

Meta-level Techniques for Planning, Search, and Scheduling

**Thesis submitted in partial fulfillment
of the requirements for the degree of
“DOCTOR OF PHILOSOPHY”**

by

Shahaf S. Shperberg

**Submitted to the Senate of Ben-Gurion University
of the Negev**

August 5, 2021

Beer-Sheva

Meta-level Techniques for Planning, Search, and Scheduling

Thesis submitted in partial fulfillment
of the requirements for the degree of
"DOCTOR OF PHILOSOPHY"

by

Shahaf S. Shperberg

Submitted to the Senate of Ben-Gurion University
of the Negev

Approved by the advisor 

Approved by the Dean of the Kreitman School of Advanced Graduate Studies

August 5, 2021

Beer-Sheva

This work was carried out under the supervision of
Prof. Eyal Shimony

In the Department of Computer Science

Faculty of Natural Sciences

Research-Student's Affidavit when Submitting the Doctoral Thesis for Judgment

I Shahaf S. Shperberg, whose signature appears below, hereby declare that:
(Please mark the appropriate statements):

X I have written this Thesis by myself, except for the help and guidance offered by my Thesis Advisors.

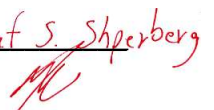
X The scientific materials included in this Thesis are products of my own research, culled from the period during which I was a research student.

X This Thesis incorporates research materials produced in cooperation with others, excluding the technical help, excluding result analysis, commonly applied during such experimental work. Therefore, I attach an additional affidavit stating the contributions made by myself and the other participants in this research, which has been approved by them and submitted with their approval.

 This thesis in in Manuscript Format, includes one or more papers in which I am an "equal contributor". I therefore attach an additional affidavit signed by other equal contributor(s) stating their contribution to the paper and their approval that that paper could not be included in another Manuscript Format Thesis.

Date: August 5, 2021

Student's name: Shahaf S. Shperberg

Signature: Shahaf S. Shperberg


Research-Student's Affidavit: Individual Contribution

This thesis includes several papers which were co-authored by other participants in the research. My contribution in these papers included the development of new ideas and methods, formal problem definitions and analysis, stating and proving interesting theorems, implementation (programming) of algorithms, and designing and running experiments for evaluating the new methods. It is of the essence to emphasize that despite having several co-authors in some of the papers, my work was done individually. Moreover, as the first author of all papers, I had the most significant contribution among all the co-authors. Finally, I confirm that the other co-authors in the relevant papers have also given their consent to incorporate the joint papers into my thesis.

Date: August 5, 2021

Student's name: Shahaf S. Shperberg

Signature: Shahaf S. Shperberg


Acknowledgements

“ Enjoy the journey as much as the destination. ”

Marshall Sylver,

The years of graduate studies have been the best years of my life so far. During these years, I have grown both personally and professionally, while enjoying (almost) every second along the way. I take the opportunity to express my gratitude to the people who accompanied and supported me during this wonderful period.

First and foremost, I thank my advisor and mentor, Prof. Solomon Eyal Shimony, for his continuous support and encouragement. Eyal has not only taught me about academic research and introduced me to the world of artificial intelligence, but also provided me with never-ending guidance. The one thing that was constant throughout the entire period of my studies is that I could always count on Eyal's help and guidance, regardless of the nature of the problem I was facing. His profound expertise, patience, bright ideas, and insightful feedback were invaluable and elevated my work to a higher level. I see Eyal as a role model of the advisor I aspire to one day become.

I would like to pay my special regards to Prof. Ariel Felner. I thank Ariel for our shared work and for the extensive advice I have received from him over the past few years. I have learned a lot from Ariel, and he had a major impact on shaping me into the researcher I am today.

I wish to extend my sincere thanks to the Department of Computer Science at Ben-Gurion University of the Negev. I am thankful for the excellent education I have received, from BSc to PhD. I thank the administrative staff for working relentlessly to provide an environment which enables graduate students to focus on their research.

Acknowledgements

To my fellow graduate students, especially Avi Hayoun, Ohad Ben-Baruch and Shakes Matar, for always being there for research discussions and for the occasional discussions over a cup of coffee, which always brightened up my days. I will always fondly remember the family atmosphere at the department which made teaching and research much more enjoyable.

Last but definitely not least, I would like to thank my family. To my parents for their wise counsel, sympathetic ear, and for always being there for me. To my children, Ofek and Eyal, that fill my life with boundless joy. And finally, a special thanks goes to my spouse Liat for her unconditional love and unwavering support. Liat has been walking with me hand in hand on this journey and has made tremendous sacrifices to help me succeed. I'm eternally grateful to her.

My research was partially supported by the Israeli Science Foundation, by the Lynne and William Frankel Center for Computer Sciences, and by the innovation authority in the Israel Ministry of Trade and Industry (as part of the MDM consortium).

Abstract

Decision making is a fundamental aspect of *artificial intelligence* (AI). The ability to make sensible decisions and to reason about these decisions is a necessary requirement to achieve any level of autonomy. There are many real-world AI applications that use decision making, such as routing and pathfinding, managing automated warehouses (e.g., Amazon's warehouses), medical treatments, autonomous (self-driving) cars, manufacturing (metal bending, submarine/aircraft assembly, etc.), smart oil drilling systems, cognitive assistants, cyber security, web service composition, space exploration (e.g., Mars rovers), creating animation films, playing games, and many more. For most interesting problems, it is impossible to explore all possible decisions and their repercussions within a feasible amount of time, especially in cases where one decision affects subsequent decisions (sequential decision making). Nonetheless, it is of great importance to be able to solve such problems. Therefore, a common practice is to limit the available time for making a decision (reasoning). As a result, algorithms that aim to solve such problems have a very limited time for computation after which they need to choose the decision that they believe would result in the best outcome. Making better decisions would improve a broad range of applications. Thus, it is very desirable to improve the way AI algorithms utilize the time available for reasoning to produce better decisions.

Search and planning algorithms solve decision-making problems by looking ahead and modeling possible future courses of action. This lookahead is structured as a so-called search tree. The root of a search tree represents the initial state of the problem being solved. The successors of a state are all states that can be reached by performing one action (corresponding to making a single decision) from that state. Figure 1 shows a partial search tree for solving a $3 \times 3 \times 3$ Rubik's cube, where each state is a position of the cube and an action is a rotation of the cube once in any direction.

Search algorithms traverse the search tree (perform search) in order to find a goal state, which is a solution to the problem (e.g., a fully arranged Rubik’s cube). Since the search tree faithfully represents the problem being solved, paths that lead to a solution in the search tree (simulation) can be traversed in the real world to solve the actual problem. There are different settings under which search algorithms operate, however, they share the most important challenge of how to spend the computational effort (e.g., which part of the search tree to explore) in order to find solutions quickly.

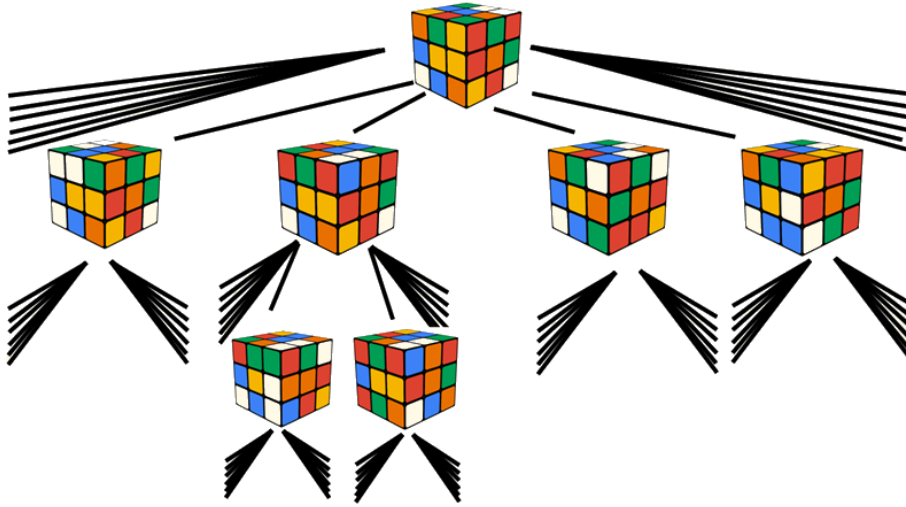


Figure 1: Partial search tree for a Rubik’s cube problem.

Metareasoning is a core idea in AI that captures the essence of being both human-like and intelligent. The idea is that much can be gained by thinking (reasoning) about one’s own thinking. In the context of search and planning, metareasoning involves making explicit decisions about computation steps, by comparing their ‘cost’ in computational resources, against the gain they can be expected to make towards advancing the search for a solution (or plan), and thus making better decisions. To apply metareasoning, a meta-level problem needs to be defined and solved with respect to a specific framework or algorithm. In some cases, these meta-level problems can also be very hard to solve (sometimes even harder than the original search problem). Yet, even a fast-to-compute approximation of meta-level problem solutions can yield good results and improve the algorithms to which they are applied.

This dissertation focuses on the development and evaluation of different meta-reasoning techniques, tailored for different problem settings, designed to improve a variety of search, planning and scheduling algorithms.

Contents

Introduction	1
1 Metareasoning in MCTS	3
2 Situated Temporal Planning	5
3 Algorithm and Instance Selection	6
4 Bidirectional Heuristic Search	6
 A Monte-Carlo Tree Search using Batch Value of Perfect Information	 11
1 Introduction	13
2 Background	14
2.1 Monte-Carlo Tree Search	14
2.2 Information Gathering in Trees	15
2.3 Conspiracy Numbers	18
3 Batch Value of Information	19
3.1 Computing BVPI	20
4 Practical Batch-selection	23
4.1 High-BVPI sets: Using Conspiracies	23
4.2 Experiments on Game Trees	26
5 Plugging BVPI into MCTS	27
5.1 Selecting Rollouts	27
6 Empirical Evaluation: MCTS	30
6.1 Canadian Traveler Problem (CTP)	30
6.2 StarCraft Battles	32
7 Conclusion	33
 B Some Properties of Batch Value of Information in the Selection Problem	 39
1 Introduction	41
2 Main Results	45

2.1	Batch VOI when the Utility of the Currently Best Item is Known	47
2.2	Batch VOI when the Utility of the Currently Best Item is Unknown	52
3	Application of Results	55
3.1	Example Setting: Wine Selection	56
4	Conclusion	63
C	Allocating Planning Effort when Actions Expire	71
1	Introduction	73
1.1	Problem Statement	74
2	Previous Work	75
2.1	Deliberation Scheduling	75
2.2	Job Scheduling	76
3	The Deliberation Scheduling MDP	77
3.1	Solution Complexity and Approximations	79
3.2	SEA2 with Diminishing Returns	84
3.3	Non-diminishing returns	85
3.4	Real-time Deliberation Scheduling	87
4	Empirical Evaluation	88
5	Discussion	91
D	Trading Plan Cost for Timeliness in Situated Temporal Planning	95
1	Introduction	97
2	Problem Statement: Cost vs Timeliness	98
3	Deliberation Scheduling MDP with Costs	100
4	Theoretical Analysis of ACE2	102
4.1	Complexity of ACE2	102
4.2	The Case of Known Costs	104
4.3	Simple Special Case: One Running Process	105
5	A Greedy Scheme With Costs	107
6	Empirical Results	109
7	Conclusion	111
E	Situated Temporal Planning Using Deadline-aware Metareasoning	115
1	Introduction	117

Contents

2	Background	119
2.1	Problem Statement	119
2.2	Metareasoning in Situated Planning	120
2.3	Basic Greedy Scheme	122
3	New Metareasoning Schemes	124
3.1	DP Solution for Known Deadlines	124
3.2	Delay-Damage Aware Greedy Scheme	126
3.3	Evaluation on $S(AE)^2$	128
4	Integrating DDA into a Planner	131
4.1	Estimating the Distributions	131
4.2	Searching with DDA	132
5	Empirical Evaluation	133
5.1	Comparing DDA to the Baseline	133
5.2	DDA Ablation Studies	135
5.3	Automated Parameter Tuning for DDA	136
5.4	Evaluating the Impact of Tight Deadlines	137
6	Discussion	138
F	Algorithm Selection in Optimization and Application to Angry Birds	143
1	Introduction	145
2	Formal Problem Statement	147
2.1	Performance Models	148
3	Analysis: Known IID Case	149
3.1	Complexity: Restricted Versions	149
3.2	Approximation Algorithms	151
4	Experiments: Known IID	152
5	Unknown Distributions	155
5.1	Unknown IID Score and Runtime	155
5.2	Experiments: Unknown IID	157
5.3	Experiments: Angry Birds Game	159
6	Discussion	160
G	Enriching Non-parametric Bidirectional Search Algorithms	167

1	Introduction and Overview	169
2	Definitions and Background	170
2.1	Guaranteeing Solution Optimality	171
2.2	The Must-Expand Graph (G_{MX})	172
2.3	The Minimum Vertex-Cover of G_{MX}	172
3	Finding All Optimal Solutions	174
3.1	G_{MX} for Finding All Optimal Solutions	174
4	A General Framework Encompassing NBS	176
4.1	The Low-Level Expansion Policy of NBS	177
4.2	Finding All Optimal Solutions with NBS	178
4.3	Finding a First Solution with NBS_A	179
5	Bidirectional Search using Dynamic VC	180
5.1	Low-Level Expansion Policy in DVCBS	180
5.2	Variants of DVCBS	181
5.3	No Upper Bound Guarantees for DVCBS	183
6	Experimental Evaluation	186
7	Conclusions and Future Research	188
H	Improving Bidirectional Heuristic Search by Bounds Propagation	193
1	Introduction	195
1.1	Definitions and Background	196
1.2	Fractional MM	197
1.3	GBFHS	198
2	The Well-Behavedness Property	199
2.1	Example of the Anomaly for fMM	200
2.2	Guaranteeing Solution Optimality	201
2.3	Conditions for Being Well-Behaved	203
3	The Reasonableness Property	204
4	Improving Heuristics by lb -propagation	205
4.1	Propagating g - and f -values	205
4.2	lb -propagation heuristic	206
5	Classification of Existing Algorithms	207
5.1	BHPA	207

Contents

5.2	BS*	208
5.3	fMM	210
5.4	NBS and DVCBS	212
5.5	GBFSH	214
6	Experimental results	214
7	Discussion	217
Discussion		221

Contents

Introduction

Artificial intelligence (AI) applications are designed for many different environments. As a result, there are various settings under which decision-making algorithms need to operate. In some cases, problems are simple enough to be completely solved (either optimally or within a constant bound from the optimal solution) in feasible time. Then, the solution (sequence of actions) can be fully executed in the real world. In other cases, problems induce deadlines which decision-making algorithms need to obey. For example, some decisions can be associated with time constraints, like deciding to take a train at a specific hour or to visit a place with limited opening hours. In these cases, decision-making algorithms need to carefully decide how to spend their computational effort, as the deadlines of some actions (decisions) can expire. Finally, there are cases in which deadlines must be imposed on algorithms solely to make them act in real-time. This restriction corresponds to the “finitary predicament” introduced by Cherniak 1986, which states that humans have a limit on their cognitive capability and available time. Likewise, agents (algorithms) have limited computational resources and the time which is available to them to make decisions is limited. For example, even for a simple problem such as a standard $3 \times 3 \times 3$ Rubik’s cube there are $\approx 4.3 \times 10^{19}$ different possible configurations (states of the cube). Algorithms cannot completely explore such a vast space of possibilities (search tree) to find an optimal solution before taking any action (making a decision) in the real world, as the search for such a solution takes an immense amount of time with the computational capabilities available today (or in the foreseeable future). Moreover, there are much more complicated real-life problems, or even games in which the state space is greater by many orders of magnitude. For example, the number of states in popular real-time strategy game *StarCraft* is conservatively estimated to be 10^{1685} (Ontañón et al. 2013). In comparison, the number of particles in the universe is esti-

mated to be between 10^{80} and 10^{100} . Therefore, there are many interesting problems for which there will likely never be sufficient computational resources for algorithms to solve optimally in feasible time. Thus, algorithms which aim to solve such problems are given a limited time to consider each decision (action), which they use to explore a small part of the search tree by performing a lookahead from the current state and estimating the value of descendant states. When the time for search has run out, these algorithms commit to a decision based on the value estimations obtained during the search, and act in the real world. The main challenges for these algorithms are to decide which part of the search tree to explore and which action to take when the time for reasoning (search) runs out.

Metareasoning (sometimes called metacognition) is the act of deliberating about one's own process of thought. This concept has been studied for many years, as early as the work of the Greek philosopher Aristotle (384–322 BC) (Colman 2015). In the context of AI systems, agents, and algorithms, metareasoning is a deliberation regarding changes in the integral (computational) state. Meta-level algorithms aspire to maximize the expected utility of solutions, also considering deliberation costs (also known as “type II rationality”, (Good 1971)); or equivalently, to optimally utilize the available (limited) computational resources and time in order to improve the quality of decisions.

A prominent approach to metareasoning is called *rational metareasoning* (Russell and Wefald 1991). In this framework, computations are treated as actions. This induces a meta-level selection problem of choosing the best action (computation) at each step. The decision of which computational action to take is made without knowing the actual outcome of the computations. Thus, computations are treated as stochastic actions, even in cases when their outcome is deterministic. Therefore, actions can be evaluated based on their (estimated) expected utility. The cost of an action is proportional to the resources it requires. Russell and Wefald 1991 defined a *net value of computation* (which is similar to Howard 1966's value of information) as the intrinsic utility of the computation minus the cost of the computation (the cost of computation should be in the same units as the utility). The action with the highest net value of computation is selected by the meta-level. While the rational metareasoning approach is very methodological, it is intractable to compute the value of information in general.

Moreover, the overhead of meta-level problems should be low, as the time required to solve meta-level problems comes at the expense of the time available for reasoning (e.g., less time for performing a lookahead). Thus, in order to practically implement this framework Russell and Wefald 1991 had to make many simplifying assumptions. For example, they only consider the utility of a single computational action at a time. Furthermore, there are problem settings which are hard, or even impossible to define using this framework. For example, a setting in which deadlines are induced by real constraints, as briefly discussed above, cannot be directly expressed in the rational metareasoning framework.

This research focuses on developing and applying metareasoning techniques for a variety of interesting problem settings. This demonstrates the versatility of meta-reasoning, as well as the potential of meta-level techniques to improve the decision-making abilities of existing algorithms and to adapt them to new settings. Solving meta-level problems is notoriously hard. Moreover, every second spent on metareasoning is less time available for reasoning (i.e., searching, planning, etc.). Nonetheless, this research shows that by considering the right meta-level problems and finding quick approximations to them, the utility of algorithms can be considerably improved. The different settings and the contributions of this research in each setting are summarized below.

1 Metareasoning in MCTS

Monte-Carlo tree search (MCTS) is an algorithmic schema commonly used to search huge trees, especially when a good heuristic is not available (Browne et al. 2012). In general, MCTS grows a search tree, using four phases: node-selection, node-expansion, simulation, and backup. When the time for search is over, the action which leads to the most-promising child of the root (the one with the highest estimated utility, denoted as α) is performed. The scheme used for node-selection essentially controls the search by deciding where to focus the computational effort. A popular node-selection approach is the Upper Confidence Bounds for Trees (UCT, Kocsis and Szepesvári 2006) which aims to minimize cumulative regret and "balancing exploration and exploitation". However, it was shown that minimizing cumulative

regret is actually inappropriate for move selection and that the simple regret and *value of information* (VOI) criteria are more appropriate and result in more efficient search (Tolpin and Shimony 2012; Feldman and Domshlak 2014). In essence, the VOI of selecting nodes for simulation is proportional to the probability of changing α (either by increasing the estimated utility of some other child of the root or by decreasing the estimated utility of α) as a result of running simulation on these nodes. VOI can be defined in different ways, each way uses different assumptions which induce different computational overheads. The two previous node-selection schemes that are based on VOI are MGSS (Russell and Wefald 1991) and “blinker” (Tolpin and Shimony 2012), both schemes are different implementations of the rational metareasoning framework under different sets of assumptions. However, both schemes only consider the VOI of individual nodes; this is a very myopic approach that often prematurely commits to α .

Our work in Paper A, (Shperberg, Shimony, and Felner 2017) attempts to relax this myopic assumption. We define a batch value of perfect information (BVPI) as a generalization to the value of computation proposed by Russell and Wefald. We show that computing BVPI is NP-hard, but it can be approximated in polynomial time. In addition, we propose a node-selection scheme that intelligently find sets of nodes with high BVPI. Finally, we apply our BVPI based selection-scheme to existing MCTS-based applications and empirically show that our methods outperform existing node-selection techniques in different scenarios.

In Paper B (Shperberg and Shimony 2017) we examine theoretical properties of VOI when aiming to select an item (or action) of unknown utility. This model assumes that measurements (possibly noisy) of item values prior to selection are allowed, at a known cost. The goal is to optimize the overall sequential decision process of measurements and selection. Unfortunately, this decision problem is intractable in general. Yet, it is important to be able to solve, at least approximately, as it has numerous potential applications. This paper analyses cases where the value of information (VOI) is submodular and supermodular, and suggests how to use these properties to approximate optimal batch measurement policies.

2 Situated Temporal Planning

Agents that plan and act in the real world must deal with the fact that time passes as they are planning. For example, an agent that needs to get to the airport may have two options: take a bus or take a train. Each of these options can be thought of as a *partial plan* to be elaborated into a complete plan that can be executed. Furthermore, consider a second example in which there are two partial plans, each estimated to require four minutes of computation to elaborate into complete plans. If only five minutes remain until both plans expire, then we would want the planner to allocate all its remaining effort to one candidate plan, rather than to fail on both.

Cashmore et al. 2018 recognized the problem of node expiration in the context of temporal planning with timed initial literals (TIL), where the TILs occur at times that are relative to when *planning* starts, rather than to when execution starts. However, their approach is relatively superficial and was used merely to prune nodes that become infeasible based on their latest start time estimation. Such a planner fails on the latter example.

In Paper C (Shperberg, Coles, Cserna, et al. 2019) we define the problem of selecting on which nodes to focus during planning as a meta-level problem called *Allocating Effort when Actions Expire* (AE2). AE2 abstracts away from the planning problem and merely assumes n independent processes, each modeled by a distribution over wall clock times denoting the deadline, and a distribution over the required time allocation to complete computation. The objective of AE2 is to schedule processing time among the n processes so that the probability that at least one process will find a solution before its deadline is maximized. We have analyzed properties of the AE2 problem, developed a pseudo-polynomial time solution for the special case of known deadlines, and proposed an effective greedy algorithm for the general case. In addition, Paper D (Shperberg, Coles, Karpas, Shimony, et al. 2020) tackles the extended problem (called ACE2) where processes (plans) have associated costs, and the aim is to minimize the expected cost. Finally, in Paper E (Shperberg, Coles, Karpas, Ruml, et al. 2021) we show how our greedy scheme for the AE2 problem can be used within a real situated temporal planner (OPTIC). An empirical evaluation suggests that the modified planner provides state-of-the-art results on problems where external deadlines play a

significant role.

3 Algorithm and Instance Selection

In the context of multiple agents sharing resources, metareasoning can be used for dividing the shared resources among all agents. In Paper F (Shperberg, Shimony, and Yehezkel 2019), we consider a set of black-box agents/algorithms (each with its own strengths and weaknesses) attempting to solve a pool of optimization problem instances. Unlike standard algorithm selection settings in which there is an individual time limit for each instance, here we have a global time limit for the entire set of instances. The goal is to maximize the sum of solution qualities, where each instance can be solved more than once, but only the best solution counts. Thus, a policy consists of a pair of problem instance and algorithm to execute at any given moment. The original motivation for this work was combining multiple programs that compete in the AI Angry Birds competition. In this competition, agents have 30 minute to play 8 unseen levels of the Angry Birds video game, with total score being the sum of level scores. We formulate the problem as a selection problem and show that it is NP-hard even when the time and score performance profiles (distributions) of agents on levels are known and independent. Nonetheless, we develop an approximation algorithm for one simple case, as well as faster greedy algorithm for the general case which works well empirically on data collected from the Angry Birds game. Then, we combine this greedy algorithm with a Bayesian learning scheme for obtaining and updating the performance profiles. An empirical evaluation using the competition settings of past years suggests that the combined algorithm outperforms the individual agents.

4 Bidirectional Heuristic Search

In bidirectional heuristic search (Bi-HS) we are given an implicit graph, a start vertex s , a goal vertex g , and a heuristic which estimates the cost of the least-cost path between vertices. The aim is to find a least-cost path between s and g . Bi-HS algorithms maintain two search frontiers, one from s and one from g , and need to connect them to find a solution.

