m - 1 ; \\
C_m : t \succ L_m \succ c_1 \succ L_{m-1} \succ \cdots \succ c_{m-1} \succ L_1 \succ c_m .
\]

Next, we consider the preferences for the agents in each block \( B_i \). The following preferences ensure that at most one of \( \overline{c}_i^p \) and \( \{o_i^p, o_i^n\} \) can be moved to the right of the block, which will indicate the corresponding variable is either true or false. If \( \overline{c}_i^p \) is moved to the right of the block, we will assign the corresponding variable false; if some of \( \{o_i^p, o_i^n\} \) is moved to the right of the block, we will assign the corresponding variable true. We use the preference of \( A_i^3 \) to control this. Furthermore, we use \( A_i^1, A_i^2 \) and \( A_i^3 \) to (temporarily) hold \( \overline{c}_i^p \) (or \( o_i^p \) and \( o_i^n \)) if they do not need to be moved to the right of the block.

\[
X_i^c_i : W_i \cup \{a_i^1, a_i^3, a_i^4, o_i^n, o_i^p\} \succ \overline{c}_i^p , \\
X_i^q_i : W_i \cup \{a_i^1, a_i^3, a_i^4, \overline{c}_i^p, o_i^p\} \succ o_i^p , \\
X_i^o_i : W_i \cup \{a_i^1, a_i^3, a_i^4, o_i^n, \overline{c}_i^p\} \succ o_i^n , \\
A_i^1 : W_i \cup \{a_i^1, a_i^3, a_i^4, o_i^n, o_i^p, \overline{c}_i^p\} , \\
A_i^2 : W_i \cup \{a_i^1, a_i^3, a_i^4, o_i^n, o_i^p, \overline{c}_i^p\} , \\
A_i^3 : W_i \cup \{a_i^1, a_i^3, o_i^n, o_i^p\} \succ \overline{c}_i^p \succ a_i^3 \] , \text{ for all } 1 \leq i \leq n .
\]

The instance \( I_{WOR} \) is to determine whether object \( t \) is reachable for agent \( C_m \).

**Lemma 15** A 2P1N-SAT instance \( I_{SAT} \) is yes if and only if the corresponding instance \( I_{WOR} \) of Weak Object Reachability in a path is yes.
Proof If \( t \) is reachable for \( c_m \), there are \( m \) swaps including \( t \) which happen between \( C_i \) and \( C_{i+1} \) for \( i \in \{0, 1, \ldots, m-1\} \), where \( C_0 = T \). Note that the swap between \( C_i \) and \( C_{i+1} \) (where \( C_i \) holds the object \( t \)) can happen if and only if \( C_{i+1} \) holds an item \( a \in L_{i+1} \). We can let the literal corresponding to the item in \( a \in L_{i+1} \) to be 1 for all agents \( C_i \) to get a satisfying assignment for \( I_{SAT} \) because the construction of each block \( B_i \) does not allow both \( X_i^{pi} \) and one of \( o_i^{pi} \) and \( o_i^{qi} \) moving to the right of this block and we proof it as follows.

Suppose there is a block \( B_i \) that both \( X_i^{pi} \) and one of \( o_i^{pi} \) and \( o_i^{qi} \) move to the right of this block. When \( X_i^{pi} \) moves to the right of this block, it must reach \( A_i^3 \) at some time. We know that \( X_i^{pi} \) is the last but one in agent \( A_i^3 \)'s given preferences and it implies that before \( X_i^{pi} \) is swapped with \( a_i^j \) between \( X_i^q \) and \( A_i^3 \), \( A_i^3 \) cannot take any exchange. Therefore, the only way to move \( X_i^{pi} \) to \( A_i^3 \) is that \( X_i^{pi} \) is swapped with \( o_i^{pi} \), \( o_i^{qi} \) and \( a_i^j \) in order. For each agent of \( \{X_i^{pi}, X_i^q, X_i^o\} \), if the object she initially held is swapped with her neighbor’s one, then she won’t accept it anymore. Thus, both \( o_i^{pi} \) and \( o_i^{qi} \) cannot move to the right of this block, it is a contradiction.

