UTMD Working Paper
The University of Tokyo
Market Design Center

UTMD-030

Anytime Capacity Expansion in Medical Residency Match by Monte Carlo Tree Search

Kenshi Abe
CyberAgent, Inc.

Junpei Komiyama
New York University

Atsushi Iwasaki
University of Electro-Communications

May 22, 2022

Anytime Capacity Expansion in Medical Residency Match by Monte Carlo Tree Search*

Kenshi Abe[1] , **Junpei Komiyama**[2] , **Atsushi Iwasaki**[3]
[1]CyberAgent, Inc., [2]New York University, [3]University of Electro-Communications
abe_kenshi@cyberagent.co.jp, junpei@komiyama.info, atsushi.iwasaki@uec.ac.jp

Abstract

This paper considers the capacity expansion problem in two-sided matchings, where the policymaker is allowed to allocate some extra seats as well as the standard seats. In medical residency match, each hospital accepts a limited number of doctors. Such capacity constraints are typically given in advance. However, such exogenous constraints can compromise the welfare of the doctors; some popular hospitals inevitably dismiss some of their favorite doctors. Meanwhile, it is often the case that the hospitals are also benefited to accept a few extra doctors. To tackle the problem, we propose an anytime method that the upper confidence tree searches the space of capacity expansions, each of which has a resident-optimal stable assignment that the deferred acceptance method finds. Constructing a good search tree representation significantly boosts the performance of the proposed method. Our simulation shows that the proposed method identifies an almost optimal capacity expansion with a significantly smaller computational budget than exact methods based on mixed-integer programming.

1 Introduction

This paper considers the capacity expansion problem in two-sided matchings, where the policymaker is allowed to allocate some extra seats as well as the standard seats. The two-sided matchings have a lot of real applications such as medical residency match [Roth, 1991] and school choice [Abdulkadiroğlu and Sönmez, 2003]. The theory has been extensively developed across computer science and economics [Roth and Sotomayor, 1990; Manlove, 2013]. To establish our model and concepts, we use medical residency match as a running example. In this literature, each hospital (school) accepts a limited number of residents (students). Such capacity constraints (or quotas) are assumed to be known and fixed in advance.

However, in practice, even for hospitals, or its stakeholders, it is often uncertain how the capacity constraints are specified beforehand. Those could be flexible and variable more than the previous studies preclude. By pooling some extra funding, hospitals can accept a few more residents. Schools may be granted some budgets or dispatched teachers from their district according to the demand.

Such a situation involves a lot of resource allocation problems where capacities are predefined, e.g., how many jobs each machine takes in the jobshop scheduling problem. How we should choose the capacities has been paid less attention than it deserves, though they influence the performance generated by allocations.

A classical comparative statistic already analyzes the effect of such expansion. If the capacity of some hospitals is expanded, the welfare of every resident weakly improves [Roth and Sotomayor, 1990]. However, it is rarely investigated how capacities should be expanded to improve the total welfare. Bobbio *et al.* [2021] have initiated the question of how limited extra seats should be allocated, keeping resulting matchings *stable*. The stability is a key concept for the two-sided matching and if a matching is stable, no groups of residents and hospitals have profitable deviations. The celebrated deferred acceptance (DA) algorithm is known to find a stable matching [Gale and Shapley, 1962]. The capacity expansion with DA, i.e., finding the optimal allocation of extra seats in the welfare of residents, is formulated as an integer quadratic programming [Bobbio *et al.*, 2021], which is computationally challenging to solve exactly. In addition to the exact method, they also developed some greedy heuristics, which run very effectively yet are suboptimal.

This paper proposes an alternative method to solve the capacity expansion problem where the upper confidence tree (UCT) searches the space of capacity expansions.[1] Each pattern of them has a resident-optimal stable assignment that DA finds. Not only does UCT obtain an optimal solution given a sufficiently large time, but also a policymaker can stop UCT anytime to obtain a reasonably good solution, which is important given the hardness of obtaining an optimal solution. We then characterize a good tree representation so that the tree search method efficiently exploits the structure of the capacity expansion problem. There have been a certain amount of studies on two-sided matchings in the AI community, although this literature has been established

*Atsushi Iwasaki was supported by JSPS KAKENHI Grant Numbers JP21H04890 and JP20K20752.

[1]An implementation of our method is available at https://github.com/CyberAgentAILab/uct_capacity_expansion.

mainly in fields across algorithms and economics [Kamada and Kojima, 2015; Biró *et al.*, 2010; Fragiadakis *et al.*, 2016; Goto *et al.*, 2014; Goto *et al.*, 2016]. In many application domains, various distributional constraints are imposed on an outcome, e.g., regional maximum quotas are imposed on hospitals in urban areas to allocate more doctors to rural areas [Kamada and Kojima, 2015].

2 Problem Setup

Let $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ be the set of residents (doctors) and $\mathcal{H} = \{h_1, h_2, \ldots, h_H\}$ be the set of hospitals. We consider a bipartite graph where $\mathcal{D}$ and $\mathcal{H}$ are connected by edges $\mathcal{E}$. Each edge represents that the resident is accepted by the hospital. Residents and hospitals have their own preferences. We denote $h \succ_d h'$ if resident d prefers hospital h to hospital h'. Similarly, we denote $d \succ_h d'$ if hospital h prefers resident d over resident d'. A vector $\boldsymbol{q} = (q_1, q_2, \ldots, q_H)$ represents the quota of the hospitals. An instance of the residency match problem is a tuple $\Gamma = (\mathcal{D}, \mathcal{H}, \succ, \boldsymbol{q})$, where $\succ$ denotes all preferences of the residents and the hospitals.

A matching $M \in \mathcal{M}$ is a subset of $\mathcal{E}$ such that the following criteria are met. Each resident $d \in \mathcal{D}$ has at most one edge. Each hospital $h \in \mathcal{H}$ can have at most q_h edges. We say that resident d is unassigned if there is no edge from d. We denote $M(d)$ and $M(h)$ to represent the hospital assigned to resident d and the set of the residents assigned to hospital h, respectively. Given a matching M, we say a pair $(d, h) \in \mathcal{E}$ is a *blocking pair* if (1) resident d is unassigned or prefers hospital h to $M(d)$, and (2) hospital h is such that $|M(h)| < q_h$ or prefers resident d over at least one resident in $M(h)$. Matching M is stable if there is no blocking pair.