A fundamental question in Bi-HS is “*how much of the search effort to invest in each frontier?*”. In an attempt to answer this question, Eckerle et al. 2017 investigated which nodes must be expanded by any Bi-HS algorithm in order to prove solution optimality. While in unidirectional search there is a specific set of nodes for every problem instance that must be expanded (Dechter and Pearl 1985), in Bi-HS there is no such unique set. Instead, for every problem instance there are pairs of nodes (called must-expand pairs), *one of which must be expanded*. Each such pair contains a node from the forward frontier and a node from the backward frontier. Thus, these pairs induce many sets (of different sizes) of nodes, from which one set needs to be fully expanded. Each set corresponds to a division of the search effort between the forward and backward frontiers. Aiming to find a small set to expand, Chen et al. 2017 converted the problem of expanding must-expand pairs to a problem of finding a vertex cover in an abstract graph which they called G_{MX} . The minimal vertex cover (MVC) of G_{MX} is the minimal number of node expansions required to prove optimality of solutions. While G_{MX} can only be fully constructed in post analysis, it is possible to obtain some of its edges during the search. NBS (Chen et al. 2017) is a prominent Bi-HS algorithm which always expands both nodes of each such edge, in order to get a $2 \times$ MVC bound on the number of expansions. Paper G (Shperberg, Felner, Sturtevant, et al. 2019) presents other ways to exploit G_{MX} structure for node expansion. Instead of choosing a single edge from G_{MX} our algorithm, called DVCBS, maintains a dynamic sub-graph of G_{MX} (denoted as DG_{MX}) using the frontier nodes. Then, DVCBS computes an MVC for this DG_{MX} (in linear time) and uses it to choose which nodes to expand. DVCBS faces a tradeoff. On the one hand, constructing a DG_{MX} and computing its MVC often is more accurate and leads to better search results, as new frontier nodes become available. On the other hand, these operations are computationally expensive. This tradeoff induces a metareasoning problem. Finally, in Paper H (Shperberg, Felner, Shimony, et al. 2019) we develop a method for enabling existing algorithm to benefit from G_{MX} structure by incorporating this information into the heuristic function. Not only do algorithms that use the enhanced heuristic require less node expansions to solve problems, some of them also benefit from desirable theoretical properties gained only by using this enhanced heuristic.

List of Papers

This dissertation consists of the following publications.

- Paper A – Shahaf S. Shperberg, Solomon Eyal Shimony, and Ariel Felner. Monte-Carlo Tree Search using Batch Value of Perfect Information. In Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence (UAI '17), 2017.
- Paper B – Shahaf S. Shperberg and Solomon Eyal Shimony. Some Properties of Batch Value of Information in the Selection Problem.
In Journal of Artificial Intelligence Research, volume 58, pages 777–796, 2017.
- Paper C – Shahaf S. Shperberg, Andrew Coles, Bence Cserna, Erez Karpas, Wheeler Ruml, and Solomon E. Shimony. Allocating Planning Effort when Actions Expire.
In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI '19), pages 2371–2378, 2019.
- Paper D – Shahaf S. Shperberg, Andrew Coles, Erez Karpas, Eyal Shimony, and Wheeler Ruml. Trading Plan Cost for Timeliness in Situated Temporal Planning.
In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI '20), pages 4176–4182, 2020.
- Paper E – Shahaf S. Shperberg, Andrew Coles, Erez Karpas, Wheeler Ruml, and Solomon E. Shimony. Situated Temporal Planning Using Deadline-aware Meta-reasoning.
In Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS '21), 2021.
- Paper F – Shahaf S. Shperberg, Solomon Eyal Shimony, and Avinoam Yehezkel. Algorithm Selection in Optimization and Application to Angry Birds.
In Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS '19), pages 437–445, 2019.
- Paper G – Shahaf S. Shperberg, Ariel Felner, Nathan R. Sturtevant, Solomon Eyal Shimony, and Avi Hayoun. Enriching Non-parametric Bidirectional Search

4. Bidirectional Heuristic Search

Algorithms.

In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI '19), pages 2379–2386, 2019.

- Paper H – Shahaf S. Shperberg, Ariel Felner, Avi Hayoun, Solomon Eyal Shimony, and Nathan R. Sturtevant. Improving Bidirectional Heuristic Search by Bounds Propagation.

In Proceedings of the Twelfth Annual Symposium on Combinatorial Search (SoCS '19), pages 106–114, 2019.

Paper A

Monte-Carlo Tree Search using Batch Value of Perfect Information

Shahaf S. Sheprberg, Solomon Eyal Shimony

The paper has been published in the
*UAI '17: Proceedings of the Thirty-Third Conference on Uncertainty in Artificial
Intelligence, UAI, 2017*

© 2017 UAI

The layout has been revised.

Abstract

This paper focuses on the selection phase of Monte-Carlo Tree Search (MCTS). We define batch value of perfect information (BVPI) in game trees as a generalization of value of computation as proposed by Russell and Wefald, and use it for selecting nodes to sample in MCTS. We show that computing the BVPI is NP-hard, but it can be approximated in polynomial time. In addition, we propose methods that intelligently find sets of fringe nodes with high BVPI, and quickly select nodes to sample from these sets. We apply our new BVPI methods to partial game trees, both in a stand-alone set of tests, and as a component of a full MCTS algorithm. Empirical results show that our BVPI methods outperform existing node-selection methods for MCTS in different scenarios.

1 Introduction

Monte-Carlo tree search (MCTS) algorithms, such as the UCT algorithm and its many variants (Browne et al. 2012), are state-of-the-art in numerous domains. A crucial phase in MCTS is the *selection phase* where a fringe node of the partially expanded tree is selected for sampling (initiating rollouts). Although UCT is a prominent approach, its node-selection criterion, based on optimization of *cumulative regret*, is actually inappropriate for move selection: it was shown in (Hay et al. 2012; Feldman and Domshlak 2013) that *simple regret* and value of information (VOI) criteria are more appropriate, and result in more efficient search. A scheme similar to the “value of computation” of (Russell and Wefald 1991b; Russell and Wefald 1991a) can be used to define the VOI (see Section 2.2). Since a single rollout cannot find the true utility of a node, the “blinker” scheme in (Hay et al. 2012) provided bounds on the VOI for a *number of samples* at a node. Despite its success, the “blinker” scheme has two shortcomings. First, it applies only at the first level of the tree. Second, it only considers the VOI of individual nodes.

Numerous authors attempted to optimize VOI for multiple sources of information. Such an optimization is intractable in general (Reches et al. 2013; Krause and Guestrin 2011; Krause and Guestrin 2009). Approximations thereof using myopic assumptions and greedy search are a common way to alleviate this problem (Krause and Guestrin 2011). The myopic-greedy schemes have a basis in theory, due to the fact that if the VOI is submodular, greedy algorithms are provably near-optimal (Krause

and Guestrin 2009; Krause and Guestrin 2011; Papachristoudis and Fisher III 2012). However, the VOI is not submodular in general (Krause and Guestrin 2009), thus myopic-greedy metareasoning can be far from optimal.

We address the above shortcomings by defining **batch value of perfect information** (BVPI) of multiple nodes in game trees, as a generalization of *value of computation* (Russell and Wefald 1991a). First, we examine the computational complexity of BVPI, and show a deterministic approximation (Section 3). Second, we introduce algorithmic variants that quickly choose a set of nodes S with high BVPI, and evaluate them on trees generated by MCTS algorithms (Section 4). Third, we present variants that quickly select nodes in S for rollouts. We provide empirical evidence of improved rollouts effectiveness without incurring too much overhead. Finally, we show how all ideas can be plugged into any MCTS algorithm by modifying only its selection phase (Section 5). Experimental results (Section 6) on two disparate domains show that our methods significantly outperform the node selection schemes used by UCT and “blinkered”, as well as that of an adaptation of MGSS* (Russell and Wefald 1991a) to MCTS.

2 Background

This paper uses techniques from Monte-Carlo tree search, value of computation (value of information) in game trees, and conspiracy numbers. We briefly examine each below.

2.1 Monte-Carlo Tree Search

Monte-Carlo tree search (MCTS) is an algorithmic schema commonly used to search huge trees. In general, MCTS grows a search tree, using four phases: node **selection**, node **expansion**, **simulation** (also called rollouts or sampling), and **updating** (also called backup). Algorithm 1 depicts this the MCTS, following (Browne et al. 2012)).

TreePolicy() consists of the **node selection** and **expansion**, DoRollout() performs simulation (sampling) from the selected node v . The computation budget (line 2) can be the number of rollouts, a time limit, etc. There are numerous schemes for deciding which nodes to expand, which nodes to select, how to propagate updates, and how to

Algorithm 1: Monte-Carlo Tree Search

```

1 function MCTS(root):
2   while computation budget not exceeded do
3      $v \leftarrow \text{TreePolicy}(\text{root})$ 
4      $U \leftarrow \text{DoRollout}(v)$ 
5      $\text{Backup}(v, U)$ 

```

do rollouts (See (Browne et al. 2012) for a deep survey). A popular approach for node selection is the Upper Confidence Bounds for Trees (UCT) (Kocsis and Szepesvári 2006), which selects a node v' repeatedly, starting from the root, down to the fringe. At each node v , select a child v' which maximizes the score:

$$\text{UCTscore}(v') = Q(v') + b\sqrt{\frac{2 \ln N(v)}{N(v')}} \quad (\text{A.1})$$

where $Q(v')$ is the previous average value of v' , $N(v)$ is the number of past visits of v , and b is a constant. If v is a fringe node, a rollout is initiated from v .

2.2 Information Gathering in Trees

We are given a game tree T consisting of MAX, MIN, and CHANCE nodes (root r is a MAX node). Each leaf has known utility value distribution P_v , as in Figure A.1, but its true utility $u(v)$ is unknown. Leaf utility distributions are assumed independent (the commonly used “subtree independence” assumption (Russell and Wefald 1991a)). The notation $[p_1 : u_1, \dots, p_n : u_n]$ represents P_v , meaning that p_i is the probability that the true utility is u_i (for all i).

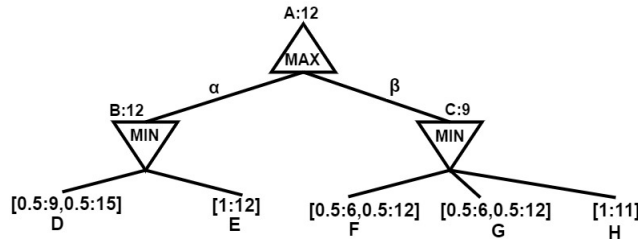


Figure A.1: MIN-MAX tree with leaf utility distributions

For each leaf v of T , we can choose to perform a measurement that costs $C(v)$, obtaining further information about v , such as the true utility of v . Optionally, there

is also a budget constraint that limits the total number, or cost, of allowed measurements. The problem of *optimal information gathering* is: what is the optimal policy of performing measurements, and then selecting the action at the root, so as to achieve maximum overall expected utility?

There are two standard settings of this problem. In the *batch setting*, all measurements are made in a single batch. Only then, the decision maker gets to observe the results of the measurements. In the *sequential setting*, the decision maker selects a measurement and immediately observes the result. This is repeated until a decision to stop is reached. In both settings, after stopping the measurement process, the decision maker selects the root action that achieves the best expected utility for the tree given all the observations.

In this section, as well as in Section 3, we assume that the distributions P_v are given, and that the measurements are abstract operators that have a known cost and reveal the true utility $u(v)$. But in the context of a search algorithm, T is actually the partially developed game tree, of which v is a fringe node. A measurement is done by a computation, obtaining additional (usually **noisy**) information about v by expanding it, or (in MCTS) by initiating additional rollouts from v . The initial distributions P_v are obtained by heuristics, by rollouts (as in Section 5) etc.

Value of Perfect Information

Suppose that the agent has developed the tree T . If no additional measurements were allowed, a MAX player should compute an EXPECTI-MINI-MAX value $\bar{U}(T)$ of the tree, treating the utility of each leaf v of T as if it were equal to its expected value. Let α be the child of the root which propagated $\bar{U}(T)$ to the root node in EXPECTI-MINI-MAX. We call this move α the “current best” (see Figure A.1). In order not to introduce additional notation, as far as utility distributions are concerned, we henceforth refer to a move, and to its respective child node, interchangeably.

In a nutshell (See (Russell and Wefald 1991a)), the *Value of (Perfect) Information* (VPI), (called *Value of Computation* in (Russell and Wefald 1991a)) is the expected gain from picking another move β_i as a result of performing the additional measurements either under α or under β_i . Let β_1 be the next-best move after α , and denote by $\bar{U}(\gamma)$ the current expected utility of any move γ . Let S_α be a set of leaves under α , and let

2. Background

$p_{\alpha S_\alpha}(x)$ be the probability density function for the utility of move α , given the utility observations at the leaves S_α . The VPI for S_α is defined as:

$$VPI(S_\alpha) = \int_{-\infty}^{\bar{U}(\beta_1)} p_{\alpha S_\alpha}(x)(\bar{U}(\beta_1) - x)dx \quad (\text{A.2})$$

Intuitively, $VPI(S_\alpha)$ is the gain due to preferring β_1 over α because measurements below α revealed that the new value of α (given the observations) is worse than the current expected value of β_1 . Likewise, for a move β_i (not the current best), let S_{β_i} be a set of leaves under β_i , and denote by $p_{\beta_i S_{\beta_i}}(x)$ the probability density function for the utility of move β_i , given utility observations at leaves S_{β_i} . Then:

$$VPI(S_{\beta_i}) = \int_{\bar{U}(\alpha)}^{\infty} p_{\beta_i S_{\beta_i}}(x)(x - \bar{U}(\alpha))dx \quad (\text{A.3})$$

The distributions in Eqs. A.2, A.3 were defined in (Russell and Wefald 1991a) for MIN-MAX trees. Equation A.4 below is an alternative statement that incorporates CHANCE nodes.

The MGSS* (Meta-Greedy Single-Step) scheme in (Russell and Wefald 1991a) uses Eqs. A.2, A.3 assumes that the measured node-set S is a single node: their “single-step assumption” of computing the value of computation under the assumption that only one step of computation will be done before the final move decision (see the following example). They extended the scheme into MGSS2 that estimates the value of computation for more than one node.

Example A.1

In Figure A.1, α is the current best move, since $\bar{U}(\alpha) = 12 > 9 = \bar{U}(\beta)$, due to the MIN-MAX computation in the tree, using the expected value of leaf node distributions as their known value. For example, the value of leaf node D is taken to be 12. Obtaining the true utility $u(v)$ of any one leaf cannot change the move selected by MAX. Thus, the VPI of any individual leaf node here is 0, so MGSS* stops further computational actions. Examining value of computation for multiple nodes under the same move (such as $\{F, G\}$) does not help in this case. But if we measure $\{D, F, G\}$, then with probability 0.5, D will show utility 9, so B will have utility 9 as well. With probability 0.25, both F and G will show utility 12, so C will have a utility of 11. So with overall probability 0.125, C will be better than

B, and MAX would change the first move to C. The set $\{D, F, G\}$ has a value of computation greater than 0, and should be measured if its cost is sufficiently low. This paper extends VOI to consider such sets.

For any set of nodes to be measured, its value of information minus the cost of the measurement is called the **net value of information**. By extension, we likewise use the terms “net VPI” and “net BVPI” (below) to mean the appropriate type of VOI minus measurement costs.

2.3 Conspiracy Numbers

Conspiracy numbers (McAllester 1988) denote the minimal number of nodes that need to change their values so as to cause the tree’s value of to change to a given value u .

Example A.2

In Figure A.2, the minimax value of the root is 10. In order to increase its value to any $u \in (10, 15]$, the minimal number of leaf nodes that need to change is 1 (node F), thus, there is a single conspirator required. In order to increase the root’s value to $u \in (15, 20]$, the values of both E and F need to change (two conspirators), and three conspirators are required to change it to any $u > 20$.

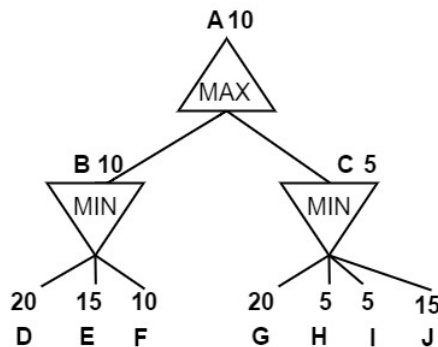


Figure A.2: Conspiracy numbers in MIN-MAX trees

Originally the idea was used to focus game tree search on such conspiracy nodes. Although conspiracy numbers are defined for known leaf values, they are loosely

related to the value of computation (Russell and Wefald 1991a). Generally speaking, a conspiracy number greater than 1 (when true leaf utilities are assumed to be equal to their expected utility as in Example A.1 and Figure A.1) indicates that MGSS* will see a VPI of 0 for every node, whereas the value of computation for numerous nodes may be non-zero (in which case the VPI is not submodular). In this paper we adapt conspiracy numbers in order to find sets of nodes that have a high *probability* of affecting the value at the root.

3 Batch Value of Information

We now define *Batch Value of Perfect Information* (BVPI). The *value of computation* (=VPI) in (Russell and Wefald 1991a) was defined for nodes all under a single child of the root. BVPI is a straightforward generalization of VPI allowing measurements at arbitrary sets S of leaves. Denote by $U_S(v)$ the utility of node v , given that measurements will be performed at a set of leaf nodes S . Note that before getting the actual observed values, $U_S(v)$ is a random variable. Since measurements are assumed to be perfect, $U_S(v)$ (at leaf nodes) is distributed as P_v . Denote the children of v by $ch(v)$, the probability of a child node c of a chance node by $p(c)$, and use appropriate predicates ($LEAF(v)$ is true if v is a leaf node, etc.) to denote node types. The distribution of $U_S(v)$ is defined recursively in Eq. A.4.

$$U_S(v) \sim \begin{cases} P_v & LEAF(v) \wedge v \in S \\ [1 : E_v[P_v]] & LEAF(v) \wedge v \notin S \\ \max_{c \in ch(v)} \{U_S(c)\} & MAXnode(v) \\ \min_{c \in ch(v)} \{U_S(c)\} & MINnode(v) \\ \sum_{c \in ch(v)} p(c) U_S(c) & CHANCEnode(v) \end{cases} \quad (A.4)$$

The *batch value of perfect information* (BVPI) for obtaining perfect information on a set S of nodes is defined as follows. Denote $U_S(\beta) = \max_i \{U_S(\beta_i)\}$, and let $p_{\alpha\beta S}(x, y)$ be the joint probability density function of $(U_S(\alpha), U_S(\beta))$ at (x, y) given the measurements at S . Then:

$$BVPI(S) = \int_{y>x} p_{\alpha\beta S}(x, y)(y - x) dx dy \quad (A.5)$$

Intuitively, $BVPI(S)$ is the gain due to a change of best move from α to some β_i ,

because observations at S revealed that the new utility of β_i is better than that of α . Note that if S is a set of leaves limited to a subtree under α then Eq. A.5 reduces to Eq. A.2. Likewise if S is limited to a subtree under β_i , in which case Eq. A.5 reduces to Eq. A.3.

In using $BVPI(S)$ (Eq. A.5) below, we re-write it as:

$$BVPI(S) = \int_{y>x} p_{\alpha S_\alpha}(x) p_{\beta S_\beta}(y) (y - x) dx dy \quad (\text{A.6})$$

$$= E[\max(U_{S_\beta}(\beta) - U_{S_\alpha}(\alpha), 0)] \quad (\text{A.7})$$

where $p_{\beta S_\beta}$ is the density function of $U_{S_\beta}(\beta)$, which is the same as $U_S(\beta)$, due to subtree independence. The first equality also follows from the subtree independence, and the second from an algebraic manipulation of the terms.

To optimize (batch setting) information gathering using BVPI, one should find a set of nodes S with the highest net $BVPI(S)$. This is hard because: **(1)** We need to compute the BVPI for each such set. **(2)** There is an exponential number of potential subsets S of leaves of T .

3.1 Computing BVPI

Theorem A.1

Computing $BVPI(S)$ for a given set of leaves S in expecti-mini-max trees is NP-hard.

Proof (outline): by reduction from the Partition problem (Garey and Johnson 1979) [SP12], defined as follows. Given a multi set S of integers $\{S_1, S_2, \dots, S_n\}$ (w.l.o.g. $\sum_{i=1}^n S_i$ is even), is there an equal partition, i.e. a set of indices $I \subseteq [1, \dots, n]$ such that $\sum_{i \in I} S_i = \sum_{i \notin I} S_i$?

The reduction uses the three-level expecti-minimax tree of Figure A.3, with two-valued distributions at the leaves. The S_i in the figure are the same numbers as in the partition problem. By computing $BVPI(S)$, with S being the set of all children of the chance node, for 2 different values of u_2 , we can decide the partition problem, as follows. Denote $\sigma = \sum_{i=1}^n \frac{S_i}{2n}$, the desired partition sum divided by n . Before any measurements are made, the expected utility of each uncertain leaf i is $\frac{S_i}{4}$, and thus the expected utility $\bar{U}(v)$ of the chance node is $\frac{\sigma}{2}$, which is less than u_1 and either value of u_2 . So indeed MAX chooses α . If all uncertain leaf nodes are measured, then there is a non-zero probability that $u(v) > u_1$, but if we also have $u(v) > u_2$ then MIN

3. Batch Value of Information

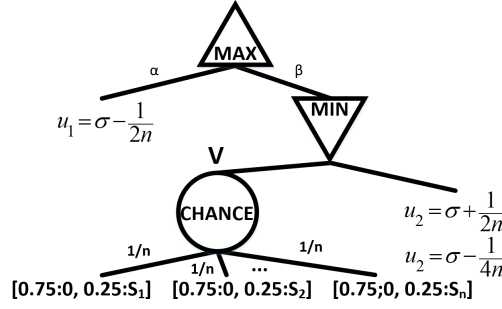


Figure A.3: NP-hardness of computing BVPI: reduction

will not pick the chance node, so the gain in such cases is limited by $u_2 - u_1$. Denote $BVPI(S)$ for the case where $u_2 = \sigma + \frac{1}{2n}$ by B_1 , and for the case where $u_2 = \sigma - \frac{1}{4n}$ by B_2 . In the first case we have:

$$B_1 = P(U_S(v) = \sigma)(\sigma - u_1) + \frac{P(U_S(v) > \sigma)}{n}$$

Since the S_i are all integers, then σ and all possible values of $U_S(v)$ are integer multiples of $\frac{1}{n}$, so $P(U_S(v) \in [u_1, u_1 + \frac{1}{4n}]) = 0$. Thus, the BVPI in the second case is:

$$B_2 = \frac{P(U_S(v) > \sigma - \frac{1}{4n})}{4n} = \frac{P(U_S(v) \geq \sigma)}{4n}$$

If we can compute B_1 and B_2 in polynomial time, we can trivially solve for $P(U_S(v) = \sigma)$, which is non-zero just when the partition problem has a solution. \square

Note that this reduction also implies NP-hardness of VPI (Eq. A.3), as all the uncertain leaves are in the same subtree. However, if the utility of the leaves is bounded, standard *sampling* techniques can be used to approximate $BVPI(S)$. We show a *deterministic* approximation result.

Theorem A.2

Given a game tree T with finite discrete distributions, the value $BVPI(S)$ where S is a subset of the leaves of T can be deterministically approximated within additive error ϵ in time polynomial in the (explicit) description size of T , $\frac{1}{\epsilon}$, and utility span bound $U_{max} - U_{min}$.

Proof (outline): We approximate $U_S(v)$ bottom up, in a manner similar to (Cohen et al. 2015). There, an approximate cumulative distribution (CDF) was computed for MIN-SUM trees, in a manner that bounded the Kolmogorov distance ($\max_x |F'(x) -$

$F(x)|)$ of the approximate CDF from the exact CDF. As the Kolmogorov distance is unhelpful in our case, we use a different version of TRIM that provides the appropriate error bound for the expected values in Eq. A.7.

Denote by $F_{U_S(v)}(x)$ the CDF of $U_S(v)$ at x . For MAX nodes we have: $F_{U_S(v)}(x) = \prod_{c \in \text{ch}(v)} F_{U_S(c)}(x)$ due to independence. Likewise, for MIN nodes we have: $F_{U_S(v)}(x) = 1 - \prod_{c \in \text{ch}(v)} (1 - F_{U_S(c)}(x))$. Chance nodes are weighted sums of the random variables, so we need to use convolution, which may grow the distributions' support exponentially. To overcome this problem, we bound the support by an appropriately defined $n = f(U_{\max} - U_{\min}, |T|, \frac{1}{\epsilon})$ and apply a TRIM operator (see Algorithm 2) to the current $U_S(v)$, assumed to be a list of (value, probability) pairs sorted by increasing value.

Algorithm 2: TRIM left operator

```

1   $m \leftarrow |\text{support}(U_S(v))|;$        $U'_S(v) \leftarrow \emptyset;$ 
2   $\text{head} \leftarrow \text{first}(U_S(v));$        $\text{tail} \leftarrow \text{rest}(U_S(v));$ 
3  while  $m > n$  and  $\text{tail}$  is non-empty do
4       $\text{next} \leftarrow \text{first}(\text{tail});$ 
5      if  $\text{value}(\text{next}) - \text{value}(\text{head}) < \frac{U_{\max} - U_{\min}}{n}$  then
6           $\text{prob}(\text{head}) \leftarrow \text{prob}(\text{head}) + \text{prob}(\text{next});$ 
7           $\text{tail} \leftarrow \text{rest}(\text{tail});$        $m \leftarrow m - 1;$ 
8      else
9           $U'_S(v) \leftarrow \text{append}(U'_S(v), \text{head});$ 
10          $\text{head} \leftarrow \text{first}(\text{tail});$        $\text{tail} \leftarrow \text{rest}(\text{tail});$ 
11 return  $\text{append}(U'_S(v), \text{tail});$ 

```

By construction, $|\text{support}(U'_S(v))| \leq n$ after TRIM. The error introduced by TRIM is bounded by $\delta = \frac{U_{\max} - U_{\min}}{n}$, i.e.: $F_{U_S(v)}(x + \delta) \geq F_{U'_S(v)}(x) \geq F_{U_S(v)}(x)$ for all x .