On the other hand, if there is a satisfying assignment \( \tau \) for \( I_{SAT} \), we can construct a reachable assignment for \( I_{WOR} \). For each variable \( v_i \), if it is true in \( \tau \), we move \( o_i^{pi} \) and \( o_i^{qi} \) to agents \( A_i^3 \) and \( A_i^2 \); if it is false in \( \tau \), we move \( X_i^{pi} \) to agent \( A_i^2 \). These objects are called true objects. All these happen within each block. After this procedure, we move one of true objects \( o_i^{pi} \), \( o_i^{qi} \) and \( X_i^{pi} \) to agent \( C_j \) with smaller \( j \) first gets its object in \( L_j \) (for every agent in blocks, almost all objects in her given preferences are indifferent with each other). During this procedure, once another
true object is moved out of its position \( A_i^3 \) or \( A_i^4 \) (this may happen when the true object is on the moving path of another true object to \( C_j' \) with \( j < j' \)), we will simply move it back by one swap (\( A_i^3 \), \( A_i^4 \) and \( A_i^4 \) have same preferences on almost all objects except \( \tau_i^a \) and \( \tau_i^b \)). So in each iteration only one true object is moved out of its current position \( A_i^3 \) or \( A_i^4 \) and it is moved to its final position \( C_j \) directly.

According to the reduction in Theorem 2 we know that if there exists a reachable assignment \( \sigma \) in \( I_{WOR} \) such that for each agent \( i \in N \) and each object \( o_j \in O \), \( \sigma(i) \succeq_i o_j \), then object \( t \) is reachable for agent \( C_m \) and it implies that \( I_{SAT} \) is true. If \( I_{SAT} \) is true, then by the construction of solutions from \( I_{SAT} \) to \( I_{WOR} \) we know that there exists a reachable assignment \( \sigma \) such that for each agent \( i \in N \) and each object \( o_j \in O \), \( \sigma(i) \succeq_i o_j \). In short, \( I_{SAT} \) is true if and only if there exists a reachable assignment in \( I_{WOR} \), where each agent takes his most preferred object. So we can reduce Weak Object Reachability in a path to Weak Pareto Efficiency in a path. We only need to check where in a Pareto optimal assignment all agents get their most preferred objects.

**Theorem 3** Weak Pareto Efficiency is NP-hard even when the network is a path.

The above results also imply the following theorem. Assume that each agent has a value function on the objects such that the most preferred object has the maximum value for him. Thus,

**Theorem 4** To find a reachable assignment which maximizes the sum of values for all the agents is NP-hard even when the network is a path.

5 Weak Preference Version in a Star

We have shown that Weak Object Reachability is NP-hard when the graph is a path. In this section, we show that Weak Object Reachability can be solvable in polynomial time if the graph is a star.

We assume the network is a star of \( n \) vertices and the center agent of the star is \( n \). We also assume without loss of generality that \( a_1 \succeq_n a_2 \succeq_n \cdots \succeq_n a_n \). The problem asks whether object \( a_l \) is reachable for agent \( k \). When agent \( k \) is a leaf in the star, the problem can be reduced to the problem of whether the center agent \( n \) can get object \( a_l \), since only when agent \( n \) gets object \( a_l \), agent \( k \) can swap with agent \( n \) to get object \( a_l \). Note that if we want to move object \( a_l \) to agent \( n \), then any object \( a_i \) with \( a_l \succeq_n a_i \) cannot be swapped to agent \( n \) before \( a_l \). So we can simply ignore all objects \( a_i \) with \( a_l \succeq_n a_i \). Since we have assumed that \( a_l \succeq_n a_j \) for \( i < j \), next we simply assume without loss of generality that the problem is to check whether object \( a_l \) is reachable for agent \( n \). We have the following lemma.