Gale and Shapley [1962] showed that a stable matching always exists and proposed the deferred-acceptance (DA) algorithm that yields a stable matching. The resident-proposing DA finds the resident-optimal stable matching where no resident is better off among all stable matchings [Roth and Sotomayor, 1990]. That matching is also obtained by minimizing the total resident ranking under the stability constraints. A resident ranking is denoted by $\mathrm{rank}_d(h)$ the rank of hospital h according to the resident d's preference. This is also referred to as the *cost* of the match (d, h). Given a matching M, the total resident ranking is defined as

$$\sum_{(d,h)\in M} \mathrm{rank}_d(h). \tag{1}$$

DA gives a matching that minimizes Eq. (1) among all stable matchings [Bobbio *et al.*, 2021].

This paper improves the resident utilities by allowing additional seats. Letting $\boldsymbol{t} \in \mathbb{N}^{\mathcal{H}}$ be a non-negative vector, we consider an expanded matching problem in which the capacity of each hospital h is the sum of regular capacity q_h and the extended capacity t_h. A reasonable extension of the capacity should not be very large and can accommodate the demand of each hospital. Given this, we consider the following optimization problem: Let

$$\mathcal{P} = \left\{ (\boldsymbol{x}, \boldsymbol{t}) \in \{0,1\}^{\mathcal{E}} \times \Theta \;\middle|\; \sum_{h\in\mathcal{H}} x_{dh} \le 1 \;\forall d \in \mathcal{D}, \right.$$
$$\left. \sum_{d\in\mathcal{D}} x_{dh} \le (q_h + t_h) \;\forall h \in \mathcal{H} \right\}, \text{and}$$
$$\Theta = \left\{ \boldsymbol{t} \in \{0,1,2,\ldots,B\}^{\mathcal{H}} \;\middle|\; t_h \le b_h \;\forall h, \sum_{h\in\mathcal{H}} t_h \le B \right\}$$

be the set of matchings. The capacity expansion problem is the following:

$$\min_{\boldsymbol{x},\boldsymbol{t}} \sum_{(d,h)\in\mathcal{E}} \mathrm{rank}_d(h) x_{dh} \tag{2}$$
$$\text{s.t. } (q_h + t_h)\left(1 - \sum_{h'\in S_{dh}} x_{dh'}\right) \le \sum_{d'\in T_{dh}} x_{d'h} \;\forall (d,h) \in \mathcal{E},$$
$$(\boldsymbol{x}, \boldsymbol{t}) \in \mathcal{P}.$$

where $S_{dh} = \{h' \in \mathcal{H} : \mathrm{rank}_d(h') \le \mathrm{rank}_d(h)\}$ is the set of indices of hopsitals that resident d prefers at least as hospital h; similarly, $T_{dh} = \{d' \in \mathcal{D} : \mathrm{rank}_h(d') < \mathrm{rank}_h(d)\}$ is the set of indices of residents that hospital h prefers more than resident d. Namely, keeping stability, we maximize the residents' welfare subject to the capacity constraints of the hospitals, where the capacity can be relaxed up to B seats. Moreover, we assume that each hospital h has its own expansion limit b_h.

Remark 1. (Complete preference) The optimization of Eq. (2) requires a complete preference of all residents and hospitals, which is often impractical. For example, in the real data of a residency matching program, an average resident only applies to four or five hospitals. In this case, we can add a dummy hospital of infinite capacity and rank all unpreferred hospitals after the dummy hospital.

Bobbio *et al.* [2021] proved the computational hardness of the problem of Eq. (2) without hospital-wise limit b_h.

Proposition 1. (Hardness result [Bobbio *et al.*, 2021]) The decision version of Eq. (2) is NP-complete even in the case where there is no hospital-wise limit b_h.

The key idea of our algorithm for this problem is that, when we fix an expansion vector $\boldsymbol{t} \in \Theta$, the capacity expansion problem of Eq. (2) boils down to the standard matching instance $\Gamma = (\mathcal{D}, \mathcal{H}, \succ, \boldsymbol{q} + \boldsymbol{t})$ that we can solve efficiently by DA.[2] Therefore, a tree search on the space of Θ combined with DA is expected to give an efficient solution.

3 Method

This section proposes our algorithm to solve Eq. (2). We apply Upper Confidence Tree (UCT) [Kocsis and Szepesvári, 2006], which belongs to a class of Monte Carlo tree search methods [Browne *et al.*, 2012]. One of the most successful examples of UCT is for abstract games such as the Game of Go [Wang and Gelly, 2007; Yoshizoe *et al.*, 2011; Silver *et al.*, 2016] where the goal is to find the best next move in a game tree. Another aspect of UCT is diverse recommendation method, which minimizes total regret, searches

[2]The computational complexity of DA is $O((D+H)^2)$.

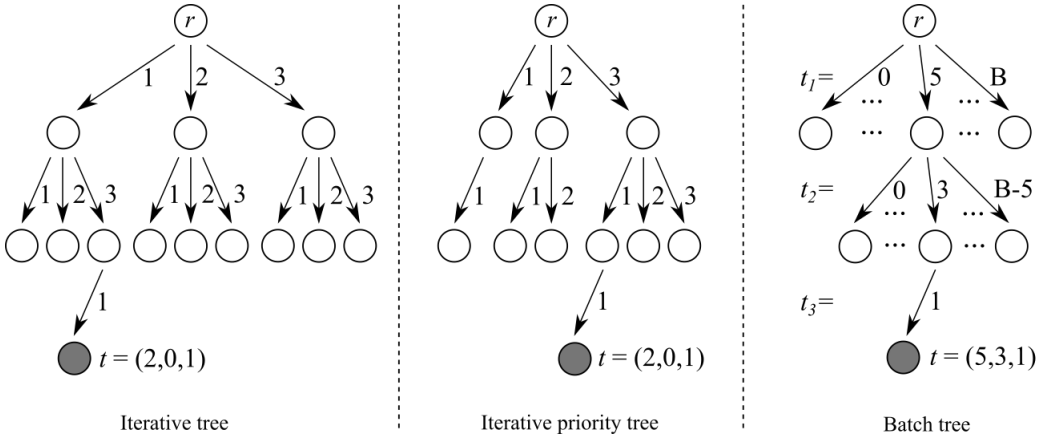

Figure 1: Three tree representations. In the iterative tree (left), the node of the grey dot corresponds to a vector $\boldsymbol{t} = (2, 0, 1)$, with its corresponding path expanding hospitals 1 and 3 once, and expanding hospital 1 once again. An edge in the iterative priority tree (middle) is the same as the iterative tree, but its path only includes nonincreasing order of edges (e.g., $1 \to 3 \to 1$ is not allowed). An edge in the batch tree (right) corresponds to how many extended seats we allocate to each hospital ($= t_1, t_2, t_3$). The node of the grey dot in the batch tree corresponds to a vector of $\boldsymbol{t} = (5, 3, 1)$. Ordering of the hospitals matters in the iterative priority and the batch trees; the hospitals that receive large expansion should be numbered earliest. All leaves of the iterative and iterative priority trees are at depth B, whereas the path length to a leaf of the batch tree varies: It stops branching when $\sum_h t_h = B$.

for different solutions of reasonable objective value [Bosc *et al.*, 2018]. In this paper, we use UCT to find a global optimum node in a tree $\mathcal{T}_{\text{all}}$.

