Towards Utilitarian Combinatorial Assignment with Deep Neural Networks and Heuristic Algorithms[*]

Fredrik Präntare, Mattias Tiger, David Bergström,
Herman Appelgren, and Fredrik Heintz

Linköping University
581 83 Linköping, Sweden
{firstname.lastname}@liu.se

Abstract. This paper presents preliminary work on using deep neural networks to guide general-purpose heuristic algorithms for performing utilitarian combinatorial assignment. In more detail, we use deep learning in an attempt to produce heuristics that can be used together with e.g., search algorithms to generate feasible solutions of higher quality more quickly. Our results indicate that our approach could be a promising future method for constructing such heuristics.

Keywords: Combinatorial assignment · Heuristic algorithms · Deep learning.

1   Introduction

A major problem in computer science is that of designing cost-effective, scalable *assignment* algorithms that seek to find a *maximum weight matching* between the elements of sets. We consider a highly challenging problem of this type—namely that of black-box *utilitarian combinatorial assignment* (UCA), in which indivisible elements (e.g., goods/courses) have to be distributed in bundles (i.e., partitioned) among a set of other elements (e.g., buyers/students) to maximize a notion of utilitarian social welfare. This is a central problem in both artificial intelligence, operations research, and algorithmic game theory; with applications in optimal task/resource allocation [9], winner determination for combinatorial auctions [13], and team/coalition formation [11].

However, UCA is computationally hard. The state-of-the-art can only compute solutions to problems with severely limited input sizes—and due to Sandholm [12], we expect that no polynomial-time approximation algorithm exists that can find a feasible solution with a provably good worst-case ratio. With this in mind, it is interesting to investigate if and how low-complexity algorithms can generate feasible solutions of high-enough quality for problems with large-scale inputs and limited computation budgets. In this paper, we present preliminary theoretical and experimental foundations for using function approximation algorithms (e.g., deep neural networks) together with heuristic

algorithms to solve UCA problems. We also develop and benchmark a general-purpose method for generating UCA heuristics that can be used together with other algorithms.

2   Related Work

The only black-box UCA algorithm in the literature is an optimal branch-and-bound algorithm [10, 11]. Although this algorithm greatly outperforms industry-grade solvers like CPLEX in difficult benchmarks, it can only solve fairly small problems.

Furthermore, a plethora of heuristic algorithms [16, 5, 3, 19, 4] have been developed for the closely related *characteristic function game coalition structure generation* (CSG) problem, in which we seek to find an (unordered) utilitarian partitioning of a set of agents. However, due to the CSG problem's "unordered nature", all of these methods are unsuitable for UCA unless e.g., they are redesigned from the ground up.

Apart from this, there has been considerable work in developing algorithms for the *winner determination problem* (WDP) [1, 13, 12]—in which the goal is to assign a subset of the elements to alternatives called *bidders* in a way that maximizes an auctioneer's profit. WDP differs from UCA in that the value function is not given in a black-box manner, but instead as a list (often constrained in size) of explicit "bids" that reveal how much the bidders value different bundles of items.

Moreover, heuristic search with a learned heuristic function has in recent years achieved super-human performance in playing many difficult games. A key problem in solving games with massive state spaces is to have a sufficiently good approximation (heuristic) of the value of a sub-tree in any given state. Recent progress within the deep learning field with *multi-layered (deep) neural networks* has made learning such an approximation possible in a number of settings. [8, 17]

Using a previously learned heuristic function in heuristic search is an approach of integrating machine learning and combinatorial optimization (CO), and it is categorized as *machine learning alongside optimization algorithms* [2]. Another category is *end-to-end learning*, in which machine learning is leveraged to learn a function that outputs solutions to CO problems directly. While, the end-to-end approach has been applied to graph-based problems such as the *traveling salesman problem* [6] and the *propositional satisfiability problem* [15], the learned heuristic-based approach remain both a dominating and more fruitful approach [18, 14].