**Lemma 16** If there is a sequence of swaps \((\sigma_0, \sigma_1, \ldots, \sigma_t)\) such that \( \sigma_t(n) = a_l \), then there is a sequence of swaps \((\sigma_0, \sigma_1', \ldots, \sigma_t')\) such that \( \sigma_t'(n) = a_l \) and each leaf agent in the star attends in at most one swap.

**Proof** Let \( \psi = (\sigma_0, \sigma_1, \ldots, \sigma_t) \) be one of the shortest sequences of swaps such that \( \sigma_t(n) = a_l \). Assume to the contrary that at least one leaf agent attends in two
swaps in $\psi$. Then $\sigma_i(n) = \sigma_j(n)$ will hold for two different indices $0 \leq i < j \leq t$.
We assume that $j$ is the maximum index such that $\sigma_{i'}(n) = \sigma_j(n)$ holds for some
index $i' \neq j$. By the selection of the index $j$, we know that any leaf agent that
attends in a swap in $(\sigma_j, \sigma_{j+1}, \ldots, \sigma_t)$ will not attend any swap before $\sigma_j$. Thus,
after deleting the sequence of swaps in $(\sigma_i, \sigma_{i+1}, \ldots, \sigma_j)$ from $\psi$, it is still a feasible
sequence of swaps. However, the length of the new sequence of swaps is shorter
than that of $\psi$, a contradiction to the choice of $\psi$. So the lemma holds. \qed

For an instance of Weak Object Reachability in a star, a sequence of swaps is
called simple if no leaf agent attends in two swaps. By Lemma 16 we know that
to solve Weak Object Reachability in a star, we only need to check whether
there is a simple sequence of swaps $(\sigma_0, \sigma_1, \ldots, \sigma_t)$ such that $\sigma_t(n) = o_1$.

Our idea to solve this special problem is coming from the algorithm for Strict
Object Reachability in a star in [17]. Note that for a simple sequence of swaps,
in each swap, the center agent and a leaf agent make a rational trade when the
leaf agent holds his endowed object. We construct a directed graph
Object Reachability
in a star in [17]. Note that for a simple sequence of swaps,

Weak Object Reachability
For an instance
\begin{equation}
\sigma(n) = (\sigma_0, \sigma_1, \ldots, \sigma_t)
\end{equation}
in a star, a sequence of swaps is

Weak Pareto Efficiency
For
\begin{equation}
I = (V,A)
\end{equation}
in a star, we do not find a polynomial-time
algorithm or a proof of the NP-hardness. On the other hand, we can prove the

Theorem 5
To find a reachable assignment that maximizes the sum of values for
all the agents is NP-hard even when the network is a star.

To prove Theorem 5, we give a reduction from the known NP-complete problem
– the Directed Hamiltonian Path problem [22], which is to find a directed path
visiting each vertex exactly once in a directed graph $D = (V,A)$. We construct an
instance $I$ of the problem in Theorem 5 such that $I$ admits a reachable assignment
with the total social welfare at least $3|V| + |A| - 1$ if and only if the graph $D$ has a
directed Hamiltonian path starting from a given vertex $s \in V$. We simply assume
that $s$ has no incoming arcs.

The instance $I$ is constructed as follows. It contains $|V| + |A| + 1$ agents and
$|V| + |A| + 1$ objects. For each arc $e \in A$, there is an associated agent $a_e$, called an
arc agent. For each vertex $v \in V$, there is an associated agent $a_v$, called a vertex
agent. So there are $|A|$ arc agents and $|V|$ vertex agents. There is also a center
agent $a_c$, which is adjacent to all arc agents and vertex agents to form a star.