UCT consists of a sequential process of traversing a tree structure where each tree node i is associated with its value v_i. We use $n \in \{1, 2, \ldots, N\}$ to denote the number of rounds. The entire tree $\mathcal{T}_{\text{all}}$ is typically very large, and UCT tries to develop a subtree of it. We denote the subtree of round n by $\mathcal{T}(n)$. Each node i in the current tree is associated with a tuple $(V_i, N_i) \in \mathbb{R}^2$, where V_i is the sum of rewards and N_i the number of times at which node i is traversed. The value $\hat{\mu}_i := V_i/N_i$ is an estimator of v_i, and its standard deviation is proportional to $\sqrt{1/N_i}$. Each round consists of *selection*, *development*, *simulation*, and *backpropagation* steps.

Selection: At each iteration, UCT traverses the tree from the root. At each node i, it chooses a child node c with the maximum UCB value:

$$\text{UCB}(c) := \hat{\mu}_c + C_p \sqrt{\frac{\log(N_i)}{N_c}},$$

where $C_p > 0$ is the parameter that determines how much exploration it attempts.

Development: When it reaches a node k that is out of the current tree, it adds the node to the current tree.

Simulation: Based on the reached node k, it conducts a random play to find l, which is a leaf that stems from k.

Backpropagation: Letting v_l be the value of the leaf l, it updates the statistics of all nodes that we have traversed in this round: $V_i \leftarrow V_i + v_l, N_i \leftarrow N_i + 1$ for each node i during the backpropagation.

The four steps above are quite standard in UCT. For a more algorithmic description, see Algorithm 1 in the appendix.

3.1 Tree representation of expansion space

Unlike abstract games where the game tree is inherent, the relation between a tree and the capacity expansion in our problem is nontrivial. This section describes the mapping from a tree node into an expansion $\boldsymbol{t} \in \Theta$.

In the capacity expansion problem defined in Eq. (2), DA yields an optimal solution given a fixed expansion vector $\boldsymbol{t}$. Therefore, we consider a tree where each node corresponds to an expansion vector $\boldsymbol{t}$, and each leaf is an expansion of limit $\sum_{h \in \mathcal{H}} t_h = B$. The value v_i at each node i is the objective value of DA for the corresponding expansion.

We consider the following criteria are important in constructing a good tree representation of an expansion.

1. Faithfulness: The tree preserves the inclusion relationship of the original expansion vector space: if node j is a descendant of node i, then $\boldsymbol{t}(j) - \boldsymbol{t}(i) \geq 0$, where $\boldsymbol{t}(i), \boldsymbol{t}(j)$ be corresponding expansions of nodes i, j and $\boldsymbol{t}(j) - \boldsymbol{t}(i) \geq 0$ means all the features of the vector $\boldsymbol{t}(j) - \boldsymbol{t}(i)$ are non-negative.
2. Nonredundancy: The mapping is one-to-one. Namely, for any expansion vector $\boldsymbol{t} \in \Theta$ such that $\sum_h t_h = B$, there exists only one leaf of tree $\mathcal{T}_{\text{all}}$. Redundancy in a tree representation compromises the search efficiency.
3. Decisiveness: Assuming that a tree representation is faithful, the tree representation is decisive if branching in a shallow layer is more informative than the one in a deep layer. In our case, allocations related to important hospitals should appear in a shallow layer of the tree.

Table 1: Comparison of the three tree representations with respect to the criteria.

	Faithfulness	Nonredundancy	Decisiveness
Iterative tree	✓		
Iterative priority tree (IPT)	✓	✓	
Batch tree (BT)	✓	✓	✓

Considering those criteria, we propose the three tree representations in Figure 1. In all of the three representations, the root node r corresponds to the zero vector $\boldsymbol{t} = (0, 0, \ldots, 0)$ that corresponds to no expansion. Table 1 illustrates which properties are satisfied in each of the representations. We will discuss the idea behind them.

The iterative representation, which is the most straightforward, builds an H-ary tree.[3] Each edge corresponds to allocating a seat to one of the hospitals. As a result, each path from the root to depth B represents an expansion of size B. Although the iterative representation is faithful, it is redundant. For example, consider the case of three hospitals. Let $1 \to 2 \to 2$ to denote a path on the tree that sequentially expands hospitals $1, 2, 2$. An expansion vector $(2, 1, 0)$ corresponds to three different path on the tree ($1 \to 1 \to 2$, $1 \to 2 \to 1$, and $2 \to 1 \to 1$). As a result, UCT with iterative tree representation searches an unnecessary large representation space, which increases the computational burden.

To deal with this issue, we introduce an improved tree representation of the expansion, which we call the iterative priority tree (IPT). For each node i, it only allocates a node with its priority higher than the most prioritized hospital that has been allocated a seat. IPT solves the issue of redunduncy in the iterative tree:

Proposition 2. (Nonredundancy of IPT) For each expansion vector $\boldsymbol{t} \in \mathbb{N}^{\mathcal{H}}$, there exists a unique node in IPT.

Proof. Each expansion uniquely corresponds to a nonincreasing sequence of numbers. For example, expansion $\boldsymbol{t} = (1, 3, 1)$ corresponds to $3 \to 2 \to 2 \to 2 \to 1$. It is easy to confirm that, in the iterative priority tree, this is the unique path that leads to the desired expansion. □

While the IPT is nonredundant, there still remains some space for improvement. The batch tree (BT) representation fully exploits the priority information to allocate more than one seats to popular nodes, which reduces the depth of the optimal solution in the tree. Namely, each depth of BT corresponds to how many seats to allocate to each hospital. The following proposition states that the properties of BT:

Proposition 3. (Nonredundancy and decisiveness of BT) BT is nonredundant. It is also decisive, that is, if the optimal solution aligns with the expansion vector (i.e., $t_1 \geq t_2 \geq t_3 \geq \cdots \geq t_H$), then BT minimizes the depth of the optimal node among all trees such that each edge allocates seats to one hospital.