3   Problem Description

The UCA problem that we investigate is defined as the following optimization problem:

Input: A set of elements $A = \{a_1, ..., a_n\}$, a set of alternatives $T = \{t_1, ..., t_m\}$, and a ("queryable black-box") function $\boldsymbol{v} : 2^A \times T \mapsto \mathbb{R}$ that maps a value to every possible pairing of a bundle $C \subseteq A$ to an alternative $t \in T$.

Output: A *combinatorial assignment* (Definition 1) $\langle C_1, ..., C_m \rangle$ over A that maximizes $\sum_{i=1}^{m} \boldsymbol{v}(C_i, t_i)$.

Definition 1. $S = \langle C_1, ..., C_m \rangle$ *is a* combinatorial assignment *over A if $C_i \cap C_j = \emptyset$ for all $i \neq j$, and $\bigcup_{i=1}^{m} C_i = A$.*

Note that there are applications for which it is realistic (or even preferred) to have the value function given in this type of black-box fashion. Examples of this include the strategy game *Europa Universalis 4*, where it is given by the game's programmers to enforce a certain behaviour from its game-playing agents [11]. Other such examples include when it can be defined concisely but e.g., not given as an explicit list due to the problem's size.

Moreover, we use $\boldsymbol{V}(S) = \sum_{i=1}^{m} \boldsymbol{v}(C_i, t_i)$ to denote the value of a *partial assignment* (Definition 2) $S = \langle C_1, ..., C_m \rangle$, and define $||S|| = \sum_{i=1}^{m} |C_i|$. The terms *solution* and *combinatorial assignment* are used interchangeably, and we often omit "over A" for brevity. We also use Π_A for the set of all combinatorial assignments over A, and define $\Pi_A^m = \{S \in \Pi_A : |S| = m\}$. We say that a solution S^* is *optimal* if and only if $\boldsymbol{V}(S^*) = \max_{S \in \Pi_A^m} \boldsymbol{V}(S)$.

Definition 2. *If S is a combinatorial assignment over some $A' \subseteq A$, we say that S is a* partial assignment *over A.*

Now, to formally express our approach to UCA, first let $\langle a_1', ..., a_n' \rangle$ be any permutation of A, and define the following recurrence:

$$\boldsymbol{V}^*(S) = \begin{cases} \boldsymbol{V}(S) & \text{if } ||S|| = n \\ \max_{S' \in \boldsymbol{\Delta}(S, a'_{||S||+1})} \boldsymbol{V}^*(S') & \text{otherwise} \end{cases} \tag{1}$$

where $\boldsymbol{\Delta}(\langle C_1, ..., C_m \rangle, a) = \{\langle C_1 \cup \{a\}, ..., C_m \rangle, ..., \langle C_1, ..., C_m \cup \{a\} \rangle\}$, and S is a combinatorial assignment over $\{a_1', ..., a'_{||S||}\}$. As a consequence of Theorem 1, UCA boils down to computing recurrence (1). Against this background, in this paper, we investigate approximating $\boldsymbol{V}^*$ using neural networks together with heuristic methods with the goal to find better solutions quicker.

Theorem 1. $\boldsymbol{V}^*(S) = \max_{S \in \Pi_A^m} \boldsymbol{V}(S)$ *if $S = \langle C_1, ..., C_m \rangle$ is a partial assignment over A with $C_i = \emptyset$ for $i = 1, ..., m$.*

Proof. This result follows straightforwardly by induction. □

4 Heuristic Function Model and Training

We approximate (1) with a fully connected *deep neural network* (DNN) $f_\theta(S)$ with parameters θ, where S is a partial assignment. Our DNN has three hidden layers using *ReLU* [8] activation functions. Each hidden layer has width $mn+1$. The input is a $m \times n$ *binary assignment-matrix* representation of S, and a scalar with the partial assignment's value $\boldsymbol{V}(S)$. See Fig. 1 for a visual depiction of our architecture.