We prove lemmas bounding combination errors: in MIN and MAX nodes errors are added, and in CHANCE nodes at worst equal to the maximum error in the children. The (low-order polynomial) function f can be chosen so that $F_{U_S(r)}(x + \epsilon) \geq F_{U'_S(r)}(x) \geq F_{U_S(r)}(x)$ for all x at the root r . These inequalities imply that the Wasserstein distance between $U_S(r)$ and $U'_S(r)$ is at most ϵ , thus $|E[U'_S(r)] - E[U_S(r)]| \leq \epsilon$. Since $|\text{support}(U'_S(v))| \leq n$, for all v , we can then compute the expectations in Eq. A.7 in polynomial time. \square

Note that we can also handle continuous distributions, by discretizing them into $\frac{U_{max}-U_{min}}{n}$ values, achieving similar approximation guarantees, assuming that we can efficiently compute their CDFs at any given point.

4 Practical Batch-selection

We now examine practical methods for choosing batches of nodes with a high net BVPI. Here, we treat this as a stand-alone problem on a partially developed tree T . In Section 5 we incorporate these methods into MCTS.

Optimizing the value of information was shown to be NP-hard even for one-level trees (see, e.g. (Reches et al. 2013; Shperberg and Shimony 2017)). Thus we are forced to introduce methods suggested by the theory, but which have no guarantees, in order to be able to meet the extremely difficult task of actually improving the runtime or quality of MCTS, which requires that we find node-sets with high BVPI in essentially negligible computation time.

4.1 High-BVPI sets: Using Conspiracies

This scheme uses an idea based on the above described conspiracy numbers (McAllester 1988); it is possible to quickly detect node-sets involved in minimal conspiracies. We desire instead a probabilistic version that can quickly detect node-sets S that have a significant contribution to Eq. A.6, and hence should have a high net $BVPI(S)$. Note that we do not need to find even a nearly optimal set, it is sufficient for our search application to find a reasonably good set, and not to return an empty set when sets with a high net BVPI are available.

We adapt the conspiracy scheme by defining a probabilistic variant of conspiracy numbers, and then by evaluating a modified version of Eq. A.6 where we replace integration by maximization, using as S the entire set of leaf nodes, and $U_S(v)$ as in Eq. A.4. In MIN-MAX trees, the probability that the value of node v will increase to at least Val if we measure nodes S is: $\uparrow \phi(v, Val) = P(U_S(v) \geq Val)$. Likewise the value v will decrease to be at most Val with probability: $\downarrow \phi(v, Val) = P(U_S(v) \leq Val)$. Now perform the following optimization (note the similarities with Eq. A.6), as done within Algorithm 3.

Algorithm 3: C-VIBES Selection Scheme

```

1 function SelectNodes(root):
2   foreach  $V \in \mathcal{V}$  do
3     storeProbabilities( $\alpha, V$ )
4     foreach Node  $c \in ch(\text{root}) - \{\alpha\}$  do
5       storeProbabilities( $c, V$ )
6    $V, V', c \leftarrow$  values which optimize Equation A.8
7   OpenList.init( $\alpha$ ); OpenList.insert( $c$ )
8   while OpenList not empty do
9      $v \leftarrow$  OpenList.pop()
10    if  $v$  is a fringe node then
11       $S \leftarrow S \cup \{v\}$ 
12    else
13       $Val \leftarrow \begin{cases} V & \text{if } v \in \text{subtree of } \alpha \\ V' & \text{otherwise} \end{cases}$ 
14      foreach  $c \in ch(v)$  do
15        if  $\phi[v, Val] \notin \{0, 1\}$  and (CHANCEnode( $v$ ) or
16           $\phi[v, Val] == \phi[c, Val]$ ) then
17            OpenList.insert( $c$ )
18  return  $S$ 

17 function storeProbabilities( $v, Val$ ):
18   foreach  $c \in ch(v)$  do
19     storeProbabilities( $c, Val$ )
20    $\phi[v, Val] \leftarrow \begin{cases} \hat{P}(U_S(v) \leq Val) & \text{if } v \in \text{subtree of } \alpha \\ \hat{P}(U_S(v) \geq Val) & \text{otherwise} \end{cases}$ 

```

$$\max_{V' > V} ((V' - V)(\downarrow \phi(\alpha, V) \max_{c \in ch(r) - \{\alpha\}} \uparrow \phi(c, V'))) \quad (\text{A.8})$$

The \hat{P} in the algorithm are probabilities approximated as in Theorem A.2, except

that for CHANCE nodes, our BVPI approximation is still too slow for the desired real-time performance, so we use: $\hat{P}(U_S(v) \leq Val) \approx \sum_{c \in ch(v)} p(c)P(U_S(c) \leq Val)$ instead of convolution in our implementation of the conspiracy scheme. Example A.3 below depicts one such node-set recovery instance.

Batch Selection Algorithms

We evaluate three batch selection algorithms:

(1) Full tree (FT): Exhaustively compute net BVPI (using BVPI approximation algorithm from Theorem A.2) for every subset S of T 's leaves; pick S with the highest net BVPI.

(2) Greedy (G): Start S as an empty set. Estimate the net VPI for every leaf node not in S , and add the best to S . Repeat until no node has a positive net VPI.

(3) Conspiracy (C): The conspiracy-based scheme described above. As the optimization in Eq. A.8 is still too slow to perform in real time for MCTS, our implementation optimizes over only a few possible values of V, V' in Eq. A.8. In the experiments we used the value set:

$$\mathcal{V} = \{0.8\bar{U}(\alpha), 0.9\bar{U}(\alpha), 0.95\bar{U}(\alpha), \bar{U}(\alpha), 1.05\bar{U}(\alpha)\}$$

Example A.3

In figure A.1, α is the current best move, since $\bar{U}(\alpha) = 12 > 9 = \bar{U}(\beta)$. The leaf node set $S = \{D, F, G\}$ is the only one to potentially change the best action from α to β , thus the only set with $BVPI(S) > 0$. FT estimates the BVPI exhaustively, and thus will correctly return S . The Greedy scheme suffers myopic assumptions like MGSS*. Recall that in this figure, the VPI of every individual node is 0, hence, Greedy will terminate without finding S and return an empty set. The Conspiracy scheme optimizes Equation A.8 in order to find a batch to sample. The optimal solution is achieved with $V' = 11$ and $V = 9$. Using these values, $\downarrow \phi(\alpha, V = 9) = 0.5$, obtained by picking D , and $\uparrow \phi(\beta, V' = 11) = 0.5$ obtained by picking both F and G . Therefore, Conspiracy returns $\{D, F, G\}$ as desired. Note that the values $V' = 11$ and $V = 9$ are outside the range \mathcal{V} used by our actual Conspiracy implementation for optimizing Equation A.8. Despite that, using $V = 0.8\bar{U}(\alpha) = 9.6$

and $V' = 0.9\bar{U}(\alpha) = 10.8$, the implementation still finds and returns the correct node-set $S = \{D, F, G\}$.

4.2 Experiments on Game Trees

CTP										StarCraft									
#	Tree Size	Tree Height	Max BF	Full Tree		Conspiracy		Greedy		#	Tree Size	Tree Height	Max BF	Full Tree		Conspiracy		Greedy	
				BVPI	T (ms)	BVPI	T (ms)	BVPI	T (ms)					BVPI	T (ms)	BVPI	T (ms)	BVPI	T (ms)
1	30	9	3	359	217	296	0.28	92	0.28	11	30	4	20	283	198	234	0.25	205	0.24
2	45	13	4	453	443	382	0.29	215	0.28	12	45	4	20	356	413	356	0.25	356	0.25
3	61	15	3	126	1868	103	0.29	19	0.29	13	60	6	20	326	1719	312	0.28	283	0.28
4	106	25	4	527	356872	480	0.34	319	0.34	14	100	11	20	147	299814	147	0.33	105	0.31
5	125	20	6	812	7129870	681	0.36	681	0.35	15	125	11	20	63	7088815	63	0.36	63	0.36
6	153	42	5	219	20676623	219	0.4	71	0.38	16	150	12	20	96	19938631	71	0.39	0	0.38
7	1018	33	6	T/O	T/O	113	0.78	0	0.73	17	1000	23	20	T/O	T/O	33	0.72	33	0.70
8	3077	29	6	T/O	T/O	65	1.89	13	1.88	18	3000	28	20	T/O	T/O	116	1.85	42	1.82
9	6820	37	5	T/O	T/O	53	4.2	0	3.9	19	7000	42	20	T/O	T/O	25	4.33	7	4.01
10	15321	69	7	T/O	T/O	277	8.5	12	8	20	15000	79	20	T/O	T/O	47	8.41	0	7.96

Table A.1: BVPI in Trees Generated by UCTO in the CTP (left) and in StarCraft (right)

To get a realistic game tree, we used a snapshot of a partially developed game tree T' from a MCTS. T' is cut off so that its fringe nodes become the leaves of our tree T . Values previously returned by rollouts form an empirical distribution at each fringe node (now leaf) v ; this distribution is assumed to be the actual distribution P_v of leaf node utilities. E.g. if we had 2 rollouts with value 10, and 6 rollouts with value 20, (from v) then we set $P_v = [0.25 : 10, 0.75 : 20]$. Measurement costs assumed are given by Equation A.9 (Sec. 5). Trees were from 2 domains.

In the (stochastic) Canadian traveler problem (Papadimitriou and Yannakakis 1991) we are given a weighted graph $G = (V, E, w)$ where $w : E \rightarrow R^+$. Each edge $e \in E$, has a known probability $p(e)$ of being blocked. The agent starts at vertex $s \in V$, and must reach a vertex $t \in V$. Whether an edge is blocked becomes known upon reaching an incident vertex. The problem is to find a policy that minimizes the expected travel distance (sum of w) before reaching t (the utility here is minus the distance). The decision version of the stochastic CTP is PSPACE complete (Fried et al. 2013). (Eyerich et al. 2010) present an effective UCT-based algorithm called UCTO which randomly searches the belief states in the given program instance, and generates EXPECTI-MAX trees.

StarCraft is a Real Time Strategy (RTS) game by Blizzard Entertainment, a popular AI competition and research platform. Our StarCraft experiments used code by (Justesen et al. 2014) that attempts to optimize StarCraft battles against an opposing team. Their code uses a UCT-based MCTS algorithm that generates MAX-MIN trees during the search.

Approximate net-BVPI and CPU time for 10 instances of appear in Table A.1(left) for CTP and in Table A.1(right) for StarCraft. In both domains, FT timed out (T/O denotes timing out after 6 hours) in large instances. Greedy is the fastest, but the BVPI it achieved was only 64% of that of FT on average (over instances where FT did not time out). Conspiracy had the best balance between time and effectiveness. Its BVPI was 89% of that of FT on average, and was only slightly slower (2%) than Greedy. In fact, the conspiracy scheme was also the best in the MCTS below.

5 Plugging BVPI into MCTS

The next challenge is to use the theory of BVPI for selecting nodes on which rollouts will be performed.

5.1 Selecting Rollouts

We describe a number of BVPI-based selection schemes and evaluate them as well as other related schemes by plugging them into existing UCT based implementations (denoted as “the host MCTS algorithm”) by changing only the selection phase. Each scheme has a different tradeoff between metareasoning overhead (time for choosing the nodes to sample), and the effectiveness of the resulting rollouts. The first scheme we implemented is the **Blinkered** scheme from (Hay et al. 2012), which replaces the UCT criterion by a VOI bound in the first tree level, resorting to UCT at deeper levels.

We now introduce our BVPI-based schemes, called *Value of Information of a Batch Efficient Selection* (VIBES). The metareasoning overhead in VIBES is relatively high, thus too expensive to use to decide every single rollout. Therefore, after a node v is selected for sampling we perform N rollouts from v . This amortizes the metareasoning overhead over N rollouts. Additionally, some of the schemes below first select a batch S of fringe nodes to sample. This set can be of any size (up to the number

of leaf nodes). In order not to over-commit a large number of rollouts in such cases, we choose K nodes from S with the highest individual net VPI, and an additional K randomly selected from S to a total of at most $2K$ nodes for rollouts. The additional K random samples are used to allow measurements on nodes with $VPI = 0$ inside the selected batch. Below we call this *the batch selection method*, $BSM(N, K)$. Values of $N = 3$, $K = 5$ proved to produce a good balance. That is, K was set to be roughly one quarter of the size of a typical set S observed in a few trial runs. Then N was set such that $2KN$ rollouts per decision delivered a metareasoning overhead of roughly 15% of the runtime.

(1) Full tree VIBES (FT-VIBES): Exhaustively check all possible sets of fringe nodes in the tree, to find a set S with the highest net $BVPI(S)$. Then use $BSM(N, K)$ on S .

(2) First level VIBES (FL-VIBES): Estimate the net BVPI for every subset of the root's children. Choose the subset S which maximizes net $BVPI(S)$. Within S , choose a child c with the maximal individual net VPI. At deeper levels, revert to selection as in the host MCTS algorithm (e.g., the UCT formula if host=UCTO).

(3) Recursive first level VIBES (RFL-VIBES): Use FL-VIBES to select node c at the first level. Then, recursively call FL-VIBES on c until reaching a fringe node.

(4) Blinkered VIBES (B-VIBES): Perform the blinkered algorithm (Hay et al. 2012) until blinkered decides to halt. Then, resort to performing FT-VIBES.

(5) Conspiracy VIBES (C-VIBES):

Select the node-set S using the Conspiracy scheme (Algorithm 3). Then use $BSM(N, K)$ on S .

In all the above methods: (a) generate some random samples to gather statistics before applying VIBES (typically 1% of the sampling budget), (b) stop the entire sampling process if the sampling budget (number of rollouts per move, or a time limit per move) is exhausted or if the scheme did not find a set S with positive net BVPI.

We also tried greedy schemes. The *basic greedy* scheme (G) repeatedly chooses the node v with the greatest individual net VPI and performs rollouts from v . This halts when the net VPI of v is non-positive or the simulation budget has been reached. This method is thus essentially the same as the Meta-Greedy Single Step (MGSS*) method of (Russell and Wefald 1991a) applied to MCTS.

Basic greedy is fast and performs well when the BVPI is submodular. Otherwise,

5. Plugging BVPI into MCTS

it suffers from premature stopping, i.e. fails to detect sets of nodes with a high combined BVPI. In order to take advantage of the speed of the greedy scheme but avoid premature stopping, we can combine it with any of the above schemes (X), as follows. Run the greedy scheme. Once the greedy scheme decides to stop due to low VPI, revert to scheme X to decide on any additional samples as long as the budget allows. We denote these combined schemes by G-X (e.g, G-FT-VIBES).

Finally, we need to address the issue of fringe node distributions used in the BVPI, and the resulting distributions after potential additional rollouts. Ideally, use a Bayesian scheme for the distributions: have some prior, and compute distributions given past rollouts, and distributions over the conditional expectation of fringe node utilities given additional potential rollouts. This issue is beyond the scope of this paper, and even if somehow this is done, it would not be obvious how to do so in real time. Instead we performed the following. We assumed that the fringe node distributions P_v are the empirical utility distributions of past rollouts, which is the reason for the initial “statistics gathering” rollouts mentioned above. We then compute the BVPI values as if rollouts from a node result in a perfect measurement of the fringe node utility. As the latter assumption is obviously incorrect, we compensated by modifying the measurement cost of a node. (Note that our BVPI schemes are not sensitive to multiplying all costs and VOI values by a constant, so this compensation makes sense.) Although the latter is somewhat of a hack, it works in practice in a way that is not too sensitive to tunable parameters.

Thus, in the above schemes, as the measurement cost $C(v)$ in the computation of the net BVPI we used:

$$C(v) = \frac{C \sqrt{N(v)}}{B - \bar{U}(\alpha)} \quad (\text{A.9})$$

with $N(v)$ as in Eq. A.1, B the maximal utility bound on the solution. The rationale is: if the $N(v)$ (=number of previous rollouts) is a large number, it will require more future rollouts to change the overall average of all the rollout values by a significant amount (on the order of $B - \bar{U}(\alpha)$). C is empirically determined over a few instances. Results were not very sensitive to C , we used $C = 96$ in our experiments. A more disciplined treatment of the costs is a non-trivial issue beyond the scope of this paper, as the computation time is not on the same scale as the game utilities.

Settings		UCTO	Blink'd	FL-VIBES	RFL-VIBES	FT-VIBES	B-VIBES	C-VIBES	Greedy	G-FL-VIBES	G-RFL-VIBES	G-FT-VIBES	G-C-VIBES
n	p												
30	0.2	3,006	2,911	2,989	2,830	3,625	2,785	2,598	2,902	2,709	2,668	2,973	2,747
	0.4	3,600	3,490	3,564	3,397	4,356	3,336	3,130	3,473	3,253	3,224	3,563	3,294
	0.6	4,198	4,070	4,172	3,959	5,095	3,901	3,630	4,098	3,804	3,758	4,149	3,829
	0.8	4,801	4,646	4,768	4,510	5,829	4,469	4,165	4,599	4,344	4,298	4,742	4,388
	1	5,401	5,241	5,378	5,099	6,521	5,028	4,675	5,226	4,893	4,838	5,346	4,933
	Avg. of n=30	4,201	4,072	4,174	3,959	5,085	3,904	3,640	4,060	3,801	3,757	4,155	3,838
60	0.2	5,049	4,836	4,870	4,804	6,157	4,753	4,340	4,842	4,590	4,480	5,091	4,495
	0.4	7,074	6,775	6,805	6,735	8,611	6,665	6,095	6,762	6,412	6,252	7,140	6,309
	0.6	8,572	8,237	8,280	8,179	10,465	8,084	7,396	8,268	7,745	7,568	8,630	7,651
	0.8	10,082	9,704	9,698	9,603	12,335	9,501	8,696	9,667	9,175	8,915	10,224	9,030
	1	11,572	11,091	11,213	11,049	14,179	10,937	9,968	11,102	10,534	10,266	11,708	10,366
	Avg. of n=60	8,470	8,129	8,173	8,074	10,349	7,988	7,299	8,128	7,691	7,496	8,559	7,570
100	0.2	7,506	7,277	7,392	7,210	8,929	6,965	6,240	7,299	6,961	6,912	7,273	6,668
	0.4	10,966	10,614	10,763	10,546	13,005	10,160	9,109	10,581	10,151	10,139	10,488	9,753
	0.6	13,459	13,032	13,232	12,932	15,961	12,450	11,162	13,096	12,424	12,422	12,810	11,957
	0.8	15,916	15,469	15,634	15,264	18,923	14,740	13,226	15,490	14,683	14,634	15,157	14,089
	1	18,346	17,886	18,076	17,702	21,851	17,066	15,273	17,787	17,005	17,014	17,461	16,284
	Avg. of n=100	13,239	12,855	13,019	12,731	15,734	12,276	11,002	12,851	12,245	12,224	12,638	11,750

Table A.2: Average results for CTP. The standard deviation was at most 0.36%.

6 Empirical Evaluation: MCTS

We now report results where the selection schemes were plugged into existing MCTS algorithms for two domains.

6.1 Canadian Traveler Problem (CTP)

CTP experiments were on Delaunay graphs following (Bnaya et al. 2009; Eyerich et al. 2010). Instance parameters were n (number of vertices), and p (probability for each edge to be potentially blocked). For each potentially blocked edge e , $p(e)$ is chosen uniformly from the range $[0, 1)$. Edge travel costs were random, uniform from $\{1, \dots, 500\}$.

We compared the original UCTO code (Eyerich et al. 2010), with the same code where the node-selection function (a UCT formula variant) was replaced by one of the above described methods. All the algorithms were evaluated as follows. 10 instances were generated for each pair of $n \in \{30, 60, 100\}$ and $p \in \{0.2, 0.4, 0.6, 0.8, 1\}$. We ran every instance 100 times using a time limit of 3 seconds to decide on each move.

Table A.2 shows the average cost of the path traveled by the agent for the different

6. Empirical Evaluation: MCTS

Selection alg.	Cost	Subopt.	#RollOuts/Move
Optimal (VI)	2,572	0%	N/A
UCTO	3,128	21.6%	12,371
Blinkered	3,021	17.5%	11,016
FL-VIBES	3,096	20.4%	4,284
RFL-VIBES	2,931	14.0%	1,576
FT-VIBES	3,778	46.9%	43
B-VIBES	2,890	12.4%	745
C-VIBES	2,699	4.5%	9,989
Greedy	2,989	16.2%	11,904
G-FL-VIBES	2,823	9.8%	10,978
G-RFL-VIBES	2,784	8.2%	8,732
G-FT-VIBES	3,084	19.9%	611
G-C-VIBES	2,847	10.7%	10,633

Table A.3: 30 node graphs, $p=0.2$, time limit 3 sec/move

algorithms. As can be seen, all the proposed schemes outperformed UCTO, except for FT-VIBES (see below). C-VIBES delivered a good rollout selection at relatively little overhead, thus the best overall performance: a major improvement of more than 15% in the **path-cost** of the resulting solutions over UCTO.

In order to gain additional insight, we chose a typical instance where computing the optimal policy by value iteration (VI) over the belief space was feasible. We examined the runtime and quality of the results vs. the optimal under two budget constraints. **(Const-1:)** a time limit of 3 sec/move (as in Table A.2); results appear in Table A.3. Here, FT-VIBES timed out after only a few samples. Conspiracy-VIBES was the best here (4.5% from optimal), as it produces good sampling decisions with only a modest overhead. **(Const-2:)** a rollout limit (but no time limit). Runtime vs. path-costs results appear in Figure A.4 for a 10K rollout limit for all algorithms. Additional runs with different limits for UCTO and C-VIBES are also shown, labeled as “C-VIBES-3K” (limit 3,000 rollouts per move) etc. C-VIBES-3K already gets better path-cost than UCTO-45K and runs 10 times faster. Furthermore C-VIBES-20K, seems to have found the optimal policy for this instance in many of the runs.

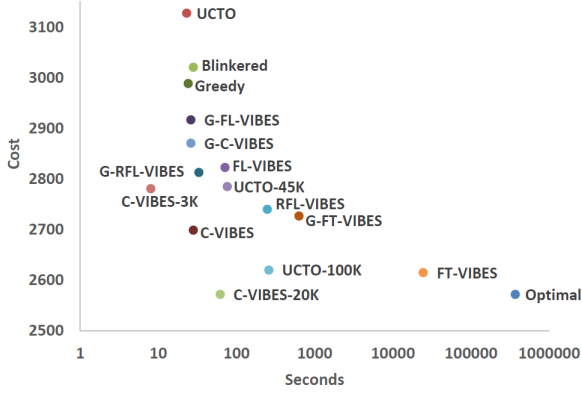


Figure A.4: 30 node graphs, $p=0.2$, w. 10,000 rollouts/move

It is also of interest that among all algorithms run with 10,000 samples, FT-VIBES was the best w.r.t. policy quality, as expected. So the BVPI scheme done exhaustively indeed resulted in the most efficient selection of nodes and samples if the meta-reasoning overhead is ignored. However, this scheme is seen here to be completely useless for MCTS due to its huge meta-reasoning overhead. Another interesting point is that for 10,000 samples the greedy scheme was already better than UCT, but only slightly. We conjecture that this is due to cases where the BVPI is not submodular. This is supported by the results for the greedy hybrids, which select further nodes after greedy decides that no nodes should be selected. All the greedy hybrids do better than greedy w.r.t. policy quality, usually with negligible additional overhead.

6.2 StarCraft Battles

Our experiments used JarCraft, an open-source java StarCraft combat simulator as well as the search algorithms from (Justesen et al. 2014). These algorithms use UCT on different action spaces as follows:

- (1) **UCTCD**: a UCT variant which handles simultaneous and durative actions. Possible actions are sets of unit commands. (Churchill and Buro 2013).
- (2) **Script-based UCTCD**: an extension of UCTCD allowing both unit commands and scripts as possible actions (Justesen et al. 2014). Also presented there were (3), (4):
- Cluster-based UCTCD**: two improvements of UCTCD that cluster units in order to decrease the number of possible actions.

We modified these UCT-based algorithms, replacing their node-selection by Blinkered, Greedy, and C-VIBES. The test scenario is based on (Justesen et al. 2014): each

7. Conclusion

competing algorithm in each run controls $\frac{n}{2}$ Protoss Zealots (close combat unit) and $\frac{n}{2}$ Protoss Dragoons (ranged combat unit) vs. an opposing team of equal size controlled by another algorithm. The units are first lined up by type and then scattered randomly. For each army size n we chose the UCT variant with the best performance according to (Justesen et al. 2014).

Representative results appear in Figure A.5 where Blinkered, Greedy, and C-VIBES competed against UCT. C-VIBES was the best, averaging a 77% win rate against UCT (see Figure A.5), though both Blinkered and Greedy were also consistently better than UCT. C-VIBES also defeated Greedy and Blinkered in head-to-head matches (not shown).

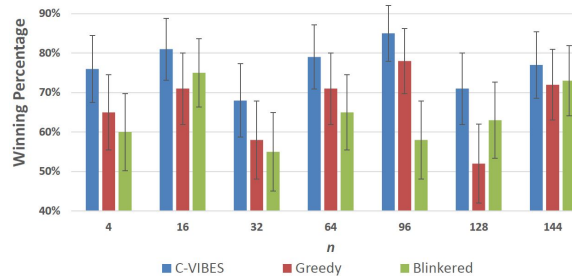


Figure A.5: Win rates vs. UCT (100 games). Error bars are 95% confidence intervals.