Proof. The nonredundancy is trivial. In the following, we derive the decisiveness of the BT. Assume that the optimal expansion vector is such that $t_1 \geq t_2 \geq t_3 \geq \cdots \geq t_H$, and let h be the first index such that $t_h = 0$. BT includes the path of depth $h - 1$ that leads to this allocation. □

3.2 Ordering hospitals

We introduce two ideas for ordering hospitals that we use in IPT and BT. The first idea is to order hospitals in terms of their popularity

$$\text{Popularity}(h) := \sum_{d \in \mathcal{D}} \text{rank}_d(h).$$

We use the term "popularity" because it is the total rank of hospital h in view of residents. Note that this value is also referred to as the Borda count in the context of social choice [Moulin, 1994]. A smaller value represents a more popular hospital.

The second idea is utilizing *potential envies* that may not be justified so that we could construct a better decision tree of the hospitals. Namely, let

$$\text{Envy}(h) := \sum_{d \in \mathcal{D}} \mathbf{1}[\text{rank}_d(h) < \text{rank}_d(M(d))]$$

be the potential envy that the residents have toward the ones matched to hospital h in matching M. The score indicates the number of residents who prefer hospital h over their matched hospital in M. For calculating M, we run DA with no expansion before running UCT to determine the ordering.

Avoiding multiple entries. Since the evaluation of each node (i.e., DA with a given expansion vector) is deterministic, we can restrict the node selection so that a node is never evaluated twice. In the backpropagation step, we mark the node k as "evaluated" if it is a leaf of $\mathcal{T}_{\text{all}}$. Moreover, for each ascendant node, if all children of the node are evaluated, then we mark that node as evaluated. In the selection and simulation phase, the evaluated nodes are never visited again.

3.3 Consistency of the method

This section analyzes the proposed method. The first result is a trivial consequence of avoiding multiple entries:

Proposition 4. (Worst-case sample complexity) UCT with iterative tree finds an optimal solution in $N = |\mathcal{T}_{\text{all}}|$ rounds.

Proof. Since the development step adds a node to $\mathcal{T}(n)$ at each round n, it follows that it obtains the optimal solution if N reaches the number of the nodes $|\mathcal{T}_{\text{all}}|$. □

We further propose a nontrivial analysis by following the analysis of the original UCT by [Kocsis and Szepesvári, 2006]. The largest difference is that Kocsis and Szepesvári [2006] analyzed the average quality of the selected action (i.e., regret), whereas we are interested in the quality of the best path, which corresponds to the optimal solution of the capacity expansion problem.

Let $\hat{\mu}_{i,m} = V_i/m$ be the mean of v_i over the first m visits, and $U_{i,m,n} = \hat{\mu}_{i,m} + C_p\sqrt{\frac{\log n}{m}}$ be the corresponding UCB value. Let W be the maximum number of the children of a node and D_T be the depth of the tree.[4] Let v_i^* be the maximum value of the leaves that are descendants of node i. Let $v^* = v_r^*$ be the global optimal value, where r is the root node. The following assumption is essentially the same as the one in [Kocsis and Szepesvári, 2006]:

[3] Note that H is the number of the hospitals.

[4] That is $W = H, D_T = B$ for iterative and iterative priority tree, or $W = B, D_T = H$ for the batch tree.

Table 2: Average percentage gaps between the solution found by the Agg-Lin and the solution found by each method for Set 1 experiments. UCT with iterative priority trees has three different orderings, i.e., random, popularity, and envy-based, each of which is denoted by IPT-R, IPT-P, and IPT-E. As well, UCT with batch trees has the three variants: BT-R, BT-P, and BT-E. "0.0" indicates the value is equal to Agg-Lin. Negative values mean the temporal value of Agg-Lin is outperformed.

H	B	α	Baseline		**UCT (proposed)**						
			LPH	Grdy	Iterative	IPT-R	IPT-P	IPT-E	BT-R	BT-P	BT-E
5	5	0.0	7.5	5.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	30	0.0	6.3	32.9	1.3	34.5	0.0	0.0	0.4	0.0	0.0
15	5	0.0	8.9	4.6	0.5	0.5	0.05	0.09	1.1	1.1	1.1
15	30	0.0	23.2	25.3	17.8	18.0	15.5	10.4	19.9	15.4	6.9
5	5	0.2	1.8	1.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	30	0.2	3.6	6.4	1.6	0.9	0.1	0.03	0.4	0.09	0.1
15	5	0.2	2.6	0.8	0.09	0.0	0.0	0.0	0.2	0.06	0.06
15	30	0.2	4.1	4.3	2.7	2.5	1.3	1.4	1.4	0.3	0.3
5	5	0.4	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	30	0.4	1.1	2.1	1.4	0.1	0.0	0.0	0.1	0.0	0.0
15	5	0.4	1.1	0.3	−0.08	−0.09	−0.1	−0.1	0.02	−0.09	−0.09
15	30	0.4	0.8	0.8	0.5	0.05	−0.6	−0.6	0.04	−0.8	−0.8

Table 3: Average run times for Set 1 (seconds).

H	B	α	Baseline			**UCT (proposed)**						
			Agg-Lin	LPH	Grdy	Iterative	IPT-R	IPT-P	IPT-E	BT-R	BT-P	BT-E
5	5	0.0	24.01	0.01	0.06	10.09	0.64	0.64	0.65	0.86	0.86	0.86
5	30	0.0	11.85	0.01	0.19	35.60	45.38	39.84	39.92	36.93	38.83	38.03
15	5	0.0	854.91	0.02	0.94	52.11	56.98	54.88	54.31	54.76	53.50	53.37
15	30	0.0	362.87	0.02	3.34	177.00	167.24	166.36	160.77	149.91	139.90	139.74
5	5	0.2	37.79	0.01	0.09	15.59	0.98	0.99	0.99	1.30	1.31	1.32
5	30	0.2	55.79	0.01	0.44	89.59	94.44	93.57	94.46	91.78	90.11	90.06
15	5	0.2	2740.02	0.03	1.80	129.87	129.35	129.18	130.38	125.60	128.63	128.88
15	30	0.2	3527.88	0.03	7.14	384.11	376.15	370.17	370.66	343.04	362.58	361.80
5	5	0.4	86.30	0.01	0.09	15.04	0.98	0.97	0.97	1.30	1.29	1.30
5	30	0.4	294.07	0.01	0.54	123.79	130.87	129.49	129.18	123.63	123.91	123.89
15	5	0.4	3600.21	0.03	2.38	169.40	166.00	166.96	168.14	168.51	166.51	166.11
15	30	0.4	3600.41	0.04	9.24	578.57	606.01	574.18	584.77	553.69	571.35	572.87