In the initial endowment $\sigma_0$, the object assigned to the center agent $a_c$ is $o_c$,
called the center object; for each vertex agent $a_v$, the object assigned to it is $o_v$,
called a vertex object; and for each arc agent $a_e$, the object assigned to it is $o_e$,
called an arc object.
Next, we construct the value function for each agent. For the center agent $a_c$, all objects have the same value of 0. For the starting vertex agent $a_s$, the initial object $o_s$ is valued as 1, the center object $o_c$ is valued as 2, and all other objects are valued as 0. For any vertex agent $a_v$ with $v \in V \setminus \{s\}$, the initial object $o_v$ is valued as 1, arc object $o_e$ with $e$ being an arc from a vertex to the vertex $v$ is valued as 2, and all other objects are valued as 0. For any arc agent $a_e$ with $e \in A$, where $e = \overrightarrow{uv}$ is an arc starting from $u$ to $v$, the initial object $o_e$ is valued as 1, the vertex object $o_u$ is valued as 2, and all other objects are valued as 0.

See Figure 10 and Table 4 for an example to construct the instance.

![Fig. 10 An example of the construction in the proof of Theorem 5](image)

| Objects | $a_s$ | $a_v$ | $a_t$ | $a_u$ | $a_{s\rightarrow}$ | $a_{v\rightarrow}$ | $a_{t\rightarrow}$ | $a_{u\rightarrow}$ |
|---------|-------|-------|-------|-------|-----------------|-----------------|-----------------|-----------------|
| $o_c$   | 0     | 2     | 0     | 0     | 0               | 0               | 0               | 0               |
| $o_s$   | 0     | 1     | 0     | 0     | 0               | 2               | 0               | 0               |
| $o_t$   | 0     | 0     | 0     | 1     | 0               | 0               | 2               | 0               |
| $o_u$   | 0     | 0     | 0     | 0     | 1               | 0               | 0               | 2               |
| $o_{s\rightarrow}$ | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| $o_{v\rightarrow}$ | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| $o_{t\rightarrow}$ | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 |
| $o_{u\rightarrow}$ | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 |

Table 4 The value functions of the example in Figure 10

From the value functions of the instance $I$, we can see the following properties.

(P1) the total social welfare of the initial endowment is exactly $|V| + |A|$;
(P2) the vertex agent $s$ can only attend in one swap that is a trade between agents $a_s$ and $a_c$, where agent $a_s$ must hold the initial object $o_s$ and agent $a_c$ must hold the initial object $o_c$;
(P3) the first swap must happen between agents $a_s$ and $a_c$, because all other agents value object $o_s$ as 0;
(P4) for a vertex object $o_v$, among all vertex agents, only vertex agent $a_v$ values it as 1 and all other vertex agents value it as 0;
(P5) for a vertex object $o_v$, among all arc agents, only arc agent $a_e$ with $e$ being an arc starting from vertex $v$ values it as 2, and all other arc agents value it as 0;
(P6) for an arc object \( o_e \), among all vertex agents, only vertex agent \( a_v \) with \( v \) being the ending point of arc \( e \) values it as 2 and all other vertex agents value it as 0;

(P7) for an arc object \( o_e \), among all arc agents, only arc agent \( a_e \) values it as 1 and all other arc agents value it as 0.

From (P4), (P5), (P6) and (P7), we can see that

(P8) when a vertex object \( o_v \) is held by the center agent \( a_c \), the next swap can only be a trade between the center agent \( a_c \) and an arc agent \( a_e \) with \( e \) being an arc starting from vertex \( v \);

(P9) when an arc object \( o_e \) is held by the center agent \( a_c \), the next swap can only be a trade between the center agent \( a_c \) and the vertex agent \( a_v \) with \( v \) being the ending point of arc \( e \).

Based on the above properties, we prove the following results.

**Lemma 18** Instance \( I \) has a reachable assignment such that the total value of all agents is at least \( 3|V| + |A| - 1 \) if and only if there is a directed Hamiltonian path starting from vertex \( s \) in the directed graph \( D \).