Our training procedure incorporates generating a data set $\mathcal{D}$ that consists of pairs $\langle S, \boldsymbol{V}^*(S) \rangle$, with randomized partial assignments $S \in \Pi_{A_i}^m$, where $A_i \subset A$ is a uniformly drawn subset from A with $|A_i| = i$, for $i = n - 1, ..., n - \kappa$, where
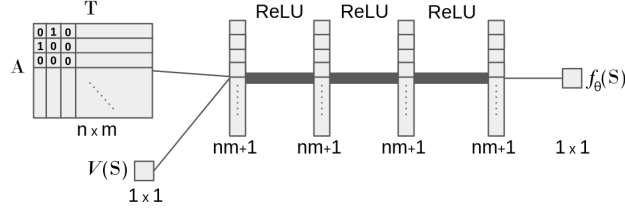
Fig. 1: Our multi-layered neural network architecture.

$\kappa \in \{1, ..., n\}$ is a hyperparameter. In our experiments, $\mathcal{D}$ consists of exactly 10^4 such pairs for every i. Note that it is only tractable to compute $\boldsymbol{V}^*$ if κ is kept small, since in such cases we only have to search a tree with depth κ and branching factor m to compute the real value of $\boldsymbol{V}^*$. For this reason, we used $\kappa \leq 10$ in our benchmarks. θ is optimized over the training data using stochastic optimization to minimize:

$$\mathbb{E}_{\langle S, \boldsymbol{V}^*(S) \rangle \sim \mathcal{D}} \big[\boldsymbol{V}^*(S) - f_\theta(S) \big]. \tag{2}$$

The data set is split $90\%/10\%$ into a training set and a test set. The stochastic optimizer Adam [7] is used for minimizing (2) over the training set. The hyperparameters *learning rate* and *mini-batch size* are optimized using grid search over the test set. In our subsequent experiments, the same $\boldsymbol{V}$ is used as the one used for generating $\mathcal{D}$ by storing the value function's values.

5   Experiments

We use the problem set distributions *NPD* (3) and *TRAP* (4) for generating difficult problem instances for evaluating our method. NPD is one of the more difficult standardized problem instances for optimal solvers [11], and it is defined as follows:

$$\boldsymbol{v}(C, t) \sim \mathcal{N}(\mu, \sigma^2), \tag{3}$$

for $C \subseteq A$ and $t \in T$. TRAP is introduced by us in this paper, and it is defined with:

$$\boldsymbol{v}(C, t) \sim \mathcal{N}(\boldsymbol{\tau}(C), \sigma^2), \tag{4}$$

for all $C \subseteq A$ and $t \in T$, where:

$$\boldsymbol{\tau}(C) = \delta \begin{cases} |C| - |C|^2, & 0 \leq |C| < \tau \\ |C| - |C|^2 + |C|^{(2+\epsilon)}, & \tau \leq |C| \end{cases}$$

for all $C \subseteq A$. $\boldsymbol{\tau}$ is defined to make it difficult for general-purpose greedy algorithms that work on an element-to-element basis to find good solutions by providing a "trap". This is because when $\epsilon > 0$, they may get stuck in (potentially arbitrarily bad) local optima, since for TRAP, the value $\boldsymbol{V}(S)$ of a partial assignment S typically provides little information about $\boldsymbol{V}^*(S)$. In contrast, for NPD, $\boldsymbol{V}(S)$ can often be a relatively accurate
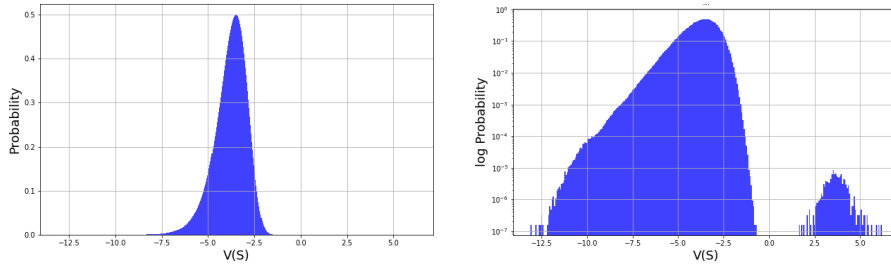
Fig. 2: Empirical estimation of $\boldsymbol{P}\big(\boldsymbol{V}(S)\big)$ for TRAP from 10^8 samples.

estimation for $\boldsymbol{V}^*(S)$. It is thus interesting to deduce whether our learned heuristic can overcome this problem, and consequently outperform greedy approximations.