7 Conclusion

Optimizing VOI for individual nodes has been shown to improve allocation of rollouts (Hay et al. 2012; Feldman and Domshlak 2013), as well as backups (Feldman and Domshlak 2013) in MCTS. Estimating the VOI for individual nodes is too limiting. We suggested a method based on value of computation that considers large batches, and suggested several effective ways to approximately optimize them. While we confirmed that previously suggested VOI methods (MGSS* and blinkered) outperform UCT, our BVPI-based selection schemes plugged into MCTS implementations outperformed them all.

Although our measurements were assumed to be rollouts in a MCTS, the analysis in Section 3 and the batch selection methods in Section 4 may be applicable to other information-gathering operations; such as computing a static heuristic evaluation function at v , expanding v , or even real-world physical measurements (whenever

the latter is meaningful).

Future improvements to our schemes are possible. First, a more disciplined way to estimate the cost of computations (e.g. by learning) would be beneficial. Second, a better defined distribution over fringe node utilities given the future rollouts is desired, such as through Bayesian updating, or estimation methods from (Feldman and Domshlak 2013), which is essentially orthogonal to our paper, can be attempted.

Acknowledgments

Supported by ISF grant 417/13, and by the Frankel Center. We thank the authors of (Justesen et al. 2014; Eyerich et al. 2010) for providing their code.

References of Paper A

- [BFS09] Zahy Bnaya, Ariel Felner, and Solomon Eyal Shimony. “Canadian traveler problem with remote sensing”. In: *Proc. of IJCAI*. Pasadena, California, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 437–442.
- [Bro+12] Cameron Browne et al. “A Survey of Monte Carlo Tree Search Methods”. In: *IEEE Trans. Comput. Intellig. and AI in Games* 4.1 (2012), pp. 1–43. doi: 10.1109/TCIAIG.2012.2186810. URL: <http://dx.doi.org/10.1109/TCIAIG.2012.2186810>.
- [CB13] David Churchill and Michael Buro. “Portfolio greedy search and simulation for large-scale combat in StarCraft”. In: *2013 IEEE Conference on Computational Intelligence in Games (CIG), Niagara Falls, ON, Canada, August 11-13, 2013*. 2013, pp. 1–8. doi: 10.1109/CIG.2013.6633643. URL: <http://dx.doi.org/10.1109/CIG.2013.6633643>.
- [CSW15] Liat Cohen, Solomon Eyal Shimony, and Gera Weiss. “Estimating the Probability of Meeting a Deadline in Hierarchical Plans”. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael Wooldridge. AAAI Press, 2015, pp. 1551–1557. ISBN: 978-1-57735-738-4. URL: <http://ijcai.org/Abstract/15/222>.
- [EKH10] Patrick Eyerich, Thomas Keller, and Malte Helmert. “High-quality policies for the Canadian traveler’s problem”. In: *In Proc. AAAI 2010*. Ed. by Maria Fox and David Poole. AAAI Press, 2010, pp. 51–58. URL: <http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1735>.
- [FD13] Zohar Feldman and Carmel Domshlak. “Monte-Carlo Planning: Theoretically Fast Convergence Meets Practical Efficiency”. In: *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. Ed. by Ann Nicholson and Padhraic Smyth. AUAI Press, 2013. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2382&proceeding_id=29.

- [Fri+13] Dror Fried et al. “Complexity of Canadian traveler problem variants”. In: *Theor. Comput. Sci.* 487 (2013), pp. 1–16. DOI: 10.1016/j.tcs.2013.03.016. URL: <http://dx.doi.org/10.1016/j.tcs.2013.03.016>.
- [GJ79] M. R. Garey and D. S. Johnson. “Computers and Intractability, A Guide to the Theory of NP-completeness”. In: W. H. Freeman and Co., 1979, p. 190.
- [Hay+12] Nicholas Hay et al. “Selecting Computations: Theory and Applications”. In: *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*. Ed. by Nando de Freitas and Kevin P. Murphy. AUAI Press, 2012, pp. 346–355. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2297&proceeding_id=28.
- [Jus+14] Niels Justesen et al. “Script- and cluster-based UCT for StarCraft”. In: *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26-29, 2014*. 2014, pp. 1–8. DOI: 10.1109/CIG.2014.6932900. URL: <http://dx.doi.org/10.1109/CIG.2014.6932900>.
- [KG09] Andreas Krause and Carlos Guestrin. “Optimal Value of Information in Graphical Models”. In: *J. Artif. Intell. Res. (JAIR)* 35 (2009), pp. 557–591. DOI: 10.1613/jair.2737. URL: <http://dx.doi.org/10.1613/jair.2737>.
- [KG11] Andreas Krause and Carlos Guestrin. “Submodularity and its applications in optimized information gathering”. In: *ACM TIST* 2.4 (2011), p. 32. DOI: 10.1145/1989734.1989736. URL: <http://doi.acm.org/10.1145/1989734.1989736>.
- [KS06] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *ECML*. 2006, pp. 282–293.
- [McA88] David Allen McAllester. “Conspiracy numbers for min-max search”. In: *Artificial Intelligence* 35.3 (1988), pp. 287–310. ISSN: 0004-3702. DOI: [http://dx.doi.org/10.1016/0004-3702\(88\)90019-7](http://dx.doi.org/10.1016/0004-3702(88)90019-7). URL: <http://www.sciencedirect.com/science/article/pii/0004370288900197>.

References of Paper A

- [PF12] Georgios Papachristoudis and John W. Fisher III. “Theoretical Guarantees on Penalized Information Gathering”. In: *Statistical Signal Processing Workshop (SSP)*. Aug. 2012, pp. 301–304. URL: <http://people.csail.mit.edu/geopapa/pubs/geopapaSSP2012.pdf>.
- [PY91] Christos H. Papadimitriou and Mihalis Yannakakis. “Shortest Paths Without a Map”. In: *Theor. Comput. Sci.* 84.1 (1991), pp. 127–150.
- [RGK13] Shulamit Reches, Ya’akov (Kobi) Gal, and Sarit Kraus. “Efficiently gathering information in costly domains”. In: *Decision Support Systems* 55.1 (2013), pp. 326–335. DOI: 10.1016/j.dss.2013.01.021. URL: <http://dx.doi.org/10.1016/j.dss.2013.01.021>.
- [RW91a] Stuart J. Russell and Eric Wefald. *Do the right thing - studies in limited rationality*. MIT Press, 1991. ISBN: 978-0-262-18144-0.
- [RW91b] Stuart J. Russell and Eric Wefald. “Principles of Metareasoning”. In: *Artif. Intell.* 49.1-3 (1991), pp. 361–395. DOI: 10.1016/0004-3702(91)90015-C. URL: [http://dx.doi.org/10.1016/0004-3702\(91\)90015-C](http://dx.doi.org/10.1016/0004-3702(91)90015-C).
- [SS17] Shahaf S. Shperberg and Solomon Eyal Shimony. “Some Properties of Batch Value of Information in the Selection Problem”. In: *J. Artif. Intell. Res.* 58 (2017), pp. 777–796. DOI: 10.1613/jair.5288. URL: <https://doi.org/10.1613/jair.5288>.

Paper B

Some Properties of Batch Value of Information in the Selection Problem

Shahaf S. Shperberg, Solomon Eyal Shimony

The paper has been published in the
Journal of Artificial Intelligence Research
volume 58, pages 777–796, 2017.

© 2017 JAIR

The layout has been revised.

Abstract

Given a set of items of unknown utility, we need to select one with a utility as high as possible (“the selection problem”). Measurements (possibly noisy) of item values prior to selection are allowed, at a known cost. The goal is to optimize the overall sequential decision process of measurements and selection.

Value of information (VOI) is a well-known scheme for selecting measurements, but the intractability of the problem typically leads to using myopic VOI estimates. Other schemes have also been proposed, some with approximation guarantees, based on submodularity criteria. However, it was observed that the VOI is not submodular in general. In this paper we examine theoretical properties of VOI for the selection problem, and identify cases of submodularity and supermodularity. We suggest how to use these properties to compute approximately optimal measurement batch policies, with an example based on a “wine selection problem”.

1 Introduction

Decision-making under uncertainty is a domain with numerous important applications. Since these problems are intractable in general, special cases are of interest. In this paper, we examine the selection problem: given a set of items of unknown utility (but drawn from a known distribution), we need to select an item with as high a utility as possible. Measurements (possibly noisy) of item values prior to selection are allowed, at a known cost. The problem is to optimize the overall decision process of measurement and selection. Even with the severe restrictions imposed by the above setting, this decision problem is intractable (Tolpin and Shimony 2012; Radovilsky, Shattah, et al. 2006; Radovilsky and Shimony 2008); and yet it is important to be able to solve, at least approximately, as it has numerous potential applications. This paper analyses cases where the value of information (VOI) is submodular, which is a sufficient condition for achieving good approximate solutions to the selection problem with appropriate greedy algorithms.

Settings where the selection problem is applicable are in meta-reasoning, i.e. considering which time-consuming deliberation steps to perform before selecting an action (Russell and Wefald 1991b; Russell and Wefald 1991a; Hay et al. 2012), as well

as settings where the items to be selected are physical objects. Examples of the latter type are: oil exploration, locating a point of high temperature using a limited number of measurements (Krause and Guestrin 2009), performing costly measurements in order to find the best time to hit a target when the system is modeled using a stochastic system estimator, such as a Kalman filter, and a good set of parameters for setting up an industrial imaging system (Tolpin and Shimony 2012). The selection problem can also be seen as a special case of Bayesian optimization, and can be used to select (from a large batch of candidates) experiments to be performed (Azimi et al. 2016).

In most of the above applications the item values are naturally represented as being dependent. A potential application used in this paper as a running example (see Section 3) is selecting one from a set of wine cases that have uncertain qualities. Similar applications are selecting a batch of applicants with unrelated backgrounds to interview (from a larger set of job applicants) before making a hiring decision, and selecting a set of apartments from disparate locations to visit (among the available listings) before making a rental or purchasing decision. In the latter type of problems, the item distributions are either truly independent, or independent in a practical sense: it is not possible or worthwhile to obtain statistics beyond individual marginal distributions. (Note that such independence assumptions may be unjustified with job applicants that have a similar background, or apartments in the same building.) In fact, in meta-reasoning in search an equivalent independence assumption called “subtree independence” is commonly made (Russell and Wefald 1991b), even though it does not truly hold in the underlying search domains. Likewise, for oil exploration one could make such an independence assumption as a reasonable first-order approximation if one is considering as items a set of disjoint oil fields that are not physically near each other.

The selection problem is also called a Bayesian ranking and selection problem (Raiffa and Schlaifer 2000; Frazier 2012; Swisher et al. 2003), where in (Frazier 2012) the measurements are usually assumed to be noisy samples of the utility value of the items. A widely adopted scheme for selecting measurements (also called sensing actions in some contexts, or deliberation steps in the context of meta-reasoning) is based on value of information (VOI) (Russell and Wefald 1991b; Russell and Wefald 1991a). Optimizing value of information is intractable in general, thus both researches and

1. Introduction

practitioners often use various forms of myopic VOI estimates (Russell and Wefald 1991b; Russell and Wefald 1991a; Hay et al. 2012) coupled with greedy search. The properties of VOI have been of interest to the research community for quite a while (Raiffa and Schlaifer 2000).

In particular, (Delara; Radner and Stiglitz 1984; Chadeand and Schlee 2001) examine conditions for nonconcavity in the value of information, a notion akin to non-submodularity in a continuous setting. Submodularity is an important property, because in cases where the VOI is submodular, simple, greedy algorithms result in provably near-optimal policies (Krause and Guestrin 2009; Krause and Guestrin 2011; Papachristoudis and Fisher III 2012). However, as stated in (Delara; Radner and Stiglitz 1984; Chadeand and Schlee 2001; Krause and Guestrin 2009), the VOI is not submodular in general, and in particular submodularity does not hold in the selection problem (Tolpin and Shimony 2012), even in a very limited case involving only two items. An interesting approximate solution for the batch version of the selection problem appears in (Reches et al. 2013), which also proved that the selection problem is NP-hard. Their theoretical bounds on the approximation error are not based on submodularity.

Specifically, the selection problem analyzed in this paper is as follows (see Section 2 for the formal definition). We have a set of items \mathcal{I} , each of which has some unknown value (or utility). The utility of each item is a random variable, and the joint distribution over the utilities of the items is known. It is possible to perform measurements on an item, thereby obtaining information about its utility. Measurements have a cost, specified by a known cost function C , which is usually an additive cost function. After performing measurements, the decision-maker selects one item. We assume a risk-neutral decision-maker, and thus the decision maker always selects an item that has the highest expected utility given the observations. The problem is to find a policy of performing measurements such that the utility of the selected item minus the cost of measurements has maximum expected value. In some settings (Tolpin and Shimony 2012; Azimi et al. 2016), a *measurement budget* is also specified, and a policy is considered valid only if this budget is not exceeded. Some *budgeted* applications (Azimi et al. 2016) optimize just the expected value of the selection (not factoring in the measurement costs), but subject to the measurement budget constraint. Another

common constraint is requiring that only a measured item may be selected, in which case submodularity holds under quite general conditions (Azimi et al. 2016); see Section 2 for a discussion on how their results relate to this paper. The latter constraint is natural for risk-averse decision makers, e.g. in the above hiring decision application, we may not wish to take the risk of hiring anyone we have not interviewed.

There are two common selection problem settings: batch, and online (also called sequential, or conditional). In the online setting, the decision-maker performs some measurements, then based on the resulting observations can decide on whether or not to perform additional measurements, etc. A policy in this case is essentially a conditional plan. In the batch setting, the decision-maker decides on a set (batch) of measurements to perform. The measurements are done essentially “in parallel”, in the sense that the decision-maker does not get to perform additional measurements after receiving observations from previous ones. Given the observed results, the decision-maker then needs to make the final selection of an item. In this paper we consider only the batch setting.

In the batch setting, the value of information is the expected value of the best item given the observations, minus the expected value of the best item according to the initial (prior) distribution. That is, before receiving the information, there is some item that has the best expected value, which we call the “current best” item α . For simplicity we assume that this item is unique. After receiving the observations O , some other item $\beta(O)$ may have the highest expected value. The expected value of the difference $u_{\beta(O)} - u_{\alpha}$ is the value of information (VOI). Note that both the identity of the resulting best item β and its utility depend on O . In the batch setting with perfect observations, the distribution over the observed values is equal to the utility distribution of the respective item, and a set of measurements is fully specified by a set of items to be measured. Thus finding an optimal policy can be done by finding a set of measurements S to perform that has the highest expected VOI minus cost (also called the *net VOI*). In this paper, we consider mostly the case of perfect observations, i.e. where as a result of performing a measurement on an item, its precise utility value becomes known. In general, measurements can generate noisy (imperfect) observations. We briefly point out the cases where our results can be extended beyond perfect observations.

2. Main Results

The theoretical results in this paper (Section 2) are as follows: the expected value of perfect information in the batch setting is neither submodular nor supermodular, in general. However, submodularity holds in the following cases:

- Theorem 1: the item utilities are jointly independent, and the utility of the currently best item α is known.
- Theorem 3: the utility of α is known, and there are only at most two additional items (may be dependent).
- Theorem 4: the item utilities are jointly independent, the utility of α is sufficiently high (condition C1), and the decision maker is constrained to *always* measure α .

Theorem 1 is important because even this simple setting leads to an NP-hard selection problem (Theorem 2). We also show by providing simple discrete-valued counterexamples where attempting to generalize these theorems fails. Theorem 3 cannot be generalized to more items, as a counterexample with three items (in addition to α) is presented. In Theorem 4, violating condition C1 makes the theorem break, even with two items (in addition to α). Finally, we capitalize on the submodularity results (Theorems 1 and 4) by suggesting a simple “compound” greedy scheme in Section 3 for near-optimal solution of the selection problem, and compare its performance to the standard greedy algorithms on a wine quality dataset.

2 Main Results

We begin by formally defining the perfect information batch selection problem.

Definition B.1 (perfect information batch selection setting)

Let $\mathcal{I} = \{I_0, I_1, \dots, I_n\}$ be a set of $n + 1$ items with uncertain utility, represented by r.v.s X_0, \dots, X_n . We assume w.l.o.g. that the current best item α is item I_0 . For a cost C_i we can measure I_i , obtaining a (perfect) observation of the utility of this item. We select a subset $S \subseteq \mathcal{I}$ to be measured as a batch, for a total cost of $\sum_{I_i \in S} C_i$, after which we observe the results O (the true utilities of the items in S), and select a final item $I_f(O)$ that has the highest expected utility given the observations.

The *optimization version* of the perfect information batch selection problem is: under the perfect information batch selection setting, find the set that achieves:

$$\max_{S \in \mathcal{I}} (E_S[I_f(O)] - \sum_{I_i \in S} C_i) \quad (\text{B.1})$$

where the subscript of E denotes the set of indices of the variables over which the expectation is performed, and O are the observations due to measuring the items in S . Optionally, in the *budget limited* version of the selection problem, we are given a budget limit C and need to optimize S under the additional constraint:

$$\sum_{I_i \in S} C_i \leq C \quad (\text{B.2})$$

In order to simplify some of the proofs below, we make the additional assumption that the decision maker always picks item α if its expected utility is at least as high as that of all other items, given the past observations. That is, “among equals, prefer item α ”. Denoting the expected value of X_i by μ_i , note that by construction $\mu_\alpha \geq \mu_i$ for all $i > 0$. For a set of items $S \subseteq \mathcal{I}$, denote the expected value of information of a (perfect) observation of the utility of all these items by $VPI(S)$, defined as the expected value $E_S[I_f(O)] - \mu_\alpha$ (with expectation taken over all possible observations on S). Denote by p_i the PDF of random variable X_i .

Example B.1

Consider a wine selection problem with quality distributions similar to Figure B.2. Suppose that one wine case α that we wish to purchase has a known quality of $u_\alpha = 8$. We have been offered two additional options, one with a quality distribution X_1 uniformly distributed in $\{5, 6, 7, 8, 9\}$, i.e. $\mu_1 = 7$, the other (X_2) with quality in $\{4, 10\}$, again uniformly distributed, so $\mu_2 = 7$. Suppose that our utility scale is linear in the quality, that all wines cases cost the same (or that cost has already been factored in negatively into the quality). Since the wines are not known to be related, we assume that the quality distributions are independent. Testing some of the wine cases is possible (at a known cost, though we ignore such costs at present), thereby revealing their true quality. Note that if we test no wines, then we should rationally pick the α wine for a quality of 8. If we choose to test wine case 2 prior to the purchase, then with probability 0.5 its quality is revealed as 10, and we purchase it,

thereby gaining 2. Otherwise, stick with α , and gain nothing. On average we gain 1, so $VPI(\{X_2\}) = 1$.

2.1 Batch VOI when the Utility of the Currently Best Item is Known

Theorem B.1

For a perfect information batch selection setting with independent item utility distributions, where the utility u_α of the currently best item α is known, the value of perfect information $VPI(S)$ is a submodular set function.

Proof: Due to independence, an item that has not been observed will never be selected (except for α). Observed item i will be selected if it is observed to have utility greater than u_α and the rest of the observed items. (For conciseness, we use $M(S)$ to denote $\max(\{u_i | i \in S\})$, and $M(X_S)$ to denote the respective random variable $\max(\{X_i | i \in S\})$.) Therefore, the VPI of observing (the utility of) a set of items S that does not include α is:

$$VPI(S) = \int_{M(S) > u_\alpha} (M(S) - u_\alpha) \prod_{i \in S} p_i(u_i) du_i \quad (\text{B.3})$$

$$= \int (\max(M(S), u_\alpha) - u_\alpha) \prod_{i \in S} p_i(u_i) du_i \quad (\text{B.4})$$

$$= E_S[\max(M(X_S), u_\alpha)] - u_\alpha \quad (\text{B.5})$$

Now write down the difference in VPI between $S \cup \{I\}$ and S , for some item $I \notin S$ using Equation B.3:

$$\begin{aligned} VPI(S \cup \{I\}) - VPI(S) &= (E_{S \cup \{I\}}[\max(M(X_{S \cup \{I\}}), u_\alpha)] - u_\alpha) \\ &\quad - (E_S[\max(M(X_S), u_\alpha)] - u_\alpha) \\ &= E_{S \cup \{I\}}[\max(M(X_S), X_I, u_\alpha) - \max(M(X_S), u_\alpha)] \\ &= E_{S \cup \{I\}}[\max(X_I - \max(M(X_S), u_\alpha), 0)] \end{aligned}$$

Consider the difference in VPI for set $S' = S \cup \{J\}$ for some item $J \neq I, J \notin S$. We

have:

$$\begin{aligned}
 VPI(S' \cup \{I\}) - VPI(S') &= E_{S' \cup \{I\}}[\max(X_I - \max(M(X_{S'}), u_\alpha), 0)] \\
 &= E_{S \cup \{I, J\}}[\max(X_I - \max(M(X_S), X_J, u_\alpha), 0)] \\
 &\leq E_{S \cup \{I, J\}}[\max(X_I - \max(M(X_S), u_\alpha), 0)] \\
 &= E_{S \cup \{I\}}[\max(X_I - \max(M(X_S), u_\alpha), 0)] \\
 &= VPI(S \cup \{I\}) - VPI(S)
 \end{aligned}$$

Where the inequality follows due to removing a term from the (negated) maximization. We have obtained that for all sets S that do not include α , the difference in VPI is non-increasing as S (setwise) increases. Therefore, $VPI(S)$ is a submodular function of S . \square

Corollary B.1

Theorem B.1 also holds given only a distribution over u_α , if there is no way to obtain additional information about u_α . That is because an optimal (risk neutral) decision maker would have to act as if $u_\alpha = \mu_\alpha$.

A similar argument leads to a generalization to noisy observations: although Theorem 1 is stated in terms of perfect information, this is not an inherent limitation. Consider a more general setting where measurements are noisy, but the value of each item can be measured only once. In this setting, one can simply use the expected posterior value instead of the actual value when making the decision, and our results still apply. However, in settings where the measurement types on an item are allowed to vary (e.g. allow a choice between one and two conditionally independent measurements, or a choice of measurements that reveal different features of an item), it is well known that submodularity does not hold (Frazier and Powell 2010; Tolpin and Shimony 2012).

Observe that Theorem B.1 is relevant to additional settings. First, consider the special case where $u_\alpha = 0$. In this case, action α can be re-cast as making no selection at all, and the conditions of the theorem hold if all items have a non-positive prior expected value. This actually is reasonable when items with uncertain value are being sold to our decision-making agent, as the seller wishes to gain from the sale, and presumably would not wish to sell an item for less than its expected value.

2. Main Results

Another setting in which Theorem B.1 applies is if the agent is *not allowed* to select an item unless its value has been measured or is previously known. In this setting the requirement that u_α is greater than the expectation of all the rest of the items can be dropped, and the results can be made significantly more general, as shown in Lemma 1 in (Azimi et al. 2016). The lemma states that the expectation of the maximum of a set of random variables is monotonic non-decreasing and submodular, and does not even require that the variables be independent. Lemma 1 can thus also be used to prove our Theorem B.1. It is interesting to note that in order to apply Lemma 1 in (Azimi et al. 2016), they also required perfect observations. However, unlike our Theorem B.1, the lemma cannot be easily applied if we relax the perfect information limitation, as that would lead to an apparent contradiction due to dependencies, as discussed in Section 2.1.

Complexity of the Selection Problem

The batch measurements selection problem was shown to be NP-hard (Reches et al. 2013), in a setting where multiple noisy measurements per item are allowed. We show that the problem gives rise to an NP-hard decision problem even if the observations are perfect.

Definition B.2 (perfect information budget-limited batch selection decision problem (PBSP))

In the perfect information batch selection setting from Definition 1, is there a subset $S \subseteq \mathcal{I}$, that has a total measurement cost not greater than C , such that the expected utility of the final item I_f selected after observing the utility of the items in S , is at least U ?

Theorem B.2

The PBSP is NP-hard.

The proof appears in Appendix A, by reduction from Knapsack to a PBSP restricted to the case where u_α is known to be 0, and where the unknown item utility distributions are independent over $\{-1, 1\}$. It follows immediately from these restrictions that the perfect information budget-limited batch selection decision problem remains NP-hard under the conditions of Theorem 1. Note, however, that if we also

restrict the measurement costs in PBSP to be all equal, a greedy algorithm would result in an optimal subset, and thus a trivial polynomial time solution to the PBSP. Whether the PBSP with equal costs but with an *arbitrary* independent discrete distributions is NP-hard is an open problem.

VPI in the Presence of Dependencies

We now consider the perfect information batch selection setting, without the independence assumption. With dependencies the amount of information obtained by additional observations, having already made some observations, is ususally reduced. Intuition would suggest that the same would therefore occur for the VPI as well. Indeed, in the base case of $n=2$ the VPI is still subadditive. For example, the wine selection problem in example 1 with the same marginal distributions, but where X_1 and X_2 are dependent, fall under this case.