**Proof** Assume that there is a directed Hamiltonian path \( v_1v_2v_3\ldots,v_{|V|} \) starting from \( s \) in the directed graph \( D \), where \( v_1 = s \) and the arc from \( v_i \) to \( v_{i+1} \) is denoted by \( e_i \). We show that there is a satisfying sequence of swaps in \( I \). Note that each swap in a star happens between the center agent and a leaf agent. So a sequence of swaps can be specified by a sequence of leaf agents. So we will use a sequence of leaf agents to denote a sequence of swaps. In fact, we will show that \( (a_{v_1}, a_{e_1}, a_{v_2}, a_{e_2}, \ldots, a_{e_{|V|-1}}, a_{v_{|V|-1}}) \) is a satisfying sequence of swaps in \( I \).

It is easy to verify that the first two swaps can be executed. We can prove by induction that, for each \( i > 1 \), before the swap between \( a_e \) and \( a_{v_i} \) (resp., \( a_{e_i} \)), agent \( a_c \) holds object \( o_{e_{i-1}} \) (resp., \( o_{e_{i-1}} \)) and agent \( a_{v_i} \) holds object \( o_{v_{i-1}} \) (resp., agent \( a_{e_{i-1}} \) holds object \( o_{c_{i-1}} \)). According to (P4), (P5), (P6), (P7) and the fact that the center agent \( a_c \) has no difference among all the objects, we know that the sequence of swaps \( (a_{v_1}, a_{e_1}, a_{v_2}, a_{e_2}, \ldots, a_{e_{|V|-1}}, a_{v_{|V|-1}}) \) can be executed. Furthermore, for each swap in the sequence, the leaf agent attended in the swap will increase his value by 1. There are \( 2|V| - 1 \) swaps and the initial endowment has a value of \( |V| + |A| \). So in the final assignment, the total value is \( |V| + |A| + (2|V| - 1) = 3|V| + |A| - 1 \).

Next, we assume that there is a reachable assignment \( \sigma \) with the total value of agents at least \( 3|V| + |A| - 1 \) in \( I \). Based on this assumption, we show a Hamiltonian path starting from \( s \) in \( D \). We let a sequence of leaf agents \( \pi = (a_{x_1}, a_{x_2}, \ldots, a_{x_{|V|-1}}) \) to denote the sequence of swaps from the initial endowment to the final assignment \( \sigma \), where we assume that \( x_i \neq x_{i+1} \) for all \( i \) since two continuous swaps happened between the same pair of agents mean doing nothing and they can be deleted. We will show that \( \pi \) is a sequence of alternating elements between vertices and arcs in \( D \).

By (P3), we know that \( x_1 = s \). We consider two cases: all vertex agents in \( \pi \) are different or not. We first consider the case that all vertex agents in \( \pi \) are different. For this case, it is easy to see that all arc agents in \( \pi \) are also different by the value functions of the agents. Since all agents in \( \pi \) are different, by (P8) and (P9), we know that \( \pi \) is a directed path starting from \( s \) in the directed graph \( D \).
We also have the following claims by the fact that each vertex and arc agent appears at most once in \( \pi \). For each swap in \( \pi \), if it is a trade between \( a_e \) and a vertex agent \( a_v \), then the object hold by agent \( a_v \) changes from the initial object \( o_e \) to an arc object \( o_e \), where agent \( a_e \) must value object \( o_e \) as 2. Thus, the value of agent \( a_v \) increases by 1. For each swap in \( \pi \), if it is a trade between \( a_e \) and an arc agent \( a_c \), then the object hold by agent \( a_c \) changes from the initial object \( o_e \) to a vertex object \( o_v \), where agent \( a_e \) must value object \( o_v \) as 2. Thus, the value of agent \( a_c \) increases by 1. For any case, each swap will increase the total value by exactly 1. By (P1), the total social welfare of the initial endowment is \( \pi \). So there are exactly \( 3|V| + |A| - 1 - (|V| + |A|) = 2|V| - 1 \) swaps in \( \pi \). The length of \( \pi \) is \( 2|V| - 1 \) and then \( \pi \) contains \(|V|\) vertices and \(|V| - 1\) arcs, where all vertices are different. This can only be a Hamiltonian path.