We used $n = 20$, $m = 10$, $\mu = 1$, $\sigma = 0.1$, $\delta = 0.1$, $\tau = n/2$ and $\epsilon = 0.1$ in our experiments. n and m are chosen to be small enough for exact methods to be tractable.

To give an idea of how difficult it is to find good solutions for TRAP, we plot an empirical estimation of it in Fig. 2, generated using 10^8 draws with (4). The probability of drawing a combinatorial assignment $S \in \varPi_A^m$ at random with a value larger than zero, i.e., $\boldsymbol{P}\big(\boldsymbol{V}(S) > 0\big)$, is approximately 7.43×10^{-6} (only 743 samples found). This was computed using Monte Carlo integration with 10^8 samples.

5.1   Training Evaluation

For NPD, Fig. 3 shows that our neural network generalizes from the 1-5 unassigned elements case to 6-10 unassigned elements with only a slight degradation in prediction error variance (figures to the left). We also see that the predictions are slightly worse for predicting higher assignment values than lower ones, but that the performance is fairly evenly balanced otherwise.

Similar figures for TRAP are also shown in Fig. 3. Here, the prediction error variance is very high around 5-7 unassigned elements. In the right-most figure, we see that the neural network has problems predicting assignment values close to TRAP's "jump" (i.e., $|C| = \tau$). However, outside of value ranges close to the jump, the prediction performance is decent, if not with as high precision as for NPD. Note that TRAP is trained for 1-10 unassigned elements, so no generalization is evaluated in this experiment.
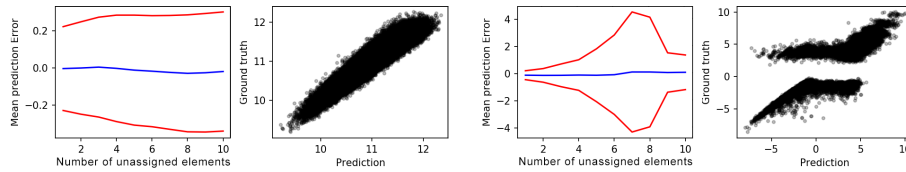

Fig. 3: Neural network results for NPD (left) and TRAP (right). The left figure of each pair shows the mean prediction error and 2 std., as a function of the number of unassigned elements of the partial assignment. The right shows the predicted value compared to the true value.

Despite the seemingly large prediction error variance we find that the neural net-work has a narrower prediction distribution than an uninformed guess. More so for NPD than TRAP, but even a slightly better prediction than random is helpful as a heuristic. This is especially true for TRAP-like distributions, since for them, we previously had no better alternative. Moreover, the prediction errors' distributions are seemingly unbiased.

5.2   Benchmarks

The result of each experiment in the following benchmarks was produced by comput-ing the average of the resulting values from 5 generated problem sets per experiment. In these benchmarks, the goal is to give an indication how well our neural networks per-form compared to more naïve approaches for estimating the optimal assignment's value (and thus their suitability when integrated in a heuristic). These estimation methods are coupled with a standard greedy algorithm to draw samples from the search space. We use the following baseline estimations: 1) *current-value* estimation, which uses the partial assignment's value (so that each evaluation becomes a greedily found local op-timum); and 2) a random approach, which is a worst-case baseline based on a random estimation (so that each evaluation is a uniformly drawn sample from the search space). The best solution drawn over a number of samples is then stored and plotted in Fig. 4. The 95% confidence interval is also plotted. The results show that our neural network is able to overcome some problems element-to-element-based heuristics may face with TRAP. For NPD, it performs almost identical to the current-value greedy approach.