Theorem B.3

For a batch selection setting with 3 items, where the utility of the currently best item α is known, the value of perfect information is subadditive, i.e. $VPI(\{1\}) + VPI(\{2\}) \geq VPI(\{1,2\})$.

Proof: Note that, unlike the independent case, when observing only one item it is actually possible to select either the other, unobserved item, or α . This results in:

$$VPI(\{1\}) = \int_{\max(u_1, \mu_{2|1}(u_1)) > u_\alpha} (\max(u_1, \mu_{2|1}(u_1)) - u_\alpha) p_1(u_1) du_1$$

where $\mu_{2|1}(u_1)$ is the expected utility of item 2 given that the item 1 was observed to have utility u_1 , defined as:

$$\mu_{2|1}(u_1) = \int_{u_2} p_2(u_2|u_1) u_2 du_2$$

the VPI for item 2 is defined symetrically, exchanging the roles of items 1 and 2 in these equations.

The value of information for observing both item 1 and item 2 (followed by selecting the best of them or α , whichever has maximal utility) is:

$$VPI(\{1,2\}) = \int \int_{\max(u_1, u_2) > u_\alpha} (\max(u_1, u_2) - u_\alpha) p_{1,2}(u_1, u_2) du_1 du_2 \quad (\text{B.6})$$

2. Main Results

Separating out the domain we can write:

$$VPI(\{1,2\}) = \int_{u_1 > u_\alpha} \int_{u_2 \leq u_1} (u_1 - u_\alpha) p_{1,2}(u_1, u_2) du_1 du_2 + \int_{u_2 > u_\alpha} \int_{u_1 < u_2} (u_2 - u_\alpha) p_{1,2}(u_1, u_2) du_1 du_2$$

Denote the first integral by J_1 and the second by J_2 , for convenience. We now rewrite $VPI(\{1\})$ as a sum over the regions:

$$VPI(\{1\}) = \int_{u_1 > u_\alpha} (\max(u_1, \mu_{2|1}(u_1)) - u_\alpha) p_1(u_1) du_1 + \int_{\{u_1 | u_1 \leq u_\alpha \wedge \mu_{2|1}(u_1) > u_\alpha\}} (\mu_{2|1}(u_1) - u_\alpha) p_1(u_1) du_1$$

Now, dropping the $\mu_{2|1}(u_1)$ from the maximization in the first integral and noting that the second integral is non-negative, we get:

$$\begin{aligned} VPI(\{1\}) &\geq \int_{u_1 > u_\alpha} (u_1 - u_\alpha) p_1(u_1) du_1 \\ &= \int_{u_1 > u_\alpha} \int_{u_2 \leq u_1} (u_1 - u_\alpha) p_{1,2}(u_1, u_2) du_1 du_2 + \int_{u_1 > u_\alpha} \int_{u_2 > u_1} (u_1 - u_\alpha) p_{1,2}(u_1, u_2) du_1 du_2 \\ &= J_1 + \int_{u_1 > u_\alpha} \int_{u_2 > u_1} (u_1 - u_\alpha) p_{1,2}(u_1, u_2) du_1 du_2 \geq J_1 \end{aligned}$$

Likewise we show that $VPI(\{2\}) \geq J_2$, and thus $VPI(\{1\}) + VPI(\{2\}) \geq VPI(\{1,2\})$.

□

Unfortunately, this submodularity result has no *practical* use, as it does not generalize to $n \geq 3$, as is evident from the following counterexample. Let $u_\alpha = 10$, and we have 3 additional items with utility distributed as binary variables, with values $\{L, H\}$. The dependency is “parity”, that is, exactly an even number of the items have value H , and the rest have value L . The distribution over the 4 possible legal configurations is uniform, i.e. each has probability 0.25. The utility values are: $u_{1_L} = u_{2_L} = 5$, and $u_{1_H} = u_{2_H} = 13$, so that $\mu_1 = \mu_2 = 9 < u_\alpha$. For the 3rd item, we have: $u_{3_L} = 0$, and $u_{3_H} = 18$, so that $\mu_3 = 9 < u_\alpha$.

Note that the marginal distribution over each of the items is uniform, and remains uniform given the observation of one other item, i.e. the variables are pairwise independent. The individual VPIs are therefore:

$$VPI(\{1\}) = 0.5 \times 0 + 0.5 \times (13 - 10) = 1.5$$

and due to symmetry we also have $VPI(\{2\}) = 1.5$. Having observed both items 1 and 2, the utility of item 3 is known with certainty, and it is selected if known to have

value H . Therefore we have:

$$VPI(\{1,2\}) = 0.25 \times (0 + (18-10) + (18-10) + (13-10)) = 4.75 > VPI(\{1\}) + VPI(\{2\})$$

Finally, note that Lemma 1 from (Azimi et al. 2016) does not require independence, and can be used to show that VOI of perfect measurements is submodular if we do not allow the agent to select an unmeasured item. The above counterexample shows that the applicability of Lemma 1 cannot easily be extended to allow imprecise measurements. Allow a very noisy measurement for item 3, e.g. $P(\text{ObserveHIGH}|u_3 = H) = 0.51$, $P(\text{ObserveHIGH}|u_3 = L) = 0.49$. The VOI values change only slightly, but now the requirement that only measured items can be selected is met. However, as shown above, the value of information is not submodular.

2.2 Batch VOI when the Utility of the Currently Best Item is Unknown

Consider now that we are given a distribution over u_α , but unlike Corollary 1, additional information about u_α can be obtained. For simplicity, consider just the case with 2 items. In the well-known case exhibited in figure B.1, we can see that it is not possible, by observing only one item, to make an optimally behaving agent change the choice from α to β . So the individual VPI are zero. But since there is some non-zero probability that u_α is less than u_β , observing both items it is possible that β will be selected to increase the utility, therefore we have $VPI(\{\alpha, \beta\}) > 0$. In this case the value of perfect information is supermodular, but does this hold in general for 2 items?

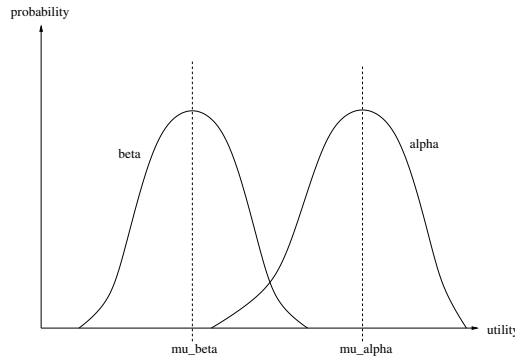


Figure B.1: Utility distributions with supermodular VPI

2. Main Results

Consider the distributions: u_α is evenly distributed: $P(u_\alpha=0) = P(u_\alpha=10) = 0.5$, and u_β is distributed as $P(u_\beta=1) = 0.7, P(u_\beta=11) = 0.3$. We get $(\mu_\alpha = 5) > (\mu_\beta = 4)$. The individual values of perfect information are:

$$VPI(\{\alpha\}) = 0.5 \times 0 + 0.5 \times (4 - 0) = 2$$

$$VPI(\{\beta\}) = 0.7 \times 0 + 0.3 \times (11 - 5) = 1.8$$

For observing both items, we get:

$$\begin{aligned} VPI(\{\alpha, \beta\}) &= 0.5 \times 0.3 \times (11 - 10) + 0.5 \times 0.3 \times (11 - 0) + 0.5 \times 0.7 \times (1 - 0) + 0.5 \times 0.7 \times 0 \\ &= 2.15 < VPI(\{\alpha\}) + VPI(\{\beta\}) \end{aligned}$$

This example is discouraging, since we have neither submodularity nor supermodularity. An interesting question is about the VPI among sets of observations that *must* include an observation of the currently best item. In general, the VPI of such sets is neither submodular nor supermodular, as shown by the following counterexample.

We have 3 items, with distributions as follows. Current best item α , distributed: $P(u_\alpha = 20) = P(u_\alpha = 0) = 0.5$. Second best item β , distributed: $P(u_\beta = 9) = P(u_\beta = 5) = 0.5$, and third item γ , distributed $P(u_\gamma = 6) = P(u_\gamma = 2) = 0.5$. This gives us: $\mu_\alpha = 10, \mu_\beta = 7$ and $\mu_\gamma = 4$. If we observe only item α , if it has a low value we pick item β so we have:

$$VPI(\{\alpha\}) = P(u_\alpha = 0) \times (\mu_\beta - 0) = 3.5$$

If we also observe item β , this makes no difference as any possible utility value for item β is still higher than μ_γ . Likewise, observing α and γ , we still select α if $u_\alpha = 20$ and β if $u_\alpha = 0$. Therefore we have:

$$VPI(\{\alpha, \beta\}) = VPI(\{\alpha, \gamma\}) = VPI(\{\alpha\}) = 3.5$$

However, the value of observing all items is higher, since item γ may be better:

$$\begin{aligned} VPI(\{\alpha, \beta, \gamma\}) &= P(u_\alpha = 0) \times [P(u_\beta = 9) \times 9 + P(u_\beta = 5) \times (P(u_\gamma = 6) \times 6 + P(u_\gamma = 4) \times 5)] \\ &= 0.5 \times [0.5 \times 9 + 0.5 \times 5.5] \\ &= 3.625 > VPI(\{\alpha, \beta\}) + VPI(\{\alpha, \gamma\}) - VPI(\{\alpha\}) \end{aligned}$$

which clearly violates submodularity for sets containing observations of item α .

However, if u_α is always sufficiently high, i.e. if every possible value of u_α is no less than μ_β , submodularity does hold among such sets. In general, denote:

Condition C 1

$P(u_\alpha < \mu_i) = 0$ for all items i other than α .

Example B.2

Consider the same wine selection problem instance as in example 1, except that the quality of the α wine case is no longer known to be 8: instead its quality is uniformly distributed among $\{7, 8, 9\}$. With the qualities X_1 and X_2 being independent and distributed as in example 1, this example obeys condition C1.

Formally, denote by $VPI^\alpha(S)$ the value of information of perfectly observing the utility of all items in S , as well as that of α . (This is equivalent to $VPI(S \cup \{\alpha\})$, but we wish to emphasise that this is a function of S not including α , hence the above notational variant.)

Theorem B.4

For a batch selection setting with jointly independent items where condition C1 holds, the value of perfect information $VPI^\alpha(S)$ is a submodular set function of S .

Proof: Note that this theorem is a strict generalization of the corollary of Theorem B.1, as condition C1 always holds trivially if α is the current best item and u_α is known. Condition C1 insures that after the observations of items in S , the optimal policy must select either one of these observed items, or α . (This is not necessarily the case if C1 is violated.) Therefore the VPI here can be obtained in a manner similar to Equation B.3, with integration over u_α :

$$VPI^\alpha(S) = \int_{u_\alpha} \int_{M(S) > u_\alpha} (\max(\{u_i | i \in S\}) - u_\alpha) p_\alpha(u_\alpha) du_\alpha \prod_{i \in S} p_i(u_i) du_i \quad (\text{B.7})$$

$$= E_{S \cup \{\alpha\}}[\max(M(X_S), u_\alpha) - u_\alpha] \quad (\text{B.8})$$

Similar to the proof of Theorem B.1, write down the difference in VPI^α between

3. Application of Results

$S \cup \{I\}$ and S , for some item $I \notin S$ using Equation B.7:

$$\begin{aligned}
 VPI^\alpha(S \cup \{I\}) - VPI^\alpha(S) &= E_{S \cup \{I, \alpha\}}[\max(M(X_{S \cup \{I\}}), u_\alpha)) - u_\alpha] \\
 &\quad - E_{S \cup \{\alpha\}}[\max(M(X_S), u_\alpha)) - u_\alpha] \\
 &= E_{S \cup \{I, \alpha\}}[\max(M(X_S), X_I, u_\alpha) - \max(M(X_S), u_\alpha)] \\
 &= E_{S \cup \{I, \alpha\}}[\max(X_I - \max(M(X_S), u_\alpha), 0)]
 \end{aligned}$$

Consider the difference in VPI for set $S' = S \cup \{J\}$ for some item $J \neq I, J \notin S$. We have:

$$\begin{aligned}
 VPI^\alpha(S' \cup \{I\}) - VPI^\alpha(S') &= E_{S' \cup \{I, \alpha\}}[\max(X_I - \max(M(X_{S'}), u_\alpha), 0)] \\
 &= E_{S \cup \{I, J, \alpha\}}[\max(X_I - \max(M(X_S), X_J, u_\alpha), 0)] \\
 &\leq E_{S \cup \{I, J, \alpha\}}[\max(X_I - \max(M(X_S), u_\alpha), 0)] \\
 &= E_{S \cup \{I, \alpha\}}[\max(X_I - \max(M(X_S), u_\alpha), 0)] \\
 &= VPI^\alpha(S \cup \{I\}) - VPI^\alpha(S)
 \end{aligned}$$

Therefore, $VPI^\alpha(S)$ is a submodular set function of S . \square

3 Application of Results

A typical application of submodularity is in algorithms that compute near-optimal policies for selection in the perfect information batch selection setting. Consider for example a batch setting selection problem where the measurement cost function \mathcal{C} is supermodular (or additive, as a special case common in applications). As a result, the optimal solution to the (batch setting) selection problem is to measure a set of items S that maximizes $VPI(S) - \mathcal{C}(S)$ (the net VPI), followed by selecting the item with the best expectation given the observations.

If we know the utility of item α , then $VPI(S) - \mathcal{C}(S)$ is submodular due to Theorem B.1. We can thus use a standard greedy algorithm that starts with an empty candidate set S , and repeatedly adds to S items that have the highest net gain (best (marginal) VPI minus cost), until no item has a positive net gain. We call this method the **(additive) greedy** algorithm. According to a fundamental result by Nemhauser et al. 1978, the greedy algorithm already guarantees an expected utility that is close to optimal. The quality of the greedy algorithm in practice is usually much better than

the guaranteed bounds, and a similar tendency can be seen in the example wine-selection application below. The quality of the results seems to occur due to the fact that submodularity is a guarantee against “premature stopping” in the greedy algorithms¹; deviations from optimality resulting from picking non-optimal items early-on seem much less problematic in practice than indicated by the worst case in theory (a factor of $1 - 1/e$ for the monotonic case, worse for non-monotonic which is the case here).

As cases where u_α is known may be rare, it is possible to use a similar scheme if u_α is not known, but the distributions obey condition C1. In this case, run the greedy algorithm twice: once for sets that do not contain measurements of item α , and once for sets that do contain such measurements; compare the expected value of both resulting measurement sets, and return the better of the two. We call this method the **compound greedy** algorithm. Again, Theorems 1 and 4 imply that the functions we optimize in both cases are submodular, thus the greedy algorithms return sets that are near-optimal.

3.1 Example Setting: Wine Selection

We examine an example application for the perfect information batch selection setting, and solve a selection problem on a typical set of items. A comparison of algorithm performance on such a dataset indicates the type of results one can observe with greedy optimization algorithms for the selection problem.

Definition B.3 (Wine selection problem)

Given a set of wine types $\mathcal{I} = \{I_0, \dots, I_n\}$, each wine has an unknown quality, but a quality distribution is known for each type. In addition, for a known cost C_i , we can purchase and send a bottle of each wine type to a sommelier for analysis and quality determination (or taste it ourselves, for the few people who actually understand wine quality, though clearly not the authors of this paper). Which subset S of the wines (if any) needs to be sent to the sommelier in order to maximize the expected utility of testing and final decision? (That is, maximize the expected quality of the final

¹“Premature stopping” is a term used to mean that although there is a set of items with a combined VOI greater than its measurement cost, the (greedy) algorithm decides not to measure any additional items because their individual VOI is too low. See example B.3 in Section 3.1.

3. Application of Results

selection, minus the sum of costs C_i of wines in S , i.e. the net VPI).

The setting for the tests was based on the UCI white wine quality dataset (Paulo Cortez 2009; P. Cortez et al. 2009). The dataset contains over 1000 wines, with 11 feature values for each wine, such as pH, alcohol level, etc. The target attribute value (quality) is based on evaluations made by wine experts, and ranges from 1 (very bad) to 10 (excellent).

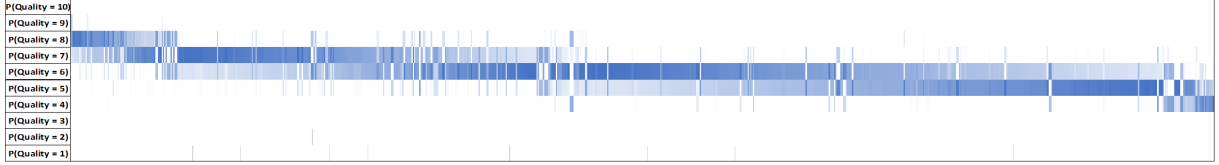


Figure B.2: Wine quality distributions

Using this dataset, we constructed for each wine a quality distribution based on the quality distribution of all wines in the data set that had the same feature values. This distribution was adjusted by applying Kernel density estimation (KDE) using a Gaussian kernel and a rule of thumb suggested in (Silverman 1986) that increases the kernel width as a function of the variance. This resulted in the wine quality distributions depicted in Figure B.2, a distribution scatter plot where darker color indicates higher probability. Each value on the X axis indicates a specific wine type, with wines sorted by expected quality value. The wine quality distributions are assumed to be independent.

Using the above distribution, the following experiments were conducted. Each experiment was on a set \mathcal{I} of $n + 1$ randomly picked wines from the dataset, where n was an experimental parameter, and for each wine a random cost C_i was drawn uniformly between 0.01 to 0.1 (assumed to be on the same scale as quality values). The wine with best expected value from \mathcal{I} is the α wine, the prior best. We then used 4 different methods to find the measurement policy (i.e. batch of wines to be tested).

1. **Exhaustive:** Every possible subset S of \mathcal{I} (both with and without the alpha wine) was examined. The S which maximized the net VPI was returned. S here is the optimal (batch) measurement policy.
2. **Greedy (additive)** approach. The wines are kept sorted according to their myopic expected net VPI w.r.t. the current batch. A batch S is incrementally con-

structed, starting from the empty set: every iteration, the best candidate wine from $\mathcal{I} - S$ is added to S , as long as the net myopic VPI for adding this wine is positive.

3. **Greedy (rate)** approach. This greedy method is the same as the additive greedy approach, except that the wines are kept sorted according to their expected VPI divided by cost of the measurement. Once this value drops below 1 for all remaining wines, the algorithm returns the current batch. This approach is the same as in (Azimi et al. 2016), modified to the wine selection problem.
4. **Compound greedy** approach: Run the greedy (additive) algorithm twice: once for sets that do not contain the α wine, and once for sets that do contain α . Compare the expected net VPI of both resulting measurement sets, and return the better of the two.

3. Application of Results

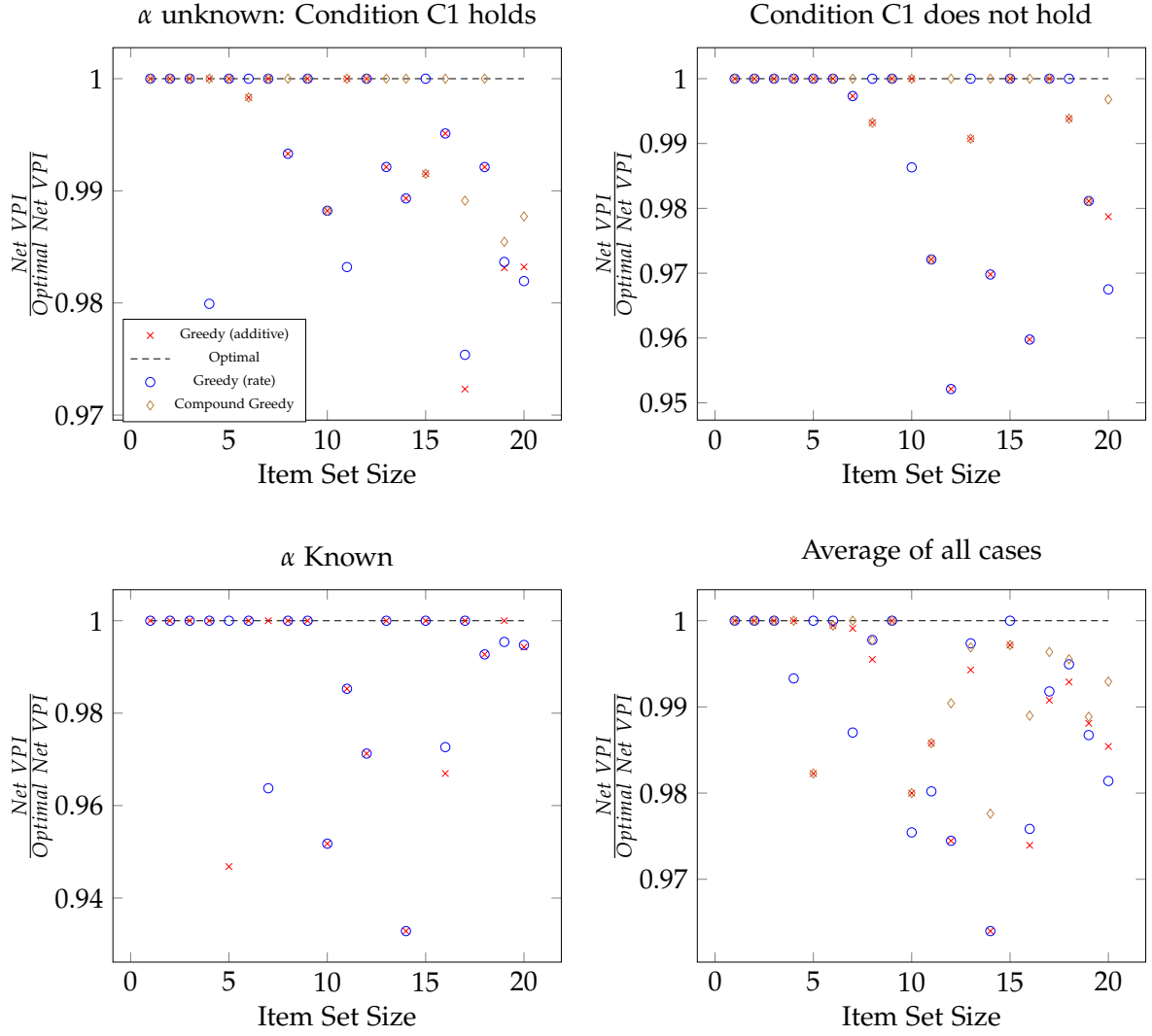


Figure B.3: Comparison of net VPI for various item set sizes

For each of the following cases, we varied n , the number of items, from 1 and 20:

1. Known u_α , created by setting the value of u_α to its mean in each randomly picked item set. Here compound greedy is the same as simple greedy, so is not shown.
2. u_α unknown, but condition C1 holds (generated by random sampling of sets, rejecting sets where C1 did not hold).
3. Instances where condition C1 does not hold (and obviously unknown u_α).

The net VPI, averaged over 5 random item sets for each item set size, is shown in Figure B.3.

Both standard greedy algorithms averaged 0.99 of the optimal net VPI, while compound greedy averaged slightly better at 0.993, all considerably better than the theoretical bound. It is interesting to observe that the greedy algorithms performed well even in many cases where the theorems do not guarantee submodularity, such as the cases where condition C1 did not hold (Figure B.3 upper right). In some cases rate greedy performed better than both of the other methods, but a rate-based version of compound greedy (not shown) dominates rate-greedy. In extreme cases (which did not occur in the above runs) the net VPI is 0 for both rate and additive greedy, even though considerable net VPI is achievable. This occurs due to the “premature stopping” phenomenon caused by non-diminishing returns.

Example B.3

Consider the case where condition C1 holds as in Example B.2, with quality distribution of the α wine being uniform among $\{7, 8, 9\}$, but in addition the best possible quality in all the other items is no better than the $E(X_\alpha)$ value, such as when the only other choice is X_1 distributed uniformly among $\{6, 7, 8\}$. In this case the VPI of every singleton set is 0, similar to the situation depicted in Figure B.1, whereas measuring both wines results in a gain of 1 with probability $\frac{1}{9}$, and thus $VPI^\alpha(\{X_1\}) = \frac{1}{9}$. This will cause the rate and additive greedy algorithms to incorrectly return an empty set of items to be measured. The compound greedy algorithm avoids exactly this pitfall.

3. Application of Results

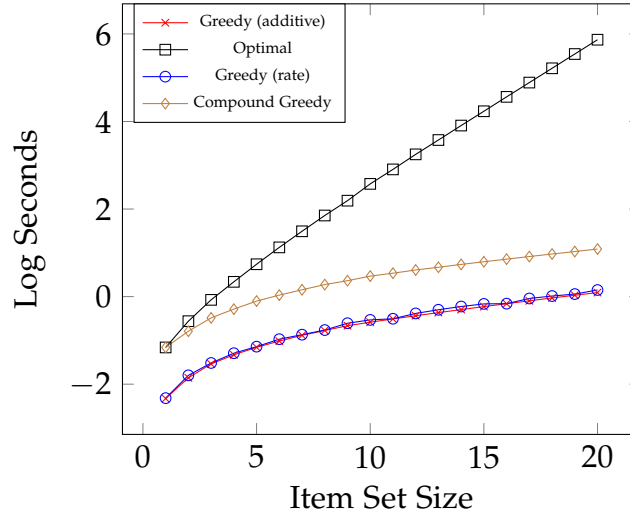


Figure B.4: Algorithm runtime comparison: time vs. item set size

We now turn to the issue of computation time. All the above algorithms require evaluation of the VPI of a batch, which can itself be non-trivial. An initial naive implementation caused even the *greedy* algorithms to time out on sets of 20 wines. This sub-problem can be handled in the general case by approximating the VPI (i.e. expectation of the maximum) of each batch by sampling (Azimi et al. 2016). In the wine selection problem, however, we have independent discrete random variables with greatly overlapping domains, so we can cheaply compute the distribution of the maximum, and from there evaluate the expectation of the maximum exactly.

