Artificial Intelligence Research and Development
T. Alsinet et al. (Eds.)

doi:10.3233/FAIA240453

A Multi-Armed Bandit Approach Using Implicit Hitting Sets for Weighted CSPs

Aleksandra PETROVA [a,1], Javier LARROSA [a] and Emma ROLLON [a]
[a] *Universitat Politècnica de Catalunya, Barcelona, Spain*
ORCiD ID: Aleksandra Petrova https://orcid.org/0000-0001-7485-5309, Javier Larrosa https://orcid.org/0000-0002-8322-0505, Emma Rollon https://orcid.org/0000-0001-8021-9464

Abstract. This ongoing work outlines a Multi-Armed Bandit (MAB) algorithm using the Implicit Hitting Set approach (IHS) to solve Weighted Constraint Satisfaction Problems (WCSP). The paper presents the idea of using MAB to choose among different versions of IHS and efficiently solve the WCSP instance. For the purpose of this work, we create an agent that employs the Roulette Wheel strategy. The agent is then tested on a set of random instances. We show that the MAB does not reach the performance of the best IHS alternative, but consistently outperforms the average, which indicates that our approach is promising.

Keywords. Graphical Models, WCSP, Implicit Hitting Set, Reinforcement Learning, Multi-Armed Bandit.

1. Introduction

Weighted Constraint Satisfaction Problems (WCSP) belong to the group of *Graphical Models* [1] which provide a robust framework for addressing combinatorial problems. In a WCSP, nodes correspond to variables, and edges correspond to cost functions between sets of variables. The main goal of WCSPs is to find the variable-value assignment that minimizes the total cost. WCSPs enable us to solve problems coming from a variety of fields such as bioinformatics, packing, satellite allocation, image analysis, and more.

The solving of Weighted Constraint Satisfaction Problems (WCSPs) has been approached most commonly using heuristics in a search space [2]. However, a new method involving Implicit Hitting Sets (IHS) has emerged [3,4,5]. This approach aims to identify unsatisfiable parts of the problem to reach an optimum. Hitting Vectors, particularly the Minimum Cost Hitting Vector, are used to construct a *Constraint Satisfaction Problem* (CSP) (see [6,7] for details). These vectors help establish lower and upper bounds, allowing for the determination of the optimal cost. When the CSP is deemed unsatisfiable, a core vector can be generated through various methods. Despite the potential for more efficient problem-solving, the most effective approach for a given instance remains unclear.

For this purpose, we consider using other approaches which may give us a better idea. *Reinforcement learning* is one such possibility. Reinforcement learning is a type

[1] Corresponding Author: Aleksandra Petrova, apetrova@cs.upc.edu.

of machine learning where an agent learns to make decisions by taking actions in an environment to maximize cumulative rewards through trial and error [8]. In the scenario where we have multiple possibilities of reaching a reward, we utilize Multi-armed bandits (MAB). The MAB, inspired by a set of slot machines, is a problem where an agent must choose between multiple arms (slots) to maximize the total reward over time, balancing the exploration of new options with the exploitation of known arms. Different strategies can be employed to solve the MAB, including, ε-*greedy*, *roulette wheel*, *UCB*, *Thompson* Sampling, and others.

Given the abilities of the MAB to work well in an online environment, we decided to create an agent that uses eight different variations of the IHS algorithm (2 for growing the core combined with 4 for computing the hitting vector) as arms that it can pull to decide how to do each iteration of the algorithm. In this paper, we present the ongoing work of the aforementioned idea. For the evaluation purposes of this paper, we have created several classes of random WCSP instances, which we solve with each of the eight versions of the arms individually and then compare to the MAB agent.