6   Conclusions

We have made the first theoretical and experimental foundations for using deep neural networks together with heuristic algorithms to solve utilitarian combinatorial assign-ment problems. Albeit much remains to be explored and tested, our preliminary results show that using neural networks together with heuristic algorithms could be a promising future method for constructing heuristics of high quality for solving UCA problems.
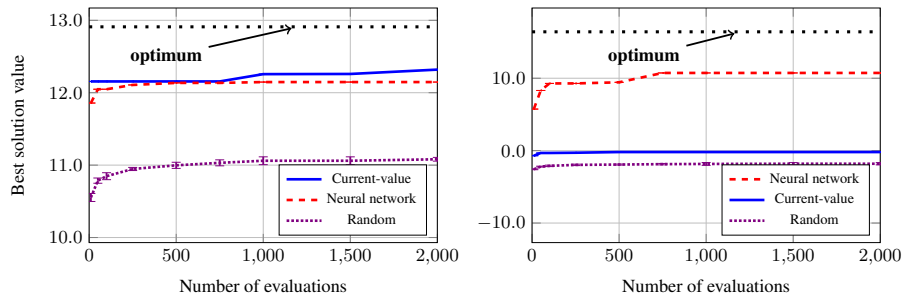

Fig. 4: The best solution values obtained by the different heuristics when using a greedy algorithm for NPD (left) and TRAP (right) problem sets with 20 elements and 10 alternatives.

References

1. Andersson, A., Tenhunen, M., Ygge, F.: Integer programming for combinatorial auction winner determination. In: Proceedings Fourth International Conference on MultiAgent Systems. pp. 39–46. IEEE (2000)
2. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. arXiv preprint arXiv:1811.06128 (2018)
3. Di Mauro, N., Basile, T.M., Ferilli, S., Esposito, F.: Coalition structure generation with grasp. In: International Conference on Artificial Intelligence: Methodology, Systems, and Applications. pp. 111–120. Springer (2010)
4. Farinelli, A., Bicego, M., Bistaffa, F., Ramchurn, S.D.: A hierarchical clustering approach to large-scale near-optimal coalition formation with quality guarantees. Engineering Applications of Artificial Intelligence **59**, 170–185 (2017)
5. Keinänen, H.: Simulated annealing for multi-agent coalition formation. In: KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications. pp. 30–39. Springer (2009)
6. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems. pp. 6348–6358 (2017)
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
8. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
9. Präntare, F.: Simultaneous coalition formation and task assignment in a real-time strategy game. In: Master thesis (2017)
10. Präntare, F., Heintz, F.: An anytime algorithm for simultaneous coalition structure generation and assignment. In: International Conference on Principles and Practice of Multi-Agent Systems. pp. 158–174 (2018)
11. Präntare, F., Heintz, F.: An anytime algorithm for optimal simultaneous coalition structure generation and assignment. Autonomous Agents and Multi-Agent Systems **34**(1), 1–31 (2020)
12. Sandholm, T.: Algorithm for optimal winner determination in combinatorial auctions. Artificial intelligence **135**(1-2), 1–54 (2002)
13. Sandholm, T., Suri, S., Gilpin, A., Levine, D.: Winner determination in combinatorial auction generalizations. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. pp. 69–76 (2002)
14. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al.: Mastering atari, go, chess and shogi by planning with a learned model. arXiv preprint arXiv:1911.08265 (2019)
15. Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., Dill, D.L.: Learning a sat solver from single-bit supervision. arXiv preprint arXiv:1802.03685 (2018)
16. Sen, S., Dutta, P.S.: Searching for optimal coalition structures. In: Proceedings Fourth International Conference on MultiAgent Systems. pp. 287–292. IEEE (2000)
17. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484 (2016)
18. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
19. Yeh, C., Sugawara, T.: Solving coalition structure generation problem with double-layered ant colony optimization. In: 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI). pp. 65–70. IEEE (2016)