Runtimes for the algorithms appear in Figure B.4, performed on an Intel(R) Core(TM) i7-4700HQ 2.40GHz with 8 GB RAM running windows 8.1 x64, using multiple-thread implementations. The software was implemented in C# with optimizations. Clearly, the exhaustive method delivers the best net VPI, but its runtime is prohibitive for large sets of wines.

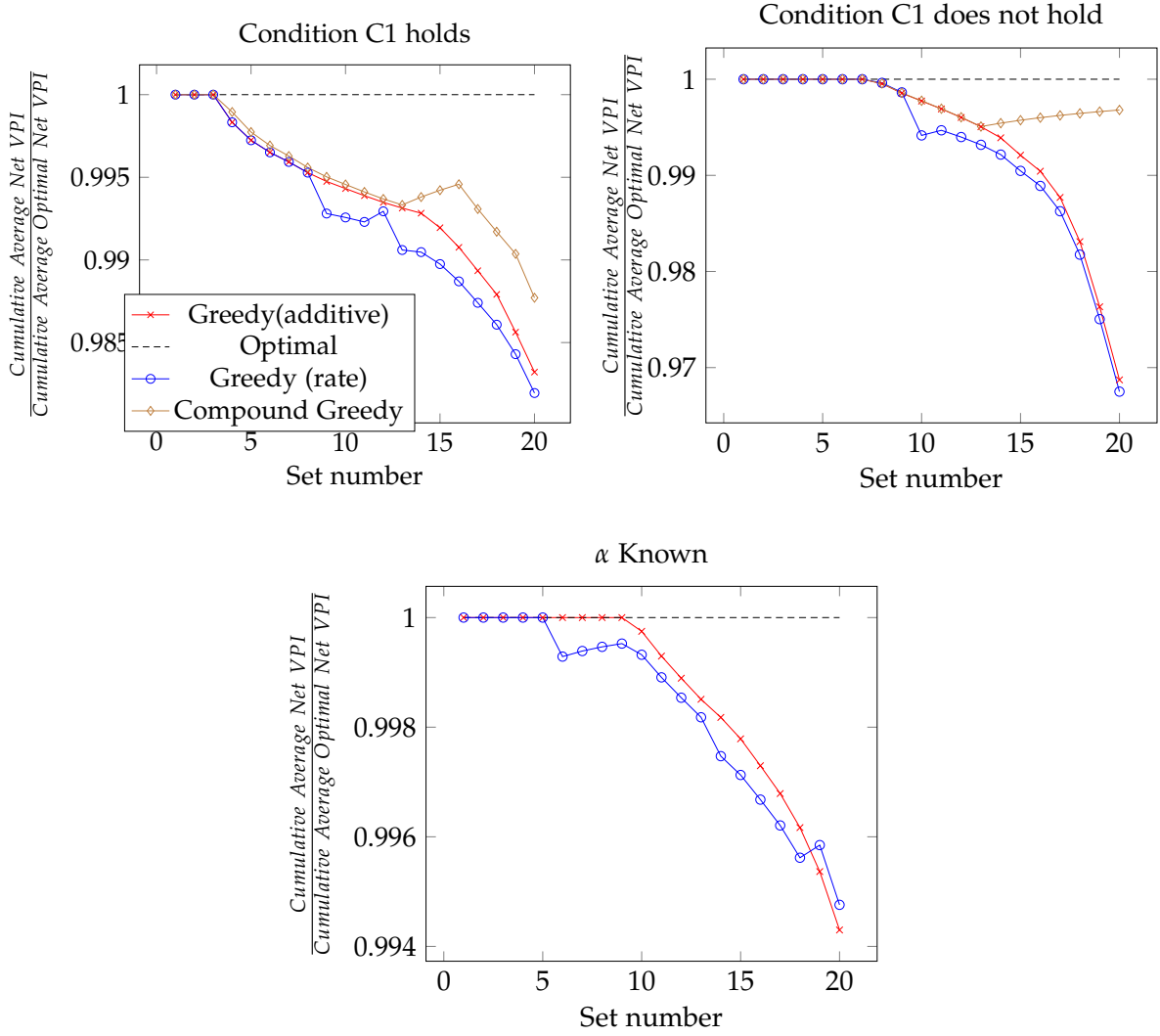


Figure B.5: Comparison of net VPI for item set size 20

Both the additive and rate greedy were the fastest, with compound greedy roughly a constant factor slower. In fact, despite the improved VPI computation, this part still dominates the runtime, and adding caching of computations of random variable maximizations resulted in the compound greedy algorithm being only a few percent slower than the other greedy algorithms (not shown). Therefore, although the improvement due to the compound greedy algorithm appears small, it comes essentially for free and is thus worthwhile. The greedy algorithms appear to be scalable: a experimental run with $n = 100$ wines resulted in runtimes of approximately 200 seconds for each of the greedy algorithms (including compound greedy, with caching).

As differences in performance were more pronounced for the larger set sizes, we tried more instances with $n = 20$, which is the largest for which we could obtain the

optimal results in reasonable time. The results are shown as a cumulative average plot (Figure B.5). While the value of information for all greedy algorithms is still close to the optimal value, the compound greedy algorithm again is slightly better, averaging 0.99 of the optimal net VPI, while the additive and rate greedy averaged roughly 0.98 of the optimal net VPI. The compound-greedy algorithm showed an improvement over the simple greedy algorithms whether or not condition C1 held.

4 Conclusion

We have examined cases where the batch value of perfect information is submodular in the selection problem, mostly in the case where the item utility distributions are independent. We have shown that a resulting optimization problem is NP-hard, even in such restricted cases. Nevertheless, greedy optimization algorithms seem to achieve good results in practice. The theoretical results suggest that greedy algorithms should be supplemented by examining sets that include the currently best item, even if its individual VPI is zero, and this is supported by empirical evidence.

We suggest that such deviations from submodularity indicate points where the greedy and myopic optimization schemes can be improved w.r.t. net VPI, at relatively little computational cost. As such, the simple method suggested in this paper complements the idea of “blinker VOI” (Hay et al. 2012). Our motivation for this work comes from meta-reasoning in search, where the information is gathered by search actions, and solving a selection problem is a first step that suggests a way to proceed at the first level in the search tree. Generalization of these methods to selecting computations at deeper levels of the search tree is a non-trivial issue for future work.

Acknowledgments

Partially supported by ISF grant 417/13. We thank the anonymous reviewers for comments that greatly shortened some theorem proofs, and other suggestions that increase the applicability of the results presented in this paper.

Appendix A

Proof (of Theorem 2): By reduction from knapsack ((Garey and Johnson 1979) problem number [MP9]), which is re-stated below.

Definition B.4 (Knapsack problem)

Given a set of items $\mathcal{S} = \{s_1, \dots, s_n\}$, each with a positive integer weight w_i and a positive integer value v_i , a weight limit W and a target value V , is there a subset S of \mathcal{S} such that the total weight of S is at most W , and the total value of the elements of S is at least V ?

We assume w.l.o.g. that $v_i < V$ for all items, as items that violate this solve the Knapsack problem trivially.

In reducing Knapsack to PBSP, each item in \mathcal{I} in the selection problem will stand for the respective element in the Knapsack problem, adding the α item that does not correspond to any item in the Knapsack problem. As this is a simple one-to-one mapping, for the sake of simplicity we therefore abuse the notation and treat the items as if they are actually the respective elements from \mathcal{S} in the Knapsack problem. The distributions of values and costs are defined as follows: let $H = \max_{1 \leq i \leq n} \{v_i\}$, and $\varepsilon = \frac{1}{2H^2n^3}$. Then $C = W$, $U = \varepsilon(V - \frac{1}{2})$, and the distributions and measurement costs are as follows:

- For X_α we have $u_\alpha = 0$ with probability 1. The cost C_α is irrelevant (because the exact value of u_α is already known) and can be taken to be 0.
- For every other item, we have a binary-valued distribution: $P(X_i = 1) = \varepsilon v_i$, and $P(X_i = -1) = 1 - (\varepsilon v_i)$. The measurement cost of these items is given by $C_i = w_i$.

Note that indeed the current best action is s_α , because $\varepsilon v_i < \frac{1}{2}$ for all $1 \leq i \leq n$. Therefore, each item s_i becomes better than s_α if and only if s_i is observed to have a positive utility. We now show that a subset $S \subseteq \mathcal{S}$ solves the knapsack problem if and only if S solves the PBSP. Let $m = |S| \leq n$, and in order to simplify the notation below, we assume w.l.o.g. that $S = \{s_1, s_2, \dots, s_m\}$.

(\Rightarrow) Let S be a solution to the knapsack problem. We have $\sum_{i=1}^m w_i \leq W = C$, so

4. Conclusion

S satisfies the budget constraint in the selection problem. Denote the probability that at least one of the items in S has value 1 by $P(S)$. Since by construction the expected utility for a set of measurements S is exactly $P(S)$, it is sufficient to show that $P(S) \geq U$. Since the value distributions are jointly independent, we have:

$$P(S) = \sum_{i=1}^m P(X_i = 1) \prod_{j=1}^{i-1} (1 - P(X_j = 1)) = \sum_{i=1}^m \varepsilon v_i \prod_{j=1}^{i-1} (1 - \varepsilon v_j)$$

Re-arranging $P(S)$ into sums according to powers of ε , we get:

$$P(S) = \sum_{i=1}^m \varepsilon^i (-1)^{(i+1)} \sum_{\{N \subseteq [1..m] \wedge |N|=i\}} \prod_{k \in N} v_k \geq \varepsilon \sum_{j=1}^m v_j - \sum_{i'=1}^{\lfloor m/2 \rfloor} \varepsilon^{2i'} \sum_{\{N \subseteq [1..m] \wedge |N|=2i'\}} \prod_{k \in N} v_k$$

where the inequality is due to dropping all the terms for odd i (which are positive), except for $i = 1$. Now, for each i' , the number of elements in N is clearly $\binom{m}{2i'}$, which is bounded by $m^{2i'}$, and thus by $n^{2i'}$. Since by definition we also have $v_k \leq H$ for all k , we get:

$$\begin{aligned} P(S) &\geq \varepsilon \sum_{j=1}^m v_j - \sum_{i'=1}^{\lfloor m/2 \rfloor} (\varepsilon H n)^{2i'} \\ &= \varepsilon \sum_{j=1}^m v_j - \sum_{i'=1}^{\lfloor m/2 \rfloor} \left(\frac{1}{2H^2 n^3} H n \right)^{2i'} = \varepsilon \sum_{j=1}^m v_j - \sum_{i'=1}^{\lfloor m/2 \rfloor} \left(\frac{1}{2H n^2} \right)^{2i'} \\ &> \varepsilon \sum_{j=1}^m v_j - \frac{n}{4H^2 n^4} = \varepsilon \sum_{j=1}^m v_j - \frac{1}{4H^2 n^3} = \varepsilon \sum_{j=1}^m v_j - \frac{\varepsilon}{2} \\ &= \varepsilon \left(V - \frac{1}{2} \right) = U \end{aligned}$$

where the last equality follows from S being a solution to the Knapsack problem. Therefore, S is a solution to the PBSP.

(\Leftarrow) Let S be a solution to the PBSP. and thus $\sum_{i=1}^m C_i \leq C = W$, so S obeys the weight limitation of the Knapsack problem. It is thus sufficient to show that $\sum_{i=1}^m v_i \geq V$. As above, we have:

$$P(S) = \sum_{i=1}^m \varepsilon^i (-1)^{(i+1)} \sum_{\{N \subseteq [1..m] \wedge |N|=i\}} \prod_{k \in N} v_k \leq \varepsilon \sum_{j=1}^m v_j + \sum_{i'=1}^{\lceil m/2 \rceil - 1} \varepsilon^{2i'+1} \sum_{\{N \subseteq [1..m] \wedge |N|=2i'+1\}} \prod_{k \in N} v_k$$

where the inequality is due to dropping all the terms for even i (which are negative). Now, for each i' , the number of elements in N is clearly $\binom{m}{2i'+1}$, which is bounded by $m^{2i'+1}$, and thus by $n^{2i'+1}$. Since by definition we also have $v_k \leq H$ for all k , we get:

$$\begin{aligned}
 P(S) &\leq \varepsilon \sum_{j=1}^m v_j + \sum_{i'=1}^{\lceil m/2 \rceil - 1} (\varepsilon H n)^{2i'+1} \\
 &= \varepsilon \sum_{j=1}^m v_j + \sum_{i'=1}^{\lceil m/2 \rceil - 1} \left(\frac{1}{2H^2 n^3} H n \right)^{2i'+1} = \varepsilon \sum_{j=1}^m v_j + \sum_{i'=1}^{\lceil m/2 \rceil - 1} \left(\frac{1}{2H n^2} \right)^{2i'+1} \\
 &< \varepsilon \sum_{j=1}^m v_j + \frac{n}{8H^3 n^6} = \varepsilon \sum_{j=1}^m v_j + \frac{1}{8H^3 n^5} \\
 &< \varepsilon \sum_{j=1}^m v_j + \frac{1}{8H^2 n^3} = \varepsilon \sum_{j=1}^m v_j + \frac{\varepsilon}{4} = \varepsilon \left(\sum_{j=1}^m v_j + \frac{1}{4} \right)
 \end{aligned}$$

Since S is a solution to PBSP, we have $P(S) \geq U = \varepsilon(V - \frac{1}{2})$, and thus $V \leq \sum_{j=1}^m v_j + \frac{3}{4}$. As both V and $\sum_{j=1}^m v_j$ are positive integers, we also have $\sum_{j=1}^m v_j \geq V$. Therefore, S is a solution to the knapsack problem. \square

References of Paper B

- [AFF16] Javad Azimi, Xiaoli Fern, and Alan Fern. “Budgeted Optimization with Constrained Experiments”. In: *Journal of Artif. Intell. Research* 56 (2016), pp. 119–152.
- [Cor+09] P. Cortez et al. “Modeling wine preferences by data mining from physicochemical properties”. In: *Decision Support Systems* 47.4 (2009), pp. 547–553.
- [Cor09] Paulo Cortez. *Wine Quality Data Set*. <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>. [Online; UCI Machine Learning Repository]. 2009.
- [CS01] Hector Chadeand and Edward E. Schlee. “Another Look at the Radner–Stiglitz Nonconcavity in the Value of Information”. In: *Journal of Economic Theory* 107 (2001), pp. 421–452.
- [FP10] Peter I. Frazier and Warren B. Powell. “Paradoxes in Learning and the Marginal Value of Information”. In: *Decis. Anal.* 7.4 (2010), pp. 378–403.
- [Fra12] Peter I. Frazier. “Optimization via simulation with Bayesian statistics and dynamic programming”. In: *Winter Simulation Conference. WSC, 2012*, 7:1–7:16.
- [GJ79] M. R. Garey and D. S. Johnson. “Computers and Intractability, A Guide to the Theory of NP-completeness”. In: W. H. Freeman and Co., 1979, p. 190.
- [Hay+12] Nicholas Hay et al. “Selecting Computations: Theory and Applications”. In: *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*. Ed. by Nando de Freitas and Kevin P. Murphy. AUAI Press, 2012, pp. 346–355. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2297&proceeding_id=28.
- [KG09] Andreas Krause and Carlos Guestrin. “Optimal Value of Information in Graphical Models”. In: *J. Artif. Intell. Res. (JAIR)* 35 (2009), pp. 557–591. DOI: 10.1613/jair.2737. URL: <http://dx.doi.org/10.1613/jair.2737>.

- [KG11] Andreas Krause and Carlos Guestrin. “Submodularity and its applications in optimized information gathering”. In: *ACM TIST* 2.4 (2011), p. 32. DOI: 10.1145/1989734.1989736. URL: <http://doi.acm.org/10.1145/1989734.1989736>.
- [NWF78] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. “An analysis of approximations for maximizing submodular set functions - I”. In: *Mathematical Programming* 14.1 (1978), pp. 265–294.
- [PF12] Georgios Papachristoudis and John W. Fisher III. “Theoretical Guarantees on Penalized Information Gathering”. In: *Statistical Signal Processing Workshop (SSP)*. Aug. 2012, pp. 301–304. URL: <http://people.csail.mit.edu/geopapa/pubs/geopapaSSP2012.pdf>.
- [RGK13] Shulamit Reches, Ya’akov (Kobi) Gal, and Sarit Kraus. “Efficiently gathering information in costly domains”. In: *Decision Support Systems* 55.1 (2013), pp. 326–335. DOI: 10.1016/j.dss.2013.01.021. URL: <http://dx.doi.org/10.1016/j.dss.2013.01.021>.
- [RS00] Howard Raiffa and Robert Schlaifer. *Applied Statistical Decision Theory*. Wiley, 2000. ISBN: ISBN: 978-0-471-38349-9.
- [RS08] Yan Radovilsky and Solomon Eyal Shimony. “Observation Subset Selection as Local Compilation of Performance Profiles”. In: *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*. Ed. by David A. McAllester and Petri Myllymäki. AUAI Press, 2008, pp. 460–467. ISBN: 0-9749039-4-9. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1965&proceeding_id=24.
- [RS84] Roy Radner and Joseph E. Stiglitz. “A Nonconcavity in the Value of Information”. In: *Bayesian Models in Economic Theory* 5 (1984), pp. 33–52.
- [RSS06] Yan Radovilsky, Guy Shattah, and Solomon Eyal Shimony. “Efficient Deterministic Approximation Algorithms for Non-myopic Value of Information in Graphical Models”. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Taipei, Taiwan, October 8-11, 2006*.

References of Paper B

- IEEE, 2006, pp. 2559–2564. ISBN: 1-4244-0099-6. DOI: 10.1109/ICSMC.2006.385249. URL: <http://dx.doi.org/10.1109/ICSMC.2006.385249>.
- [RW91a] Stuart J. Russell and Eric Wefald. *Do the right thing - studies in limited rationality*. MIT Press, 1991. ISBN: 978-0-262-18144-0.
- [RW91b] Stuart J. Russell and Eric Wefald. “Principles of Metareasoning”. In: *Artif. Intell.* 49.1-3 (1991), pp. 361–395. DOI: 10.1016/0004-3702(91)90015-C. URL: [http://dx.doi.org/10.1016/0004-3702\(91\)90015-C](http://dx.doi.org/10.1016/0004-3702(91)90015-C).
- [Sil86] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. London: Chapman & Hall, 1986.
- [SJY03] James R. Swisher, Sheldon H. Jacobson, and Enver Yücesan. “Discrete-event simulation optimization using ranking, selection, and multiple comparison procedures: A survey”. In: *ACM Trans. Model. Comput. Simul.* 13.2 (2003), pp. 134–154.
- [TS12] David Tolpin and Solomon Eyal Shimony. “Semimyopic Measurement Selection for Optimization Under Uncertainty”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 42.2 (2012), pp. 565–579. DOI: 10.1109/TSMCB.2011.2169247. URL: <http://dx.doi.org/10.1109/TSMCB.2011.2169247>.

Paper C

Allocating Planning Effort when Actions Expire

Shahaf S. Shperberg, Andrew Coles, Bence Cserna, Erez Karpas,
Wheeler Ruml, Solomon E. Shimony

The paper has been published in the
AAAI '19: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, 2019
pp. 2379–2386

© 2019 AAI

The layout has been revised.

Abstract

Making plans that depend on external events can be tricky. For example, an agent considering a partial plan that involves taking a bus must recognize that this partial plan is only viable if completed and selected for execution in time for the agent to arrive at the bus stop. This setting raises the thorny problem of allocating the agent’s planning effort across multiple open search nodes, each of which has an expiration time and an expected completion effort in addition to the usual estimated plan cost. This paper formalizes this metareasoning problem, studies its theoretical properties, and presents several algorithms for solving it. Our theoretical results include a surprising connection to job scheduling, as well as to deliberation scheduling in time-dependent planning. Our empirical results indicate that our algorithms are effective in practice. This work advances our understanding of how heuristic search planners might address realistic problem settings.

1 Introduction

Agents that plan and act in the real world must deal with the fact that time passes as they are planning. For example, an agent that needs to get to the airport may have two options: take a taxi, or take a bus. Each of these options can be thought of as a *partial plan* to be elaborated into a complete plan before execution can start. Clearly, the agent’s planner should only elaborate the partial plan that involves taking the bus if it can be elaborated into a complete plan before the bus leaves. Furthermore, consider a second example. When faced with two partial plans that are each estimated to require five minutes of computation to elaborate into complete plans, if only six minutes remain until they both expire, then we would want the planner to allocate all of its remaining planning effort to one of them, rather than to fail on both.

Cashmore et al. 2018 recognized the problem of node expiration in the context of temporal planning with timed initial literals (TIL) (Cresswell and Coddington 2003; Edelkamp and Hoffmann 2004), where the TILs occur at times that are relative to when *planning* starts, rather than to when execution starts. However, their approach to addressing it is relatively superficial in that, after estimating the latest time when execution can start for each search node, this information is used merely to prune

nodes that become infeasible. Such a planner, while handling the first example given above, can fail in our second example. In this paper, we investigate the problem more deeply, explicitly using rational metareasoning (Russell and Wefald 1991) to choose which node to expand. We formalize this metareasoning problem, which we call *Allocating Effort when Actions Expire* (AE2), as an MDP, allowing us to define the optimal solution in a manner similar to Hansen and Zilberstein 2001. We establish close connections between AE2, deliberation scheduling in time-dependent planning (Boddy and Thomas L Dean 1994), and a seemingly unrelated problem in job scheduling with deadlines (Yedidsion 2012). We describe several efficient ways of solving the metareasoning problem, although not necessarily optimally, and evaluate them empirically over several types of distributions. The empirical results suggest that taking estimated node expiration times into account can lead to a better planning strategy.

In this paper, we examine only the one-shot version of the metareasoning problem. Integrating the solutions we present here into a temporal planner can involve solving this problem repeatedly, possibly after each node expansion, in addition to gathering the requisite statistics. These issues are beyond the scope of the current paper. Nevertheless, we devote attention to developing effective metareasoning algorithms that can be useful in practice. When testing our algorithms, we use scenarios based on realistic search trees generated by OPTIC (Benton et al. 2012), the same planner that was adapted in the experiments of Cashmore et al. 2018. Our work provides a firm basis for further efforts to design planners for agents that interact with a wider world containing exogenous processes and other agents, one in which time passes and opportunities can be fleeting.

1.1 Problem Statement

To formalize AE2, we abstract away from any particular planning methodology and merely posit the existence of n computational processes, all attempting to solve the same problem. For example, these may represent promising partial plans for a certain goal, implemented as nodes on the frontier of a search tree. There is a single computing thread or processor to run all the processes, so it must be shared. When process i terminates, it will, with probability P_i , deliver a solution or, otherwise, indicate its failure to find one. For each process, there is a deadline, defined in absolute wall

2. Previous Work

clock time, by which the computation must be completed in order for any solution it finds to be valid, although that deadline may only be known to us with uncertainty. For process i , let $D_i(t)$ be the CDF over wall clock times of the random variable denoting the deadline. Note that the actual deadline for a process is only discovered with certainty when its computation is complete. This models the fact that, in planning, a dependence on an external timed event might not become clear until the final action in the plan is added. If a process terminates with a solution before its deadline, we say that it is *timely*. The processes have performance profiles described by CDFs $M_i(t)$ giving the probability that process i will terminate given an accumulated computation time on that process of t or less. Although some of the algorithms we present may work with dependent random variables, we assume in our analysis that all the variables are independent. Given the $D_i(t)$, $M_i(t)$, and P_i , the objective of AE2 is to schedule processing time over the n processes such that the probability that at least one process finds a solution before its deadline is maximized. This is the essential metareasoning problem in planning when actions expire.

2 Previous Work

There is much related work on planning under time constraints, such as that by Thomas L. Dean et al. 1995. For an appropriate early survey see Garvey and Lesser 1994. In this section we refer only to existing work that is directly used in developing results for AE2: work in deliberation scheduling (Boddy and Thomas L Dean 1994) and job scheduling (Yedidsion 2012). In both of these problems, we have a set of n computational processes that need to be allocated processing time on a single processor. Each process $1 \leq i \leq n$ has a known deadline d_i by which computation in that process must be completed.

2.1 Deliberation Scheduling

In deliberation scheduling for time-dependent planning (Boddy and Thomas L Dean 1994), typically what is being scheduled are anytime algorithms, which exhibit a trade-off between runtime and solution quality (utility). We are thus given a *performance profile*, a mapping $v_i(t_i)$ from the total processing time t_i allocated to process

i to the expected utility value generated by that process. The problem is to find a schedule, mapping time to process number, such that the total expected utility $U = \sum_{i=1}^n v_i(t_i)$ is maximized, subject to the constraint that all processing allocated for process i occurs no later than d_i . The objective function here is not the same as for AE2, and there is no notion of complete failure to find a solution. However, there is a direct mapping between this version of deliberation scheduling and AE2 with known deadlines, which we mention and exploit later on.