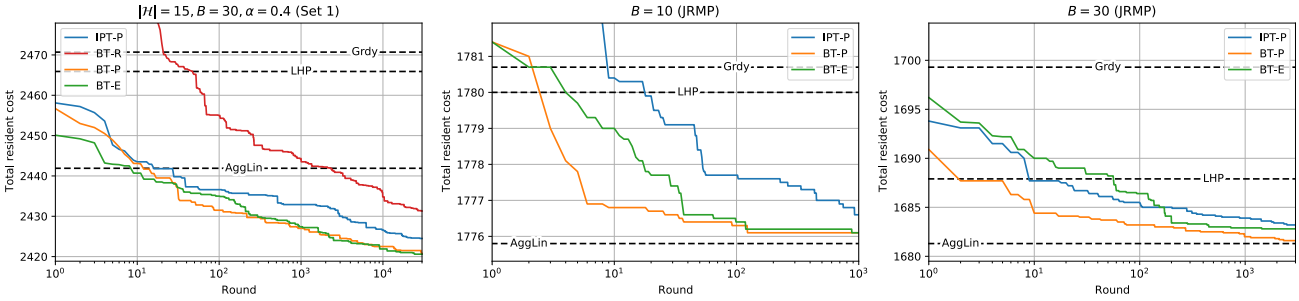

Figure 2: Total resident rankings (costs) of the proposed algorithms. Horizontal axis indicates the number of rounds. The dotted lines represent the total costs obtained by Grdy, LPH, and Agg-Lin.

Assumption 1. (Confidence bound) We assume

$$\mathbb{P}\left[U_{i,n,m} \geq v_i^*\right] \geq 1 - 1/n^2.$$

Definition 1. (Δ-optimal tree) Let $\Delta > 0$ be arbitrary. The Δ-optimal subtree is defined recursively as follows. First, it includes the root of the original tree. For each node i in the subtree, add each children c such that $v_i^* - v_c^* \leq \Delta$. In other words, Δ-optimal subtree is a subtree where each edge is suboptimal at most Δ. Let $S(\Delta)$ be the number of the nodes in the Δ-optimal subtree.

Theorem 5. If Assumption 1 holds for any node i, then with probability at least $1 - o(N^{-1})$, UCT finds at least one node in the Δ-suboptimal tree if N satisfies

$$N > WS(\Delta)\frac{(C_p)^2 \log N}{\Delta^2}.$$

The proof of Theorem 5 is in Appendix C.

Remark 2. (Implication) The solution of the node i in Δ-

Table 4: Average percentage gaps between the solution found by the Agg-Lin and the solution found by each method for JRMP.

B	Baseline		**UCT (proposed)**						
	LPH	Grdy	Iterative	IPT-R	IPT-P	IPT-E	BT-R	BT-P	BT-E
10	0.2	0.3	1.3	1.1	0.04	0.06	0.9	0.02	0.02
30	0.4	1.1	4.8	3.7	0.1	0.4	1.7	0.02	0.09

Table 5: Average run times for JRMP (seconds).

B	Baseline			**UCT (proposed)**						
	Agg-Lin	LPH	Grdy	Iterative	IPT-R	IPT-P	IPT-E	BT-R	BT-P	BT-E
10	247.89	0.08	9.35	20.42	22.11	21.49	22.08	20.58	21.13	21.16
30	438.20	0.08	27.12	62.72	74.68	73.18	73.65	77.30	74.98	78.20

suboptimal tree is at least

$$v_i \geq v^* - D_T \Delta.$$

Therefore, letting Δ be sufficiently small and N be sufficiently large, UCT finds an almost optimal solution. The required number of N depends on $S(\Delta)$, which corresponds to the number of close-to-optimal children.

Although our analysis gives a complexity bound based on the structure of the tree, we still think this analysis (as well as any existing analysis of UCT) is not very satisfying. We discuss the limitation of this analysis in Appendix D.

4 Evaluation

This section empirically evaluates our algorithm via synthetic and real datasets. We compare our algorithms with the ones by [Bobbio *et al.*, 2021]: the greedy algorithm (Grdy), the linear programming-based heuristic (LPH), and the aggregated linearization (Agg-Lin).

Let us enumerate existing algorithms: First, Grdy is an algorithm that allocates B expansion seats iteratively such that each allocation maximizes the marginal cost reduction. Second, LPH first computes a minimum cost matching without stability constraints via minimum-cost flow to fix the expansion seats, and then uses DA to obtain a stable matching. Finally, Agg-Lin is an exact method that uses McCormick envelopes and solves a mixed integer programming, which is computationally intensive. [5] We warm-up Agg-Lin as suggested by [Bobbio *et al.*, 2021].

For our UCT method, we set $N = B \times 10^3$ in synthetic data experiments, $N = B \times 10^2$ in real data experiments. The value of C_p, which determines the tradeoff between exploration and exploitation, is set to be $\sqrt{0.002}$. We consider this parameter is robust enough to cover all settings. We implement our simulation with Python 3 and solve the mathematical programming using Gurobi, which is in favor of Agg-Lin. We restrict the runtime for each method to one hour.

Let us describe the procedure of generating the datasets. First, we build two synthetic datasets. **Set 1** involves B but each hospital does *not* have its expansion limit b_h. **Set 2** involves B as well as b_h for each hospital. Note that the setting of Set 1 is similar to the experimental section in [Bobbio *et al.*, 2021]. Regarding the preference among the residents, we follow the setting [Goto *et al.*, 2016], which involves a correlation parameter $\alpha \geq 0$. Larger value of α implies a stronger correlation among the preferences. Note that $\alpha = 0$ (no correlation) corresponds to the setting of [Bobbio *et al.*, 2021]. We set $D := |\mathcal{D}| = 1,000$, and conduct experiments varying the parameters $H := |\mathcal{H}|$, B, b_h, and α. For each combination of parameters, we average the results for 10 instances. The details of the data generation are in Appendix E.2. The limitation of this analysis in Appendix D.