2. Multi-armed bandit IHS

The Multi-armed bandit agent selects an arm from 8 different options. Each option has different ways of computing a hitting vector and growing it. For growing the core we have used 2 variations: growing the core in each dimension to the maximum point of unsatisfiability for each, and growing the core until we encounter the first dimension where we cannot increase it anymore. The variations for computing the hitting vector are the following: variation *A* computes a Minimum Cost Hitting Vector, verifies if it satisfies the CSP problem, and grows the core if unsatisfiable. Variation *B* computes a low (but not necessarily minimum) cost Hitting Vector using a greedy algorithm. Variation *C* computes a Hitting Vector with a cost smaller than the current upper bound. Variation *D* computes a Hitting Vector with a cost smaller than the midpoint between the lower and upper bounds. Using variation *A*-*D* for computing the hitting vector and the two variations for growing it, we have 8 different alternatives. We propose an MAB algorithm that alternates these options to find the optimal one for the problem instance.

We have chosen the *Roulette Wheel* approach for the MAB agent for two main reasons. Firstly, variation B may end up in infinite loops, so we need to make sure that another arm is used with some significant probability. Secondly, we have observed that using a single version of the IHS algorithm is not the most efficient approach, given that we may choose a slow variation. Therefore, our goal is to learn the probabilities of each arm instead of slowly converging to a single arm for exploitation. To achieve this, we need to be able to identify which arm performs well at a given iteration. For this purpose, we have defined the *reward* as $\frac{(|K|+\sum_{k\in K} cost(k)-cost(h))}{T}$, where K is the set of cores that the arm produced in the iteration, h is the hitting vector obtained in the iteration, and T is the time taken by the arm in seconds.

3. Experiments and results

We conducted an assessment of our proposal by creating a benchmark of various WCSP problems and utilizing them to evaluate our proposal. The first set of instances that we

use is *uniform* random instances, characterized by five parameters: the number of variables (n), domain size (d), number of binary cost functions (m), number of different weights at each cost function (w), and number of tuples with the non-zero cost at each cost function (t). The second set of instances that we generated is *scale-free* networks using the Barabási-Albert model. These instances also consist of five parameters (*n, d, m, w, t*), with n and m being the two parameters of the Barabási-Albert model and *d, w*, and t being the same as in the uniform random instances. For uniform random problems, we generated samples of 50 instances. For scale-free random problems, we generated samples of 20 instances.

We tested the 8 different algorithm variations on their own with a 1-hour timeout and evaluated their average performance. We developed various rewards and approaches and found that the Roulette Wheel approach with the reward we presented above yielded the best results. We then re-ran all instances using this configuration, on the same machine with the same exact resources. The experiment results are presented below.

Problem (n,d,m,w,t)	Best variation	IHS (best)	IHS (average)	IHS (worst)	MAB
Random-15-35-70-700-7	C with 2	237.65	484.87	613.85	438.33
Random-25-30-50-750-5	C with 2	17.43	39.97	53.01	34.77
Random-25-5-50-20-1000	A with 1	875.00	2309.33	3219.27	1439.53
Random-30-8-100-32-150	A with 1	772.37	2050.04	3003.98	862.45
Random-35-6-75-30-12	B/C with 1	144.23	1559.66	3435.64	565.71
Random-40-3-150-3-5	B/C with 1	38.14	581.14	1663.92	104.11
Random-50-5-100-20-5	B/C with 1	20.15	569.65	1979.94	170.99
Random-70-4-175-12-4	B/C with 1	1777.80	2814.29	3600	2802.14
Random-100-4-250-6-15	B with 1	15.85	520.78	1711.57	41.05
Scale-free-4-25-5-20-5	C with 1	36.55	1282.37	3264.93	300.53
Scale-free-4-50-6-15-10	B/C with 1	2045.26	3014.28	3600	2390.68
Scale-free-5-25-5-20-5	B/C with 1	283.73	2108.85	3471.02	1555.46
Scale-free-7-20-3-7-10	C with 1	1.66	33.19	138.19	4.24

Table 1. Performance of the IHS algorithms and the MAB agent. The first column shows the different groups of instances with the parameters (n,d,m,w,t). The second column tells the best IHS version. The letter indicates the hitting vector approach and the number indicates the growing approach (maximal - 1, greedy - 2). In the initial implementation of IHS, variation D was not created, which is why it is not in the column. The times in the table are average running time (in seconds).