Although the problem is NP-hard, in the special case of *diminishing returns*, it can be solved optimally in polynomial time. The diminishing returns requirement is that the returns slope $\frac{dv_i(t)}{dt}$ be non-increasing in t . We define a similar notion of diminishing returns for AE2 below.

Boddy and Thomas L Dean 1994 present a deliberation scheduling algorithm for diminishing returns piecewise linear performance profiles. It scans from the **latest** deadline backwards. In the current inter-deadline segment, select, from all profiles whose deadline has **not** expired, the profile i with the greatest slope. Then allocate to process i time sufficient to exhaust this slope segment or to reach the previous deadline, whichever is first. When an earlier deadline is reached, additional profiles become relevant, which may introduce profiles with a better slope. Upon scan is completion, re-arrange the schedule into contiguous segments, in order of deadlines.

2.2 Job Scheduling

The minimum tardiness job scheduling problem (Yedidsion 2012) differs from deliberation scheduling in that all processes must be run to completion, and no uncertainty is involved. However, a process i can run faster by paying a cost c_i , modeling the allocation of additional resources. The total job i runtime is a known function $T_i(c)$ of the cost. The goal is to find costs and a schedule such that the sum of the costs is minimized, while ensuring that all jobs finish before their respective deadlines. (Actually, Yedidsion 2012 allows processes to run beyond the deadline by a certain *tardiness* value that is either to be constrained or minimized, but for our purposes we need only refer to the variant that constrains the maximum tardiness to be zero.)

In job scheduling, a polynomial-time scheme is possible if the the speedup is a diminishing-returns function of the cost, i.e. if $T'_i(c) = \frac{dT_i(c)}{dc}$ is an always non-

3. The Deliberation Scheduling MDP

decreasing function of c . Note that $T_i(c)$ is non-negative and $T'_i(c) < 0$.

In the case of diminishing returns, it was shown that there exists an optimal schedule such that

- A) time allocations are contiguous, i.e. the schedule can be represented as a list of start-length pairs (s_i, l_i) ;
- B) the allocations are in order of the respective deadlines. We assume w.l.o.g. that $s_1 \leq s_2 \leq \dots \leq s_n$; and
- C) the performance slopes $T'_i(c)$ obey $T'_i(c_i) \geq T'_{i+1}(c_{i+1})$ for all $1 \leq i < n$.

As discussed below, these properties also hold for diminishing returns in AE2 (with property C adapted as discussed below). Using these properties, Yedidsion 2012 provides an algorithm to find an optimal schedule in polynomial time, with a complexity depending on speedup profiles T_i representation. An analytical representation manipulable in $O(1)$ is assumed therein, resulting in a runtime $O(n)$.

3 The Deliberation Scheduling MDP

We now address the AE2 problem of deliberation scheduling with uncertain deadlines. For simplicity, we initially assume that time is discrete and the smallest unit of time is 1. Allowing continuous time is more complex because one needs to define what is done if some time-slice is allocated to a process i , and that process terminates before the end of the time-slice. Discretization avoids this complication.

We can now define our deliberation scheduling problem as an MDP, with distributions represented by their discrete probability function (pmf). Denote $m_i(t) = M_i(t) - M_i(t-1)$, the probability that process i completes after exactly t time units of computation time, and $d_i(t) = D_i(t) - D_i(t-1)$, the probability that the deadline for process i is exactly at time t . Without loss of generality, we can assume that $P_i = 1$: otherwise modify the deadline distribution for process i to have $d_i(-1) = 1 - P_i$, simulating failure of the process to find a solution at all with probability $1 - P_i$, and multiply all other $d_i(t)$ by P_i . This simplified problem we call SEA2. We formalize the SEA2 MDP as an indefinite duration MDP with terminal states, where we keep track

of time as part of the state. (An alternate definition would be as a finite-horizon MDP, given a finite value d for the last possible deadline.)

The actions in the MDP are: assign the next time unit to process i , denoted by a_i with $i \in [1, n]$. We allow action a_i only if process i has not already failed.

The state variables are the wall clock time T and one state variable T_i for each process, with domain $\mathcal{N} \cup \{F\}$. T_i denotes the cumulative time assigned to each process i until the current state, or that the process has completed computation and resulted in failure to find a solution within the deadline. We also have special terminal states SUCCESS and FAIL. Thus the state space is:

$$\mathcal{S} = (\text{dom}(T) \times \bigtimes_{1 \leq i \leq n} \text{dom}(T_i)) \cup \{\text{SUCCESS}, \text{FAIL}\}$$

The initial state is $T = 0$ and $T_i = 0$ for all $1 \leq i \leq n$.

The transition distribution is determined by which process i has last been scheduled (the action a_i), and the M_i and D_i distributions. If all processes fail, transition into FAIL with probability 1. If some process is successful, transition into SUCCESS with probability 1. More precisely:

- The current time T is always incremented by 1.
- Accumulated computation time is preserved, i.e. for action a_i , $T_j(t+1) = T_j(t)$ for all processes $j \neq i$.
- $T_i(t) = F$ always leads to $T_i(t+1) = F$.
- For action a_i (assign time to process i), the probability that process i 's computation is complete given that it has not previously completed is $P(C_i) = \frac{m_i(T_i+1)}{1-M_i(T_i)}$. If completion occurs, the respective deadline will be met with probability $1 - D_i(T_i)$. Therefore, transition probabilities are: with probability $1 - P(C_i)$ set $T_i(t+1) = T_i(t) + 1$, with probability $P(C_i)D_i(T_i)$ set $T_i(t+1) = F$ (process i failed to meet its deadline), and otherwise (probability $P(C_i)(1 - D_i(T_i))$) transition into SUCCESS (the value of T_i in this case is 'don't care').
- If $T_i(t+1) = F$ for all i , transition into FAIL.

The reward function is 0 for all states, except SUCCESS, which has a reward of 1.

3.1 Solution Complexity and Approximations

Solving the SEA2 MDP implies finding an optimal conditional (also called *adaptive*) policy, which is a hard computational problem because the state space of the MDP is exponential in n . We show below that the problem is NP-hard, but conjecture that it is PSPACE-complete. As this is a meta-reasoning problem, we cannot afford to devote a lot of computational resources to solving it; instead we attempt to solve easier problems that may give approximately optimal solutions, or optimal solutions to special cases. We begin by considering the optimal *linear* policy. A linear allocation policy is simply a (possibly infinite) sequence A of integers, where $A[t] = i$ means ‘assign time slice t to process i .’

There are two possible ways to execute the linear policy. The *basic* scheme (also called “batch”, or “non-adaptive”) simply executes process $A[t]$ at time t , except when that process has already failed, in which case we leave the processor idle. Upon success, we stop all the computations. Clearly, a better expected performance would result from reallocating the time allocated to an already failed process to the next process in the sequence that has not failed yet. We call this execution method the *semi-adaptive* execution scheme. The semi-adaptive scheme is easy to implement during execution, but hard to analyze, therefore our analysis below is for the basic execution scheme. In some special cases discussed below, the basic and semi-adaptive schemes are equivalent.

As shown below, finding the optimal linear policy is also NP-hard, although it may be easy to approximate using a greedy scheme. However, even if the optimal linear policy can be found, it is not necessarily an optimal solution to the MDP, as shown in the following counter-example.

Example 1: We are given processes $\{1, 2, 3\}$ with deadline distributions $d_1 = [0.5 : -1, 0.5 : 2]$, $d_2 = [0.5 : -1, 0.5 : 4]$, and $d_3 = [0.4 : -1, 0.6 : 4]$. The computation completion time distributions are: $m_1 = [0.1 : 1, 0.9 : 2]$, $m_2 = [1 : 2]$, and $m_3 = [1 : 3]$, the latter two being degenerate distributions, i.e. known runtimes times.

The optimal linear policy is $A_{12} = (1, 1, 2, 2, \dots)$, where the subscript denotes the processes to which time is allocated. A_{12} delivers a timely solution if either process 1 succeeds (its deadline is not -1) or if process 1 fails yet process 2 succeeds, so we get $P_{12} = 0.5 + (0.5 \times 0.5) = 0.75$. To see that A_{12} is optimal, consider the alternatives.

Sequence $A_{13} = (1, 1, 3, 3, 3, \dots)$ succeeds only if process 1 succeeds, or if it fails after 1 time unit and process 3 succeeds. If process 1 takes too long to fail, process 3 will not meet its deadline. We get: $P_{13} = 0.5 + 0.5 \times 0.1 \times 0.6 = 0.53$. Any sequence A_3 starting with process 3 succeeds only if process 3 succeeds, with $P_3 = 0.6$. All other sequences are dominated by A_3 , A_{13} , or A_{12} , thus A_{12} is the (linear) optimum.

However, the following *adaptive* policy is better than A_{12} : Run process 1 for one time unit. If it terminates with success, then we are done. If it terminates with failure, then run process 3. If it has not terminated, run process 1 for one more unit and allocate the two subsequent time units to process 2. The probability of a timely successful solution for this policy is greater than P_{12} , because in the case where process 1 fails in one time unit, we take advantage of this knowledge to run process 3 instead of process 2 because it has a higher probability of success and can still finish within its deadline.

Whether linear plans are *near-optimal* is an open problem. However, we show below that in the special case where the deadlines are known, the optimal linear policy is also an optimal solution to the MDP. Unfortunately, finding an optimal linear policy remains NP-hard. But if, in addition, the performance profile obeys a certain diminishing returns criterion, the optimal linear policy (and thus the true optimal policy) can be found in polynomial time. Although realistic cases do not necessarily exhibit these properties, we will see that the solution to the special case can be used effectively as part of an efficient heuristic solution to the MDP.

We begin by formulating the objective function in the cases of a linear policy and a linear *contiguous* policy. For simplicity, we provide the equations for the *basic* execution scheme. For a given scheduled sequence A , we define an *individual allocation* for a task i as a function A_i from non-negative integers to $\{0, 1\}$. $A_i(t)$ is 1 if time slot t allocated for process i and 0 otherwise. Denote by $S_i(t)$ the total number of time slots allocated to i on or before t , i.e.: $S_i(t) = \sum_{t'=1}^t A_i(t')$. Then the probability that process i under allocation A_i will find a timely solution is

$$PS_i(A_i) = \sum_t A_i(t) m_i(S_i(t)) (1 - D_i(t))$$

In a *contiguous* linear policy, $A_i(t)$ contains only a single contiguous block of 1's. A policy can then be represented as an array of (s_i, l_i) pairs denoting (start, length) of

3. The Deliberation Scheduling MDP

the processing time, respectively, and we can re-write PS_i as

$$PS_i(A_i = (s_i, l_i)) = \sum_{t=0}^{l_i} m_i(t)(1 - D_i(t + s_i))$$

With this representation, the problem becomes: Find an array of pairs $A_i = (s_i, l_i)$ maximizing

$$P_{Succ} = 1 - \prod_i (1 - PS_i(A_i))$$

Optimization of a product term is inconvenient. We can equivalently minimize probability of failure P_{FAIL} , which is $1 - P_{Succ}$. Taking logarithms, we seek to minimize:

$$\log(P_{FAIL}) = \log\left(\prod_i (1 - PS_i(A_i))\right) = \sum_i \log(1 - PS_i(A_i)) \quad (C.1)$$

An important special case is known deadlines, where we can re-write the probability of success for process i as:

$$PS_i(A_i) = \sum_{t \leq d_i} A_i(t) m_i(S_i(t)) = \sum_{t \leq S_i(d_i)} m_i(t) = M_i(S_i(d_i)) \quad (C.2)$$

because $1 - D_i(t) = 1$ for all $t \leq d_i$ and zero for $t > d_i$. Using this probability of success in Equation C.1 and using $LF_i(\cdot)$ to denote the log probability of failure, $\log(1 - M_i(\cdot))$, the objective function to minimize becomes

$$\log(P_{FAIL}) = \sum_i LF_i(S_i(d_i)) \quad (C.3)$$

Note that in this case of known deadlines, we can map SEA2 into an equivalent (time-dependent planning) deliberation scheduling problem simply by creating performance profiles $v_i(t) = -LF_i(t)$, and keeping the same deadlines. Additionally, if all the known deadlines are also *equal*, we now have a problem equivalent to that of allocating runtimes in *algorithm portfolios* (Gomes and Selman 2001), where we wish to maximize the probability that some algorithm in the portfolio can find a solution before the (common) time limit per problem instance.

Theorem C.1

In SEA2 with known deadlines, the optimal linear contiguous policy is an optimal policy for the MDP.

Proof (outline): Consider any SEA2 MDP policy for known deadlines, represented as a decision tree. Each state-node has a single action edge, leading to a stochastic

(chance) node. Each chance node has one or more outgoing edges, each leading to a state node. Given a (state, action) pair $((T, T_1 \dots T_n), a_i)$, there are two possible cases. If $T \geq d_i$, the deadline for process i is past, and process i fails to find a timely solution (with probability 1): the respective chance node is degenerate, and has only *one* next state with $T_i = F$. If $T < d_i$, the following chance node has only *two* outgoing edges: either process i terminates successfully (reaching a terminal state SUCCESS), or does not terminate (and the next state has T and T_i incremented by 1). That is, only the latter *one* edge leads to a node that can have outgoing action edges. Thus, a maximum-length path in the tree determines a unique sequence of actions A , which is a linear policy equivalent to this MDP policy.

Now consider any linear policy. The probability of success for process i for known deadlines (Equation C.2) depends only on the total processing time given to process i before its deadline d_i . Thus, any schedule that allocates processing time beyond the deadline (called a *tardy* schedule) can be improved (or at least made no worse) by re-allocating such processing time to some other process. We thus consider below only non-tardy schedules.

Under our SEA2 simplifying assumption that $P_i = 1$, if a process completes the computation this ends in SUCCESS. Since Equation C.2 depends only on the *total* amount of computation time (before the deadline) for process i , rearranging the schedule by moving allocations makes no difference as long as the resulting schedule remains non-tardy. Therefore the schedule can be rearranged to make allocations contiguous, by making them appear in the same order as the deadlines (i.e. $s_i \leq s_j$ is $d_i \leq d_j$). Thus the optimal contiguous linear sequence is also an optimal solution to the MDP. \square

Note that for non-tardy schedules with certain deadlines, since a completed computation never fails, the simple execution scheme and the semi-adaptive execution scheme are equivalent. Although it is nice to know the form of optimal policies for known deadlines, we have:

Theorem C.2

For a compact representation of the distributions, finding the optimal contiguous linear SEA2 policy in the case of known deadlines is NP-hard.

Proof (outline): by reduction from the optimization version of knapsack ((Garey and

3. The Deliberation Scheduling MDP

Johnson 1979) problem [MP9]):

Definition C.1 (Knapsack problem)

Given a set of items $\mathcal{S} = \{s_1, \dots, s_n\}$, each with a positive integer weight w_i and value v_i , a weight limit W , find a subset S of \mathcal{S} such that the total weight of S is at most W with a maximal total value.

In the reduction, each process represents an item in the Knapsack problem. We only need degenerate performance profiles: for each process i , let its deadline be $d_i = W$, and its performance profile M_i be a piecewise constant function:

$$M_i(t) = \begin{cases} 0, & t < w_i \\ \epsilon v_i, & w_i \leq t \leq W \\ 1, & W < t \end{cases}$$

ϵ is chosen such that the probability of success of at least one process is "almost" as much as the sum of success probabilities of all the selected processes. Let $H = \max_{i=1}^n v_i$. Setting $\epsilon \leq \frac{1}{2H^2n^3}$, we can show that every set of items (processes) S we have:

$$\epsilon \sum_{s_i \in S} v_i \geq P_{succ}(S) = 1 - \prod_{s_i \in S} (1 - \epsilon v_i) > \epsilon \left(\sum_{s_i \in S} v_i - 1 \right)$$

Let S be an optimal contiguous schedule. W.l.o.g. we can assume that the processing time assigned to each process $s_i \in S$ is equal to w_i , and that the sum of processing times is at most W . Abusing the notation, we use S to also denote the set of items s_i in the Knapsack problem corresponding to the processes assigned time w_i in S . The Knapsack value of S is $V = \sum_{s_i \in S} v_i$. By construction, the sum of weights of the items in S is at most W , so S is a Knapsack solution.

Assume, in contradiction, that S is suboptimal. Then exists an (integer) Knapsack solution S' , with value $V' \geq V + 1$. Taking S' as a schedule (assigning time w_i to each process $s_i \in S'$) creates a schedule where all processes run before time W as well, with success probability:

$$\begin{aligned} P_{succ}(S') &= 1 - \prod_{s_i \in S'} (1 - \epsilon v_i) > \epsilon \left(\sum_{s_i \in S'} v_i - 1 \right) \\ &\geq \epsilon \left(\sum_{s_i \in S} v_i \right) \geq P_{succ}(S) \end{aligned}$$

that is, schedule S' has a greater success probability than S , a contradiction. \square

Note that the seemingly obvious proof of using the mapping between our deliberation scheduling and the NP-hard equivalent deliberation scheduling in time-dependent planning does not generate a correct NP-hardness proof, as the mapping involves an exponent. This results in number descriptions of exponential size given the size of the description of the problem. Due to Theorem C.1, this result (Theorem C.2) also implies NP-hardness of the linear (not necessarily contiguous) policy, as well as the NP-hardness of solving the general SEA2 MDP. Observe that a compact representation of the MDP is assumed, rather than a complete transition distribution array, and that solving the MDP in the general case may even be PSPACE-hard (conjecture).

As the linear policy optimization is NP-hard, we now consider further restrictions, and in particular consider the case of diminishing logarithm of returns in the performance profile. That is, suppose that $LF_i(t+1) - LF_i(t)$ is a non-decreasing function of t for all i .

Theorem C.3

The optimal SEA2 policy for the case of known deadlines and diminishing logarithm of returns can be computed in polynomial time.

Proof (outline): Although it seems to be a different problem entirely, the theorem follows from results on job scheduling with diminishing returns (Yedidsion 2012). The problems are equivalent when we map log probability of failure into costs, both of which need to be minimized. We set the speedup function such that $T_i(LF_i(t)) = T_i(\log(1 - M_i(t))) = t$, i.e. set $T_i(c) = \exp(1 - M_i^{-1}(c))$, where M_i^{-1} is the inverse function of M_i . Then use any algorithm for the job scheduling problem to get an optimal solution and map it back to SEA2. \square

If we have an analytical representation of $LF_i(t)$, we can use the above conversion and the algorithm of Yedidsion 2012 directly. Likewise, if our distribution $LF_i(t)$ is piecewise linear diminishing returns, we can use the algorithm for deliberation scheduling in time-dependent planning (Boddy and Thomas L Dean 1994).

3.2 SEA2 with Diminishing Returns

As in general the performance profile may not be provided analytically, and may also not be piecewise linear, we adapt the above ideas to apply to the discrete distribution

3. The Deliberation Scheduling MDP

representation that we have in SEA2. Properties A and B of an optimal job schedule hold in SEA2. The solution property C for the discrete case is slightly different, the inequality needed here is:

$$LF_i(l_i) - LF_i(l_i - 1) \geq LF_{i+1}(l_{i+1} + 1) - LF_{i+1}(l_{i+1})$$

Proving this condition is similar to the proof for the continuous case. That is, if condition C does not hold, then we have for some i :

$$LF_i(l_i) - LF_i(l_i - 1) < LF_{i+1}(l_{i+1} + 1) - LF_{i+1}(l_{i+1})$$

where modifying the deliberation schedule to have: $l_i \leftarrow l_i - 1$, $l_{i+1} \leftarrow l_{i+1} + 1$, $s_{i+1} \leftarrow s_{i+1} - 1$, decreases logarithm of probability of failure, without causing it to be tardy.

In the algorithm, which we call ScheduleDR, we keep a queue Q containing the current active profiles, and for each profile the currently allocated time l_i . Q is kept sorted in non-decreasing order of the gain $g_i(t) = LF_i(t) - LF_i(t - 1)$. Psuedo-code is given in Algorithm 1.

By construction, the complexity of ScheduleDR is $O((n + d) \log n)$, where $d = \max_{i=1}^n d_i$, the latest deadline. Note that this algorithm is essentially the same as the deliberation scheduling for piecewise linear profiles (Boddy and Thomas L Dean 1994), adapted to the discrete case.

3.3 Non-diminishing returns

In general, as well as in the planning application, the M_i distributions do not have diminishing returns, as we expect $M_i(t)$ to be near zero until some critical value of t (the expected planning time for process i , also called “startup time”), and then quickly increase, followed by a region that behaves according to a diminishing returns rule. An additional complication is that the deadlines can be stochastic. Therefore, we cannot directly use ScheduleDR.

Nevertheless, if the time allocated to each process is at least equal to the critical value, thereby reaching the diminishing returns region, it is still possible to use the algorithm for diminishing returns, as long as we make sure that the algorithm does not ignore processes that have a significant startup time. Therefore, we can convert

Algorithm 1: Scheduling for Diminishing Returns

```

1 ScheduleDR(Profiles)
2 For all  $1 \leq i \leq n$ , let  $l_i = 0$ 
3 Let  $Q = \emptyset$ ;  $Order = \emptyset$ 
4 for  $t = \max_{i=1}^n d_i$  downto 0 do
5     for each  $i$  for which  $d_i = t$  do
6         Insert profile  $i$  into  $Q$ 
7         Prepend  $i$  to  $Order$ 
8     Retrieve profile  $j$  from  $Q$ 
9     Increment  $l_j$ , and re-insert into  $Q$ 
10 Let  $t = 0$ 
11 for  $j$  from 1 to  $n$  do
12     Let  $s_{Order[j]} = t$ 
13     Let  $t = t + l_{Order[j]}$ 
14 Return the pairs  $(s_i, l_i)$  of all profiles.
    
```

such profiles that have a startup time into diminishing returns by modifying them to have a rampup starting at 0 (see below).

Under the assumption that all processing time is allocated to process i , starting at time 0, the success distribution for process i is:

$$f_i(t) = PS_i(A_i = (0, t)) = P_i \sum_{t'=0}^t m_i(t')(1 - D_i(t'))$$

Define the *most effective computation time* for process i under this assumption to be:

$$e_i = \operatorname{argmin}_t \frac{\log(1 - f_i(t))}{t}$$

Now, the function M_i can be modified in the region from 0 to e_i to have linear slope in probability of failure. That is, set:

$$LF_i(t) = t \frac{\log(1 - M_i(e_i))}{e_i}$$

for all $0 \leq t \leq e_i$. We now have performance profiles with initially diminishing returns. (For some unimodular distribution this actually results in diminishing returns for all t .)

3. The Deliberation Scheduling MDP

We also need to handle the uncertain deadlines. We do so by setting a deadline proxy value $d_i(th)$ as a function of D_i and a “confidence level threshold” $0 \leq th \leq 1$, defined by: $d_i(th) = \text{the first } t \text{ such that } D_i(t) \geq th$. Another possibility is to use the expected value of the deadline as if it were a known deadline.

After making these modifications, we can use the algorithm for diminishing returns to create a schedule that is optimal for the modified profiles and the proxy deadlines. However, the resulting schedule may be very far from optimal, as in fact the time allocated for many processes can be significantly lower than its most effective computation time. We have attempted such a solution, and the empirical results were not good. Therefore, other than mentioning these negative results for completeness, we do not further elaborate this method. Nevertheless, some of the intuitions and definitions from this section are still usable in the better-performing real-time greedy scheme we describe below.

3.4 Real-time Deliberation Scheduling

Faster meta-reasoning is achievable if we try to allocate computation time just for the first process to run, and defer the rest of the scheduling. This makes sense as in practice the initial computations actually may provide additional information, which the full scheduling does not take into consideration. Note that algorithm `ScheduleDR()` tends to allocate computation time for a process that has an early deadline, but may decide to throw it out (allocate it zero processing time) if its performance slope is too low or it is unlikely to complete computing before its deadline.

The intuition from the diminishing returns optimization is thus to prefer the process i that has the best utility per time unit. However, it is also important to allocate the time now to a process i that has a deadline as early as possible, as this is most critical. We therefore suggest the following greedy algorithm. Whenever assigning computation time, allocate t_d units of computation time to process i that maximizes:

$$Q(i) = \frac{\alpha}{E[D_i]} - \frac{\log(1 - f_i(e_i))}{e_i} \quad (\text{C.4})$$

where α and t_d are positive empirically determined parameters, and $E[D_i]$ is the expectation of the D_i distribution, which we use as a proxy for “deadline of process i ”. The α parameter trades off between preferring earlier expected deadlines (large α)

and better performance slopes (small α).

4 Empirical Evaluation

In order to empirically evaluate our scheduling methods, we generated several types of performance profiles and deadline distributions based on the following distributions: Uniform (U), with minimal range value $a = 1$ and maximal range value b uniformly drawn from $\{[5, 10], [50, 100], [100, 200], [150 - 300]\}$, we denote the set of possible $[a - b]$ ranges by R ; Boltzmann (B), truncated exponential distribution with the diminishing return property, using a $\lambda \in \{0.1, 1, 2\}$ and range drawn from R ; Truncated Normal Distribution (N) with $\mu \in \{5, 50, 100, 150\}$, $\sigma \in \{1, 5, 10\}$, and range drawn from R ;

Finally, we used distributions collected from search trees of the Robocup Logistics League (Niemueller et al. 2015) domain generated by the OPTIC planner (denoted by P in the table). To acquire the distribution, A^* was executed from each node of the dumped search tree. The result of each of these searches provides the number $N(v)$ of expansions necessary to find the goal under a node v . These numbers were binned separately for each $(h(v), g(v))$ pair. Then, a number of nodes V in the tree was selected randomly, each one standing for a process. For each such $v \in V$, the list of numbers of expansion in the bin corresponding to $g(v)$ and $h(v)$ was treated as a distribution over completion times (in terms of number of expansions). Likewise for creating the latest start times for the resulting plan (the deadline distribution). When using our method as part of a planner, one would need to create such statistics on-the-fly.