Second, we generate the dataset, **JRMP**, based on Japan Residency Matching Program 2007 [Kamada and Kojima, 2015], which matches medical hospital students (residents) with residency training programs. We extracted $1,287$ residents in the Tokyo district who match with $50 + 1$ (1 for a dummy) hospitals with a resident-side preference. We set $B \in \{10, 30\}$ for the limit of capacity expansion. For each B, we average the results for 10 instances. We place the details of the dataset in Appendix E.3.

Tables 2 and 3 illustrate the quality of solutions and its runtime for Set 1, respectively. We place the results for Set 2 in Appendix E.4. Also, we place Tables 4 and 5 for the JRMP data. Note that Table 2 indicates the average percentage gap in the total resident ranking (TRR) defined in Eq. (1):

$$100 \times \frac{\text{(TRR of the method)} - \text{(TRR of Agg-Lin)}}{\text{(TRR of the method)}}.$$

Agg-Lin always outputs an optimal solution upon the completion. However, as in Table 3, it does not run within one hour for large instances with positive correlation α. In that case, we use the temporal value that Gurobi outputs, which can be suboptimal and the percentage gap can be negative.

Among the algorithms we consider, the two greedy methods (Grdy and LPH) run very fast (< 10 seconds) and output a suboptimal solution. Both of them are outperformed by all variants of ours. In particular, BT outperformed iterative tree and IPT, and using the popularity and envy-based orderings outperformed the random one.

Figure 2 describes the objective value as a function of rounds, which indicates an early termination of BT often has a satisfying solution. In summary, BT with envy/popularity-based ordering yields the best results among our algorithms. Our algorithms run 2–20 times faster than Agg-Lin, and the quality of the solution is close to optimal.

[5] See Appendix A for the comparison between general methods for solving mixed integer programming and our UCT.

5 Conclusion

This paper sheds light on the capacity expansion in the two-sided matching to consider a flexible allocation of extra seats within a given limit beforehand. To handle this NP-Complete problem, we develop a UCT-based search method and verify that it outperforms the previous approaches. Future works include (1) extending it to matchings with constraints that need not admit stable matching and (2) utilizing other tree search algorithms such as Nested Monte-Carlo Search [Cazenave, 2009; Rosin, 2011].

References

[Abdulkadiroğlu and Sönmez, 2003] Atila Abdulkadiroğlu and Tayfun Sönmez. School choice: A mechanism design approach. *American Economic Review*, 93(3):729–747, 2003.

[Biró *et al.*, 2010] Peter. Biró, Tamas. Fleiner, Robert.W. Irving, and David.F. Manlove. The college admissions problem with lower and common quotas. *Theoretical Computer Science*, 411(34-36):3136–3153, 2010.

[Bobbio *et al.*, 2021] Federico Bobbio, Margarida Carvalho, Andrea Lodi, and Alfredo Torrico. Capacity expansion in the college admission problem, 2021.

[Bosc *et al.*, 2018] Guillaume Bosc, Jean-François Boulicaut, Chedy Raïssi, and Mehdi Kaytoue. Anytime discovery of a diverse set of patterns with monte carlo tree search. *Data Min. Knowl. Discov.*, 32(3):604–650, 2018.

[Browne *et al.*, 2012] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[Cazenave, 2009] Tristan Cazenave. Nested monte-carlo search. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 456–461, 2009.

[Feldman and Domshlak, 2014] Zohar Feldman and Carmel Domshlak. Simple regret optimization in online planning for markov decision processes. *J. Artif. Int. Res.*, 51(1):165–205, sep 2014.

[Fragiadakis *et al.*, 2016] Daniel Fragiadakis, Atsushi Iwasaki, Peter Troyan, Suguru Ueda, and Makoto Yokoo. Strategyproof matching with minimum quotas. *ACM Transactions on Economics and Computation*, 4, 2016. (an extended abstract appeared in AAMAS, pages 1327–1328, 2012).

[Gale and Shapley, 1962] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[Goto *et al.*, 2014] Masahiro Goto, Naoyuki Hashimoto, Atsushi Iwasaki, Yujiro Kawasaki, Suguru Ueda, Yosuke Yasuda, and Makoto Yokoo. Strategy-proof matching with regional minimum quotas. In *Thirteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1225–1232, 2014.

[Goto *et al.*, 2016] Masahiro Goto, Atsushi Iwasaki, Yujiro Kawasaki, Ryoji Kurata, Yosuke Yasuda, and Makoto Yokoo. Strategyproof matching with regional minimum and maximum quotas. *Artificial Intelligence*, 235:40–57, 2016.

[Kamada and Kojima, 2015] Yuichiro Kamada and Fuhito Kojima. Efficient matching under distributional constraints: Theory and applications. *American Economic Review*, 105(1):67–99, 2015.

[Kaufmann and Koolen, 2017] Emilie Kaufmann and Wouter M. Koolen. Monte-carlo tree search by best arm identification. In *NIPS*, pages 4897–4906, 2017.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.

[Manlove, 2013] David F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific Publishing Company, 2013.

[Moulin, 1994] Hervé Moulin. Chapter 31 social choice. In *Handbook of Game Theory with Economic Applications*, volume 2, pages 1091–1125. Elsevier, 1994.

[Rosin, 2011] Christopher D. Rosin. Nested rollout policy adaptation for monte carlo tree search. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 649–654, 2011.

[Roth and Sotomayor, 1990] Alvin E. Roth and Marilda A. Oliveira Sotomayor. *Two-Sided Matching: A Study in Game-theoretic Modeling and Analysis (Econometric Society Monographs)*. Cambridge University Press, 1990.

[Roth, 1991] Alvin E. Roth. A natural experiment in the organization of entry level labor markets: Regional markets for new physicians and surgeons in the u.k. *American Economic Review*, 81:415–440, 1991.

[Schaeffer, 1989] J. Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[Wang and Gelly, 2007] Yizao Wang and Sylvain Gelly. Modifications of UCT and sequence-like simulations for monte-carlo go. In *CIG*, pages 175–182. IEEE, 2007.

[Yoshizoe *et al.*, 2011] Kazuki Yoshizoe, Akihiro Kishimoto, Tomoyuki Kaneko, Haruhiro Yoshimoto, and Yutaka Ishikawa. Scalable distributed monte-carlo tree search. In *SOCS*. AAAI Press, 2011.