Next, we consider the case that some vertex agents appear at least twice in \( \pi \). Let \( a_v \) be the first vertex agent that appears at least twice in \( \pi \). Let \( \pi' \) be the subsequence of \( \pi \) from the beginning agent \( a_s \) to the second \( a_v \). Thus, each agent in \( \pi' \) except the last one appears at most once in \( \pi' \). By (P8) and (P9), we know that \( \pi' \) is corresponding to a directed path starting from \( s \) in \( D \), where only the last vertex \( v \) in the path appears twice. By (P2), we know that \( v \) cannot be \( s \). Thus, there is an arc agent \( a_{c_1} \) before the first appearance of \( a_v \) in \( \pi' \). Let \( a_{c_2} \) be the agent before the second appearance of \( a_v \) in \( \pi' \), i.e., the last but one agent in \( \pi' \). We can see that before the second swap between \( a_e \) and \( a_v \), center agent \( a_c \) holds object \( o_{e_1} \) and vertex agent \( a_c \) holds object \( o_{e_1} \), where both of \( e_1 \) and \( e_2 \) are arcs with the ending point being \( v \). After this swap, center agent \( a_c \) will hold object \( o_{e_1} \) and vertex agent \( a_c \) will hold object \( o_{e_2} \). From then on, only vertex agent \( a_v \) can make a trade with center agent \( a_c \). We have assumed that there are no two continuous swaps happened between the same pair of agents. Thus the second \( a_v \) must be the last agent in \( \pi \). Furthermore, in the last swap between \( a_e \) and \( a_v \), the value of \( a_v \) will not increase. By the above analysis for the first case, we know that \( \pi'' = (a_{c_1}, a_{c_2}, \ldots, a_{c_{2n}}) \) is corresponding to a simple direct path, where \( a_{c_{2n}} \) is a vertex agent. In \( \pi \), the total value of the agents can increase by at most 1 during the last but one swap between \( a_e \) and \( a_{c_{2n-1}} \), and can increase by 0 during the last swap between \( a_e \) and \( a_{c_{2n}} \). There is an odd number of agents in \( \pi'' \) since vertex agents and arc agents appear alternately in it and both of the first and the last agents are vertex agents. Each swap in \( \pi'' \) can increase the total value by 1. Since the total value of the last assignment is at least \( 3|V| + |A| - 1 \) and the initial value is \(|V| + |A| \), we know that the length of \( \pi'' \) is at least \( 2|V| - 1 \), which implies that \( \pi'' \) is corresponding to a Hamiltonian path.

Lemma 15 implies the correctness of Theorem 6.

6 Conclusion

In this paper, we investigate some problems about how to obtain certain assignments from an initial endowment via a sequence of rational trades between two agents. Two different preferences of the agents are considered: ties are allowed (weak preferences) or not (strict preferences). The computational complexity of the problems under different types of preferences will be different. For strict preferences, it is known that OBJECT REACHABILITY can be solved in polynomial
Whether **Strict Object Reachability** in a path is NP-hard or not was left as an interesting open problem and we design a polynomial-time for it in this paper. For **Strict Pareto Efficiency**, polynomial-time algorithms for the problem in a path or a star are known. However, the computational complexity of it in a general tree is unknown. Since strict preferences are contained in weak preferences, we know that hardness results for strict version problems imply the same hardness results for the corresponding weak version problems. So we know that **Weak Pareto Efficiency** and **Weak Object Reachability** in general graphs are NP-hard by previous results in [17]. In this paper, we further show that **Weak Object Reachability** in a star is polynomially solvable, and **Weak Object Reachability** in a path and **Weak Pareto Efficiency** in a path are NP-hard. For **Weak Pareto Efficiency** in a star, we do not prove or disprove its NP-hardness. However, we show that to find a reachable assignment maximizing total social welfare is NP-hard even the network is a star or a path.