We can observe that there is a big difference between the best-performing IHS and the worst. We can also observe that there is no dominant algorithm and different classes of problems have different best algorithm. This means that simply choosing a variation of the IHS algorithm is not an easy of task for a user, which further motivates the approach of using Reinforcement Learning. From the table, we can observe that in all of the cases, MAB performs faster than the average, and in the more complex problem types, MAB can cut down twice the average time. With our MAB we can ensure that the performance will be at least as good as the average of all of the algorithms, and in some cases clearly better.

4. Conclusions and future work

In this paper, we present the progress of our work for using the MAB approach for IHS applied on WCSPs. As part of our work, we explored creating the MAB by using different elements that can influence the rewards to obtain a system that allows us to learn the probabilities of each arm. We additionally explored strategies of the MAB which could help guide the agent to better explore and exploit the space. From our initial results, we can see that indeed the RL approach is a good strategy for choosing an efficient version of the IHS algorithm. Our experiment showed that by using the agent we can ensure that we will solve the instance faster than solving it with the average IHS.

Although the results are positive, there remains a noticeable gap between the performance of the best IHS and our MAB. One potential direction for improvement involves expanding the model to encompass additional arms, each offering distinct strategies for growing the core. Another approach might involve providing the agent with contextual information to inform its decision-making process, thereby transitioning from a Multi-Armed Bandit framework to a Contextual Bandit setting.

References

[1] R. Dechter, Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms, Second Edition, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2019. doi:10.2200/S00893ED2V01Y201901AIM041.
URL https://doi.org/10.2200/S00893ED2V01Y201901AIM041

[2] D. Allouche, S. de Givry, G. Katsirelos, T. Schiex, M. Zytnicki, Anytime hybrid best-first search with tree decomposition for weighted CSP, in: G. Pesant (Ed.), Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings, Vol. 9255 of Lecture Notes in Computer Science, Springer, 2015, pp. 12–29. doi:10.1007/978-3-319-23219-5_2.
URL https://doi.org/10.1007/978-3-319-23219-5_2

[3] J. Davies, Solving MAXSAT by decoupling optimization and satisfaction, Ph.D. thesis, University of Toronto, Canada (2014).
URL http://hdl.handle.net/1807/43539

[4] J. Berg, F. Bacchus, A. Poole, Abstract cores in implicit hitting set maxsat solving, in: L. Pulina, M. Seidl (Eds.), Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings, Vol. 12178 of Lecture Notes in Computer Science, Springer, 2020, pp. 277–294. doi:10.1007/978-3-030-51825-7_20.
URL https://doi.org/10.1007/978-3-030-51825-7_20

[5] P. Saikko, C. Dodaro, M. Alviano, M. Järvisalo, A hybrid approach to optimization in answer set programming, in: M. Thielscher, F. Toni, F. Wolter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018, AAAI Press, 2018, pp. 32–41.
URL https://aaai.org/ocs/index.php/KR/KR18/paper/view/18021

[6] E. Delisle, F. Bacchus, Solving weighted csps by successive relaxations, in: C. Schulte (Ed.), Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings, Vol. 8124 of Lecture Notes in Computer Science, Springer, 2013, pp. 273–281. doi:10.1007/978-3-642-40627-0_23.
URL https://doi.org/10.1007/978-3-642-40627-0_23

[7] J. Larrosa, C. Martinez, E. Rollon, Theoretical and empirical analysis of cost-function merging for implicit hitting set wcsp solving, in: The 38th Annual AAAI Conference on Artificial Intelligence February 20-27, 2024; Vancouver, Canada, AAAI Press, 2024.

[8] R. S. Sutton, A. G. Barto, Reinforcement learning - an introduction, Adaptive computation and machine learning, MIT Press, 1998.
URL https://www.worldcat.org/oclc/37293240