A Comparison Between Branch-and-bound and UCT

The aggregated linearization method [Bobbio *et al.*, 2021] linearlizes the original objective (i.e., Eq. (2)) and uses an off-the-shelf solver of mixed integer programming. The branch-and-bound (BB) method is one of the most promising methods for the mixed integer programming.[6] BB for a minimization problem involves a lower bound (= solution of the linear relaxation) and an upper bound (= the current best feasible solution). BB and UCT both utilize a tree structure and are an iterative process of refining a solution. Here are the differences between BB and UCT.

- Branch-and-bound, applied to our problem, searches the (linearlized version of) joint space of $(\boldsymbol{x}, \boldsymbol{t}) \in \{0,1\}^{\mathcal{E}} \times \Theta$ whereas UCT only searches for the space of $\boldsymbol{t} \in \Theta$, which is significantly smaller than the joint space. UCT exploits the fact that DA optimizes $\boldsymbol{x}$ given $\boldsymbol{t}$.
- For obtaining a feasible solution, general solvers adopt general heuristics that are ignorant of the problem structure of capacity expansion, whereas UCT exploits the structure of the problem (i.e., the priority of the hospitals and good tree representation on Θ). Giving a good feasible solution enables a more aggressive branching, and we may combine UCT with branch-and-bound in that sense.

In summary, UCT exploits the structure of the problem unlike general solvers with BB. In our simulation, we demonstrate UCT provides a better solution than the intermediate solution of the exact method for large instances with a shorter amount of runtime, even though the implementation of UCT in Python is in favor of the exact method.

B Algorithmic Description of UCT

Algorithm 1 describes the steps in our UCT method. UCT can be viewed as a combination of multi-armed bandit algorithm and monte-carlo search.

C Proof of Theorem 5

Proof. Let $N' = N - 1$. We show that, with a high probability, any node that is *not* in the Δ-optimal subtree is visited as most $\max\left(\frac{N'}{WS(\Delta)}, \frac{(C_p)^2 \log N}{\Delta^2}\right)$ rounds. This implies that UCT spends at most $\max(N-1, WS(\Delta)\frac{(C_p)^2 \log N}{\Delta^2})$ rounds[7] in a path that includes a node outside the Δ-optimal subtree; from which it visits a path in the Δ-optimal subtree if

$$N > WS(\Delta)\frac{(C_p)^2 \log N}{\Delta^2}$$

rounds.

[6]Note that BB is also adapted by Gurobi, which combines BB with many efficient heuristics https://www.gurobi.com/resource/mip-basics/

[7]The number of the nodes that has an incoming edge from a node of Δ-optimal subtree is at most $WS(\Delta)$.

Algorithm 1 Upper Confidence Tree for Searching Capacity Expansion

Require: # of Rounds N.
  Initialize the tree with the root node $\mathcal{T}(1) = \{r\}$.
  for $n = 1, 2, \ldots, N$ **do**
    Set the current node i to the root node: r.
    while Current node i is in $\mathcal{T}(n)$ **do**
      Find the most promising child c and set the current node to c, where

$$\arg\max_{c'} \mathrm{UCB}(c').$$

      {Selection}
    end while
    $k \leftarrow i$.
    Add the current node k to the tree $\mathcal{T}(n+1) \leftarrow \mathcal{T}(n) \cup \{k\}$ and initialize the related statistics $(V_k, N_k) = (0, 0)$. {Development}
    Randomly select a leaf l of $\mathcal{T}_{\mathrm{all}}$, which is one of the descendants of the current node k. Evaluate v_l that is the result of DA with the corresponding expansion. {Simulation}
    Backpropagate value v_l from k up to the root node: For each node i between k and r, it updates $V_i \leftarrow V_i + v_l, N_i \leftarrow N_i + 1$.{Backpropagation}
  end for

Assumption 1 implies that the event

$$\bigcup_{n \geq \max\left(\frac{N'}{WS(\Delta)}, \frac{(C_p)^2 \log N}{\Delta^2}\right)} \left\{U_{i,n,N_c(n)} \geq v_i^*\right\} \tag{3}$$

occurs with probability at most

$$\sum_{n \geq \frac{N'}{WS(\Delta)}}^{N} \frac{1}{n^2} \leq N\left(\frac{WS(\Delta)}{N'}\right)^2 \leq \frac{2(WS(\Delta))^2}{N}$$

and its union bound over $S(\Delta)$ nodes occurs with probability at most $2W^2(S(\Delta))^3 N^{-1}$. In the follows, we assume Eq. (3) for any node in the Δ-suboptimal tree.

Let i be an arbitrary node in the Δ-suboptimal tree and n be an arbitrary round. Suppose that UCT visits a node j that is not in the Δ-suboptimal tree for more than

$$N_j(n) > \max\left(\frac{N'}{WS(\Delta)}, \frac{(C_p)^2 \log N}{\Delta^2}\right) \tag{4}$$

times. We have

$$\begin{aligned} U_{j,n,N_j(n)} &\geq U_{i,n,N_c(n)} \\ &\geq v_i^* \quad \text{(by Ineq. (3))} \end{aligned}$$

Moreover, by definition of the UCB value, we have

$$\begin{aligned} U_{j,n,N_j(n)} &:= \hat{\mu}_{j,N_j(n)} + C_p\sqrt{\frac{\log n}{N_j(n)}} \\ &\leq v_j^* + C_p\sqrt{\frac{\log n}{N_j(n)}} \end{aligned}$$

$$\leq v_j^* + C_p\sqrt{\frac{\log N}{N_j(n)}}$$

Combining these two equations yields

$$N_j(n) \leq \frac{(C_p)^2 \log N}{(\mu_i^* - \mu_j^*)^2} \leq \frac{(C_p)^2 \log N}{\Delta^2}$$

which contradicts with Eq. (4). By contradiction, node j is never visited again after Eq. (4) is satisfied.

In summary, with probability at least $1 - 2W^2(S(\Delta))^3 N^{-1}$, Eq. (3) holds for any node in the Δ-suboptimal tree. Under Eq. (3), any node that is not in the Δ-optimal subtree is visited as most $\max\left(\frac{N'}{WS(\Delta)}, \frac{(C_p)^2 \log N}{\Delta^2}\right)$ rounds, from which it visits at least one path in the Δ-optimal subtree if

$$N > WS(\Delta)\frac{(C_p)^2 \log N}{\Delta^2}.$$

□

D Comparison of Existing Theoretical Results

Assumption 1, which is essentially the same as the one in the literature,[8] is very strong and is not satisfied unless the tree is fully developed.[9] We are not sure if any analysis of UCT gets rid of this limitation. Still, given the overwhelming utility of the UCT method in many practical domains, we consider the value of UCT cannot be overstated.