For further study, we think it is interesting to consider the two unsolved problems in Table 1 and also consider trades among three or more agents.

**Acknowledgement**

The work is supported by the National Natural Science Foundation of China, under grants 61772115 and 61370071.

**References**

1. Peter Gardenfors. Assignment problem based on ordinal preferences. *Management Science*, 20(3):331–340, 1973.
2. Leslie Blackett Wilson. Assignment using choice lists. *Journal of the Operational Research Society*, 28(3):560–578, 1977.
3. Julien Lesca, Michel Minoux, and Patrice Perny. The fair owa one-to-one assignment problem: Np-hardness and polynomial time special cases. *Algorithmica*, 81(1):98–123, Jan 2019.
4. Atila Abdulkadiroğlu and Tayfun Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–701, 1998.
5. David F Manlove. *Algorithmics of matching under preferences*, volume 2. World Scientific, 2013.
6. Lloyd Shapley and Herbert Scarf. On cores and indivisibility. *Journal of mathematical economics*, 1(1):23–37, 1974.
7. Atila Abdulkadiroğlu and Tayfun Sönmez. House allocation with existing tenants. *Journal of Economic Theory*, 88(2):233–260, 1999.
8. Alvin E Roth, Tayfun Sönmez, and M Utku Ünver. Kidney exchange. *The Quarterly Journal of Economics*, 119(2):457–488, 2004.
9. Jorge Alcalde-Unzu and Elena Molis. Exchange of indivisible goods and indifferences: The top trading absorbing sets mechanisms. *Games and Economic Behavior*, 73(1):1–16, 2011.
10. Paula Jaramillo and Vikram Manjunath. The difference indifference makes in strategy-proof allocation of objects. *Journal of Economic Theory*, 147(5):1913–1946, 2012.

11. Haris Aziz and Bart De Keijzer. Housing markets with indifferences: a tale of two mechanisms. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1249–1255, 2012.

12. Daniela Saban and Jay Sethuraman. House allocation with indifferences: a generalization and a unified view. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 803–820, 2013.

13. Lars Ehlers. Top trading with fixed tie-breaking in markets with indivisible goods. *Journal of Economic Theory*, 151:64–87, 2014.

14. Akihisa Sonoda, Etsushi Fujita, Taiki Todo, and Makoto Yokoo. Two case studies for trading multiple indivisible goods with indifferences. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 791–797, 2014.

15. Ghufran Ahmad. *Essays on Housing Market Problem*. PhD thesis, Texas A & M University, 2017.

16. Anastasia Damamme, Aurélie Beynier, Yann Chevaleyre, and Nicolas Maudet. The power of swap deals in distributed resource allocation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 625–633, 2015.

17. Laurent Gourves, Julien Lesca, and Anaëlle Wilczynski. Object allocation via swaps along a social network. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 213–219, 2017.

18. Rediet Abebe, Jon Kleinberg, and David C Parkes. Fair division via social comparison. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 281–289, 2017.

19. Xiaohui Bei, Youming Qiao, and Shengyu Zhang. Networked fairness in cake cutting. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3632–3638, 2017.

20. Robert Bredereck, Andrzej Kaczmarczyk, and Rolf Niedermeier. Envy-free allocations respecting social networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 283–291, 2018.

21. Aurélie Beynier, Yann Chevaleyre, Laurent Gourvès, Julien Lesca, Nicolas Maudet, and Anaëlle Wilczynski. Local envy-freeness in house allocation problems. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 292–300, 2018.

22. Sen Huang and Mingyu Xiao. Object reachability via swaps along a line. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, 2019.

23. Bengt Aspvall, Michael F Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

24. Ryo Yoshinaka. Higher-order matching in the linear lambda calculus in the absence of constants is np-complete. In *International Conference on Rewriting Techniques and Applications*, pages 235–249, 2005.

25. M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0-7167-1044-7.