Recently, there are more solid analyses that rely on less restrictive conditions. However, these analyses tend to modify the original UCT and do not apply to our case. One of the seminal papers by [Feldman and Domshlak, 2014] analyzed a variant of UCT called BLUE. To get rid of stringent assumptions, BLUE considers a two-phase strategy that consists of the development and evaluation phases. Another notable work by [Kaufmann and Koolen, 2017] brought a solid analysis on a fully developed game tree.

Note that the game tree search is inherently different from our problem. In a two-player zero-sum game, the best move for the black player is the worst move of the white player, and thus the value is flipped at each edge, which enables safe pruning of the tree (i.e., alpha-beta cuts [Schaeffer, 1989], or its confidence-based pruning in [Kaufmann and Koolen, 2017]).

E Details of simulations

E.1 Computational environments:

Our program is implemented in Python 3. Linear programming problems and mixed integer programming problems are solved by the Gurobi optimizer. We consider this setting is in favor of the aggregated linearization method, which fully utilizes the power of the state-of-the-art optimizer, whereas Python implementation has some space for improvement (e.g., reimplementation via low-level programming languages). All simulations are conducted on the Google cloud platform (e2-standard-8 instance, eight-core, 32GB memory). All programs are single-threaded.

E.2 Details of synthetic data

We build two sets of experiments. **Set 1** involves B but each hospital does *not* have its expansion quota b_h. **Set 2** involves B as well as b_h for each hospital h. Note that the setting of Set 1 is similar to the experimental section in [Bobbio *et al.*, 2021].

We generate preference lists for hospitals and capacities uniformly at random satisfying the following provisions: (1) no hospital has capacity zero; (2) $\sum_{h\in\mathcal{H}} q_h = D$. Preference lists of residents are generated by the following procedure: (1) generate a common preference vector $\boldsymbol{p}_{\text{common}} \in [0,1]^{\mathcal{H}}$ uniformly at random; (2) generate an vector $\boldsymbol{p}_d \in [0,1]^{\mathcal{H}}$ for each resident $d \in \mathcal{D}$ uniformly at random; (3) calculate each resident's preference by $(1-\alpha)\boldsymbol{p}_d + \alpha\boldsymbol{p}_{\text{common}}$ where $\alpha \in [0,1]$ is the parameter that controls the correlation level of resident preferences; (4) calculate the value $\text{rank}_d(h)$ as the order of the features in $\boldsymbol{p}_d$. For Set 2 experiments, we generate $b_h \in [0, B)$ uniformly at random satisfying $\sum_{h\in\mathcal{H}} b_h \in [B, B \times H)$.

We set $D = 1,000$, and conduct experiments varying the parameters D, B, and α. For each combination of parameters, we average the results for 10 instances.

E.3 Details of real data

We tested the performance of the proposed method in the dataset of Japan Residency Matching Program (JRMP) 2007 [Kamada and Kojima, 2015], which matches medical hospital doctors with residency training programs. For ease of discussion, we call each training program "hospital". Unlike its U.S. counterpart (i.e., the National Resident Matching Program), JRMP does not have "match variation" (e.g., consideration of married couples) and adopts the resident-proposing DA algorithm. There are approximately 10,000 doctors in JRMP 2007. Due to computational limitations, we only used a subset of the data; each region has a regional cap, and we focus on the Tokyo region that admits $1,287$ doctors. The original data contains 123 hospitals, and we clustered them into $C = 50$ (batched) hospitals. Each hospital h has a maximum number of admissions a_h, which we set as $q_h + b_h = a_h$. We set q_h to be proportional to a_h such that $\sum_h a_h = 1,287$. The dataset also contains the number of applications for each hospital. There are total $6,233$ applications (4.84 per hospital). Unfortunately, we do not have the information of which doctor applied to which hospital: We randomly allocate applications such that (1) each doctor applies to at most 8 hospitals[10] and (2) the number of applications for each hospital is the same as the original data. Since the preference is partial (i.e., each doctor only ranks a very limited number of hospitals), we introduce a dummy hospital that has infinite capacity and its rank for each doctor is immediately after the least preferred hospital that the doctor applies. By the definition of

[8] Eq. (3)–(4) in [Kocsis and Szepesvári, 2006]

[9] Here, we use the word "development" to denote the expansion of the upper confidence tree so that it is not confused with the capacity expansion.

[10] Which aligns the original data [Kamada and Kojima, 2015].

stable matching, doctors who do not match the hospitals they applied match the dummy hospital.

In summary, we have $1,287$ residents (doctors) who match $50 + 1$ (1 for a dummy) hospitals with a doctor-side preference. We set $B \in \{10, 30\}$ for the limit of capacity expansion. For each B, we average the results for 10 instances.

E.4 Simulation results for Set 2

Tables 6 and 7 show the results for Set 2. The high-level conclusion of these results is not very different from the results for Set 1.

H	B	α	Baseline		**UCT (proposed)**						
			LPH	Grdy	Iterative	IPT-R	IPT-P	IPT-E	BT-R	BT-P	BT-E
15	30	0.0	20.3	28.0	18.5	25.9	13.6	5.0	18.2	13.5	4.4
15	30	0.2	4.1	3.5	2.6	3.4	1.2	1.5	2.0	1.1	1.0
15	30	0.4	1.5	1.0	0.5	0.02	−0.4	−0.4	0.00	−0.8	−0.9

Table 6: Average percentage gaps between the solution found by the Agg-Lin and the solution found by each method for Set 2 experiments.

H	B	α	Baseline			**UCT (proposed)**						
			Agg-Lin	LPH	Grdy	Iterative	IPT-R	IPT-P	IPT-E	BT-R	BT-P	BT-E
15	30	0.0	247.95	0.02	2.56	143.65	137.46	136.33	127.89	111.65	114.10	109.93
15	30	0.2	2752.07	0.03	6.02	426.97	413.66	412.17	416.50	356.22	389.07	389.58
15	30	0.4	3600.08	0.04	9.38	657.50	624.13	625.07	632.34	592.58	618.25	619.15

Table 7: Average run times for Set 2 experiments (seconds).