Experiments were both with unknown deadline (UK) or with a known deadline (K) which was randomly drawn from the corresponding distribution before the execution.

We compared the results of ScheduleDR and the greedy scheme to several naive schemes: (1) random - allocate time to a random process that did not already fail; Most promising plan (MPP) - allocate time to the plan with the highest probability to finish successfully and meet the deadline, in case of failure, the algorithm chooses the next most promising plan and repeats the process; Round-robin (RR) - allocate time

4. Empirical Evaluation

units to each non-failed process in equal portions and in circular order. Whenever possible, we also compared with the optimal MDP solution computed using value-determination of the Bellman equation. Unfortunately the space required for the MDP is $O(d^n)$ just for enumerating the state space, even with a compact representation of the transition distribution. Although the runtime is $O(d^{n+1})$, space was the main limiting factor, so we could only compute the optimal score for a few of the smallest instances. We tested ScheduleDR using $th \in \{0.2, 0.5, 0.7, 1\}$ and greedy with $\alpha \in \{0, 0.2, 0.5, 1, 20\}$, $td \in \{1, 5, 10\}$. However, the reported results include only the best configuration for each algorithm: $th = 0.5$, $\alpha = 0$, and $td = 1$.

Evaluating the quality of a solution (policy) is not a trivial task, especially for adaptive policies which depend on the state to make a decision. In order to tackle this issue, we ran the algorithms on each setting for 500 attempts and reported the fraction of successful runs out of the total number of attempts as the solution quality. The results are shown in Table E.1. The Q column indicates the solution quality (probability of success using the policy created by the algorithms); the T column is the runtime in seconds. The rows denoted "average" are average solution quality and geometric mean of the runtimes. As expected, ScheduleDR had the best performance when using known deadlines with diminishing returns performance profiles (B). However, in most other cases ScheduleDR performed poorly, especially when the deadlines were unknown. Greedy resulted in the best policy in most cases, and the best average solution quality for both known and unknown deadlines. Overall, the greedy scheme demonstrates a significant improvement over the naive schemes in terms of solution quality. Nonetheless, greedy had the worst runtime out of all other algorithms (except for the MDP) with an average runtime of 1.24 seconds. Note that the time reported is the total time for an *entire* policy evaluation, therefore, it includes hundreds of decisions at different points. Thus, despite being the slowest of the efficient algorithms, the greedy scheme is sufficiently fast to be used for metareasoning.

Paper C. Allocating Planning Effort when Actions Expire

K / UK	Dist	# pr	MDP		Greedy		ScheduleDR		MPP		RR		Random	
			Q	T	Q	T	Q	T	Q	T	Q	T	Q	T
K	B	2	0.67	0.05	0.61	0.01	0.67	0.00	0.70	0.00	0.55	0.00	0.54	0.00
		5			0.72	0.04	0.82	0.00	0.61	0.00	0.54	0.00	0.57	0.00
		10			0.60	0.07	0.88	0.00	0.71	0.00	0.48	0.00	0.51	0.00
		100			0.81	0.74	0.99	0.01	0.91	0.03	0.62	0.00	0.60	0.00
	N	2	0.61	7.49	0.56	0.01	0.45	0.00	0.33	0.00	0.04	0.00	0.07	0.00
		5			0.83	0.02	0.72	0.01	0.27	0.00	0.01	0.00	0.03	0.00
		10			0.93	0.04	0.41	0.01	0.09	0.00	0.00	0.00	0.03	0.00
		100			1.00	0.20	0.70	0.03	0.23	0.02	0.00	0.00	0.00	0.00
	U	2	0.68	1.20	0.61	0.04	0.65	0.01	0.50	0.00	0.47	0.00	0.48	0.00
		5			0.90	0.13	0.88	0.01	0.75	0.00	0.49	0.00	0.47	0.00
		10			0.98	0.28	0.98	0.01	0.66	0.01	0.44	0.00	0.44	0.00
		100			1.00	2.36	1.00	0.02	0.80	0.03	0.42	0.00	0.45	0.00
	P	2			0.72	0.02	0.79	0.00	0.01	0.00	0.01	0.00	0.04	0.00
		5			0.78	0.06	0.81	0.07	0.79	0.02	0.38	0.02	0.54	0.02
		10			1.00	0.05	0.87	0.10	0.99	0.00	0.85	0.01	0.82	0.01
		100			1.00	0.42	0.91	0.25	0.86	0.04	0.00	0.03	0.04	0.05
	Known Average				0.82	0.08	0.78	0.01	0.58	0.00	0.33	0.00	0.35	0.00
UK	B	2	0.68	147.34	0.61	0.07	0.35	0.00	0.64	0.00	0.59	0.00	0.57	0.00
		5			0.65	0.18	0.36	0.00	0.63	0.01	0.60	0.00	0.60	0.00
		10			0.70	0.45	0.45	0.00	0.66	0.04	0.62	0.00	0.62	0.00
		100			0.70	5.45	0.44	0.01	0.65	0.68	0.58	0.00	0.61	0.00
	N	2	0.66	26.95	0.63	0.07	0.37	0.01	0.20	0.00	0.14	0.00	0.13	0.00
		5			0.70	0.19	0.35	0.01	0.09	0.00	0.02	0.00	0.06	0.00
		10			0.65	0.41	0.30	0.01	0.15	0.01	0.00	0.00	0.02	0.00
		100			0.76	4.02	0.32	0.05	0.06	0.05	0.00	0.00	0.00	0.00
	U	2	0.73	103.28	0.68	0.33	0.39	0.01	0.53	0.01	0.54	0.00	0.55	0.00
		5			0.70	1.25	0.43	0.01	0.57	0.03	0.43	0.00	0.45	0.00
		10			0.78	2.07	0.46	0.02	0.59	0.05	0.47	0.00	0.44	0.00
		100			0.86	16.56	0.52	0.03	0.59	0.16	0.43	0.00	0.44	0.00
	P	2			0.61	0.01	0.24	0.00	0.46	0.00	0.47	0.00	0.52	0.00
		5			0.90	0.05	0.54	0.04	0.45	0.03	0.56	0.03	0.59	0.06
		10			0.90	0.45	0.32	0.06	0.62	0.06	0.60	0.04	0.62	0.07
		100			0.85	3.54	0.77	0.13	0.38	0.01	0.20	0.89	0.33	0.50
	Unknown Average				0.73	0.47	0.41	0.01	0.45	0.02	0.39	0.00	0.41	0.00
	Total Average				0.77	0.20	0.60	0.01	0.52	0.01	0.36	0.00	0.38	0.00

Table C.1: Solution quality and runtime of the algorithms on different settings

5 Discussion

This paper defined a deliberation scheduling problem that models optimization of the probability of success for finding a timely plan that depends on external events. This dependence can cause a plan of action to expire, and thus our deliberation scheduling problem is similar to that of time-dependent planning, though with a different objective function. In fact, there is a direct mapping between these deliberation scheduling problems that we can exploit. However, in our case the time at which the actions expire (also called deadlines) are uncertain, adding complexity to our version of the problem. An additional surprising connection exists to job scheduling, where some results similar to those in time-dependent planning are useful.

We introduce a formal MDP model of our deliberation scheduling problem, and analyze its complexity. As solving the MDP is computationally hard, we examine the possibility to provide simple ("linear") schedules as a solution, rather than a full MDP policy. Such solutions are shown by counter-example to be suboptimal, except when the problem is restricted to known deadlines, in which case the optimal constant schedule is also an optimal solution to the MDP. Unfortunately we show that even finding the optimal simple schedule is NP-hard. By examining the relationship to time dependent planning and to job scheduling, we can use similar results for the further restricted case of an appropriate form of diminishing returns, where an optimal solution is possible in low-order polynomial time (Boddy and Thomas L Dean 1994; Horvitz 2001; Yedidsion 2012).

As the restrictions that allow polynomial-time optimal solutions usually do not hold in practice, we develop algorithms that use intuitions from the special case. These are evaluated empirically; one of them, a greedy scheduling algorithm, seems to be close to optimal for many distributions.

Nevertheless, several issues remain open, both theoretical and practical. Some immediate theoretical questions are: Can the optimal policy be approximated in polynomial time within a small constant factor (whether multiplicative or additive)? What is the actual complexity class of the deliberation-scheduling MDP? On the practical side, faster algorithms with good practical results, are needed. **Dynamic** algorithms are especially important due to the main motivation of our problem, which comes

from allocating time for search. For example, the different “processes” could actually represent different nodes in a planner’s search for a timely plan. In this case, however, nodes are added (and possibly pruned) during the search, thereby adding and deleting processes that need to be scheduled, in some cases modifying node statistics. The allocation effort thus needs to be very fast, but may take advantage of there being only a few changes in the setup each time the search effort is reallocated. An adaptation of our greedy scheme is likely to be applicable, but additional research is required before it can be fully integrated into an existing planner.

Acknowledgements

Partially supported by ISF grant #844/17, and by the Frankel center for CS at BGU. Project also funded by the European Union’s Horizon 2020 Research and Innovation programme under Grant Agreement No. 730086 (ERGO).

References of Paper C

- [BCC12] J. Benton, Amanda Jane Coles, and Andrew Coles. “Temporal Planning with Preferences and Time-Dependent Continuous Costs”. In: *ICAPS*. Atibaia, São Paulo, Brazil: AAAI Press, 2012, pp. 2–10.
- [BD94] Mark Boddy and Thomas L Dean. “Deliberation scheduling for problem solving in time-constrained environments”. In: *Artificial Intelligence* 67.2 (1994), pp. 245–285. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/0004-3702\(94\)90054-X](https://doi.org/10.1016/0004-3702(94)90054-X). URL: <http://www.sciencedirect.com/science/article/pii/000437029490054X>.
- [Cas+18] Michael Cashmore et al. “Temporal Planning while the Clock Ticks”. In: *ICAPS*. AAAI Press, 2018, pp. 39–46. URL: <https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17724>.
- [CC03] Stephen Cresswell and Alexandra Coddington. “Planning with Timed Literals and Deadlines”. In: *Proceedings of 22nd Workshop of the UK Planning and Scheduling Special Interest Group*. 2003, pp. 23–35.
- [Dea+95] Thomas L. Dean et al. “Planning under Time Constraints in Stochastic Domains”. In: *Artif. Intell.* 76.1-2 (1995), pp. 35–74. DOI: 10.1016/0004-3702(94)00086-G. URL: [https://doi.org/10.1016/0004-3702\(94\)00086-G](https://doi.org/10.1016/0004-3702(94)00086-G).
- [EH04] Stefan Edelkamp and Jörg Hoffmann. *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition*. Tech. rep. 195. University of Freiburg, 2004.
- [GJ79] M. R. Garey and D. S. Johnson. “Computers and Intractability, A Guide to the Theory of NP-completeness”. In: W. H. Freeman and Co., 1979, p. 190.
- [GL94] Alan Garvey and Victor R. Lesser. “A Survey of Research in Deliberative Real-Time Artificial Intelligence”. In: *Real-Time Systems* 6.3 (1994), pp. 317–347. DOI: 10.1007/BF01088630. URL: <https://doi.org/10.1007/BF01088630>.

- [GS01] Carla P. Gomes and Bart Selman. “Algorithm portfolios”. In: *Artif. Intell.* 126.1-2 (2001), pp. 43–62. DOI: 10.1016/S0004-3702(00)00081-3. URL: [https://doi.org/10.1016/S0004-3702\(00\)00081-3](https://doi.org/10.1016/S0004-3702(00)00081-3).
- [Hor01] Eric Horvitz. “Principles and applications of continual computation”. In: *Artif. Intell.* 126.1-2 (2001), pp. 159–196.
- [HZ01] Eric A. Hansen and Shlomo Zilberstein. “Monitoring and control of any-time algorithms: A dynamic programming approach”. In: *Artif. Intell.* 126.1-2 (2001), pp. 139–157. DOI: 10.1016/S0004-3702(00)00068-0. URL: [https://doi.org/10.1016/S0004-3702\(00\)00068-0](https://doi.org/10.1016/S0004-3702(00)00068-0).
- [NLF15] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. “The RoboCup Logistics League as a Benchmark for Planning in Robotics”. In: *WS on Planning and Robotics (PlanRob) at Int. Conf. on Aut. Planning and Scheduling (ICAPS)*. 2015.
- [RW91] Stuart J. Russell and Eric Wefald. “Principles of Metareasoning”. In: *Artif. Intell.* 49.1-3 (1991), pp. 361–395. DOI: 10.1016/0004-3702(91)90015-C. URL: [http://dx.doi.org/10.1016/0004-3702\(91\)90015-C](http://dx.doi.org/10.1016/0004-3702(91)90015-C).
- [Yed12] Liron Yedidsion. “Bi-criteria and tri-criteria analysis to minimize maximum lateness makespan and resource consumption for scheduling a single machine”. In: *J. Scheduling* 15.6 (2012), pp. 665–679. DOI: 10.1007/s10951-012-0290-0. URL: <https://doi.org/10.1007/s10951-012-0290-0>.

Paper D

Trading Plan Cost for Timeliness in Situated Temporal Planning

Shahaf S. Shperberg, Andrew Coles, Erez Karpas, Eyal Shimony,
Wheeler Ruml

The paper has been published in the
*IJCAI '20: Proceedings of the Twenty-Ninth International Joint Conference on Artificial
Intelligence, 2020*
pp. 4176–4182

© 2020 IJCAI

The layout has been revised.

Abstract

If a planning agent is considering taking a bus, for example, the time that passes during its planning can affect the feasibility of its plans, as the bus may depart before the agent has found a complete plan. Previous work on this situated temporal planning setting proposed an abstract deliberation scheduling scheme for maximizing the probability of finding a plan that is still feasible at the time it is found. In this paper, we extend the deliberation scheduling approach to address problems in which plans can differ in their cost. Like the planning deadlines, these costs can be uncertain until a complete plan has been found. We show that finding a deliberation policy that minimizes expected cost is PSPACE-hard and that even for known costs and deadlines the optimal solution is a contingent, rather than sequential, schedule. We then analyze special cases of the problem and use these results to propose a greedy scheme that considers both the uncertain deadlines and costs. Our empirical evaluation shows that the greedy scheme performs well in practice on a variety of problems, including some generated from planner search trees.

1 Introduction

Situated temporal planning (Cashmore et al. 2018) is a model for the planning problem faced by an agent for whom significant time passes as it plans. In this setting, external temporal constraints (e.g., deadlines) can be introduced depending on the actions included in a plan. For example, taking the 9:00 bus introduces a new constraint that the agent must be at the bus stop by 9:00. These plan-specific constraints make the problem different than real-time search (e.g., (Koenig and Sun 2009; Sharon et al. 2014; Cserna, Ruml, et al. 2017; Cserna, Doyle, et al. 2018)), deadline-aware search (Dionne et al. 2011), or Best-first Utility Guided Search (Burns et al. 2013).

Situated temporal planning calls for a search strategy different from traditional offline search algorithms, as the choice of which node to expand must account for the fact that time spent exploring one part of the search space passes in the real world and may invalidate other partial plans. Shperberg et al. 2019 suggested a rational meta-reasoning (Russell and Wefald 1991) scheme for situated temporal planning. They formalized the problem as an MDP (called AE2 for Allocating Effort when Actions

Expire) whose actions allocate a unit of time to one of n running processes, showed that solving this MDP optimally is NP-hard, and suggested a greedy decision rule (denoted P-Greedy henceforth) that worked well in their empirical evaluation. However, P-Greedy attempts merely to maximize the chance of finding a timely plan, without considering plan cost. For example, if taking a taxi does not introduce a deadline but is much more expensive than taking the bus, P-Greedy always chooses to take the taxi, even if there is very little uncertainty about whether the agent could catch the bus on time.

In this paper, we extend the metareasoning problem for situated temporal planning to include plan cost. First, we provide a formal characterization of the metareasoning problem with plan costs as an MDP (which we call ACE2 for Allocating Computational Effort when Actions with Costs Expire), and show that optimally solving this MDP is PSPACE-complete. We also show that even when the deadlines and costs are known with certainty, an optimal solution for this problem requires a contingent policy, in contrast with AE2 with known deadlines, where previous work showed that an optimal policy can be linear. Finally, we provide an analytical solution to the special case where only one process may be scheduled, and use this solution to construct a greedy decision rule for the general case. Our empirical evaluation suggests that the new greedy scheme performs significantly better than various baseline algorithms and the P-Greedy scheme on benchmarks featuring several families of distributions, including distributions taken from actual runs of the OPTIC planner (Benton et al. 2012). This work brings situated temporal planning closer to practical applicability, as plan cost is often an important factor in applications.

2 Problem Statement: Cost vs Timeliness

The desire to achieve one’s goals as cheaply as possible, as well as in a timely manner, induces a tradeoff between plan cost and the probability of successfully finding a plan that is still executable at the time it is found (except in the rare case where all potential plans have equal cost). In order to make the metareasoning problem well defined, we introduce a cost of failure c_f and require minimization of the expected cost over outcomes representing either costs of timely plans or the cost of failure.

2. Problem Statement: Cost vs Timeliness

When the value of c_f is high, optimal policies will aim to maximize the probability of finding a plan, while for low values of c_f , optimal policies will focus on finding a cheap plan, at the risk of not finding a plan at all. Moreover, the simpler problem of maximizing the probability of success is a special case where the cost of all correct and timely plans is zero and $c_f > 0$.

Following Shperberg et al. 2019, we abstract away from any specific planning methodology and merely posit the existence of n computational processes, all attempting to solve the same problem, and a single computing core for running all of the processes. When a process completes, it delivers a solution plan with a known execution cost. However, every solution also has a (possibly unknown) deadline, and if the solution is delivered at a time that is later than that deadline, it cannot be used. A solution delivered at or before its deadline is called *timely*.

We can now define the ACE2 problem, in which the objective is to minimize the expected cost of the timely solution found by the set of processes. The following distributions are assumed to be known: **(i)** $D_i(t)$, the cumulative distribution function (CDF) over wall clock times of a random variable denoting the deadline for each process i . **(ii)** $M_i(t)$, the CDF giving the probability that process i will terminate when given an accumulated computation time of t or less. **(iii)** C_i , a probability mass function (PMF) over solution costs for process i . We denote by d_{\max_i} the time of the latest possible deadline for process i , i.e. the smallest t for which $D_i(t) = 1$.

The true values of the deadline for process i and the cost of plan i are revealed only when process i completes its computation. We call such a process *completed*, otherwise it is *incomplete*. The cost of a plan of an incomplete process, as well as a completed process that has failed to find a timely solution, is assumed to be c_f . Thus, the probability of a process failing to find a plan at all can be incorporated into its cost distribution.

The problem is to find a policy for allocating the computing core's time among the processes, as well as optionally stopping deliberation and executing a complete plan already computed by one of the processes, so as to minimize expected cost of the executed plan. Note that in AE2 there is no need to include an explicit decision to start executing a plan, as once the first feasible plan is found, there is no benefit in searching for better plans. However, when cost is considered, even after the first

timely plan is found, we may want to delay executing it in the hope of finding a better plan.

When modeling processes solving the same problem, the distributions over costs and deadlines may have dependencies, complicating both the distribution estimation in practice and the deliberation scheduling. In fact we show below that optimally solving ACE2 with dependencies is PSPACE-hard. Therefore, we make the metareasoning assumption that all these distributions are independent, which in certain simple special cases allows for efficient solutions. Since these simple cases do not cover our desired scenarios, we also present a greedy algorithm for effectively handling the problem, based on intuitions from the special cases.

3 Deliberation Scheduling MDP with Costs

Following Shperberg et al. 2019, we formulate a discrete-time version of the problem, DACE2, allowing us to model the problem as an MDP. We define $m_i(t) = M_i(t) - M_i(t-1)$, the probability that process i completes after exactly t steps of computation, and $d_i(t) = D_i(t) - D_i(t-1)$, the probability that the deadline for process i is exactly at time t .

We formalize the DACE2 MDP as an indefinite duration MDP with terminal states, where we keep track of time as part of the state. The state variables are the wall clock time T , and one state variable T_i for each process, with domain \mathbb{N} , which represents the cumulative time assigned to process i so far. In addition, we have state variables ct_i and dl_i , the cost and deadline (respectively) of each process that has completed its computation. We also have a special terminal state DONE. Since the ct_i and dl_i variables are irrelevant for incomplete processes, and the time assigned to a process is irrelevant to completed processes, the state space can be stated as:

$$\mathcal{S} = \{\text{DONE}\} \cup \left(\text{dom}(T) \times \bigtimes_{1 \leq i \leq n} (\text{dom}(T_i) \cup (\text{dom}(ct_i) \times \text{dom}(dl_i))) \right)$$

There are $2n$ actions in the MDP, $\{a_1, \dots, a_n\} \cup \{g_1, \dots, g_n\}$. Action a_i assigns the next unit of computation time to incomplete process i . Action g_i denotes giving the plan computed by completed process i the go-ahead to execute, and transitioning into

3. Deliberation Scheduling MDP with Costs

a terminal state. Note that for every process i , either a_i or g_i is applicable but not both.

The initial state S_0 has $T = 0$ and $T_i = 0$ for all $1 \leq i \leq n$. We use the notation $T[S_0] = 0$ and $T_i[S_0] = 0$ (i.e. state variable as a function of the state) as a shorthand to denote this. The transition distribution of the action a_i is determined by the M_i and D_i distributions. If a process has just completed its computation in the transition from state S to state S' , then $ct_i[S']$ and $dl_i[S']$ are assigned according to the actual deadline and cost of the solution obtained by process i . The state variables for other processes remain unchanged.

More precisely, when transitioning from state S to S' by applying action a_i : **(1)** The current time $T[S'] = T[S] + 1$. **(2)** The computation time of every other process remains unchanged, that is $\forall j \neq i : T_j[S'] = T_j[S]$. **(3)** The probability that process i 's computation completes in this transition is $P_{term} = \frac{m_i(T_i[S]+1)}{1-M_i(T_i[S])}$. Therefore, with probability $1 - P_{term}$, process i does not complete and we have $T_i[S'] = T_i[S] + 1$. Conversely, with probability P_{term} , process i completes. In this case, $ct_i[S']$ and $dl_i[S']$ are assigned values according to distributions C_i and D_i , respectively. For example, in the independent cost case, for all $x \leq d_{max_i}$, $dl_i[S'] = x$ with probability $d_i(x)$; if $x < T[S']$, then $ct_i[S'] = c_f$, otherwise, for all y , $ct_i[S'] = y$ with probability $C_i(y)$.

The reward for executing action a_i before the last deadline of process i has passed ($T[S] < d_{max_i}$) is always 0, but when a_i is applied in state S with $T[S] \geq d_{max_i}$, the reward is $-c_f$. In the latter case, transition into $S' = \text{DONE}$ with probability 1. This exception is in order to avoid useless allocation of time to processes where certainly no timely plan can be found, as well as infinite allocation action sequences.

When applying action g_i in state S , that is, executing the plan found by completed process i , the transition is always to terminal state $S' = \text{DONE}$. The reward in this case is $-ct_i[S]$ if $dl_i \geq T[S]$ and $-c_f$ otherwise.

Although the MDP is a formal statement of the DACE2 problem, we assume that it is specified implicitly through a compact representation of the distribution (including any possible dependencies), as the state space of the MDP is exponential in the number of processes.

4 Theoretical Analysis of ACE2

4.1 Complexity of ACE2

Optimally solving the ACE2 problem is intractable. In fact, for the general case with unrestricted dependencies we have:

Theorem D.1

Finding the first action in an optimal policy for a DACE2 problem is PSPACE-hard.

Proof. By reduction from quantified Boolean formulas (QBF). Let α be a QBF specified in prenex normal form, i.e.:

$$\alpha = \exists x_1 \forall y_1 \dots \exists x_n \forall y_n \Phi(x_1, y_1, \dots, x_n, y_n) \quad (\text{D.1})$$

with Φ being a 3-CNF Boolean formula in the x_i, y_i variables. Determining the truth of α , or equivalently the existence of a policy for assigning the x_i such that Φ is always satisfied, is PSPACE-hard (Garey and Johnson 1979, problem LO11).

We transform an instance of QBF into an instance of DACE2 with $2n + 2$ processes, whose optimal allocation policy reveals the truth status of α . Among these processes, there are $2n$ ‘variable setting’ processes $p_{x_i}^j$, one for each of the n existential variables and for each $j \in \{t, f\}$. Each $p_{x_i}^j$ always takes exactly 1 time unit. Allocating time to $p_{x_i}^j$ represents assigning the value j to x_i . The uncontrollable y_i variables are modeled using the stochastic cost outcomes of the $p_{x_i}^j$ processes. Let $c_f = 2^{n+2}$. Each $p_{x_i}^j$ has three possible costs: 1) $\frac{c_f}{2}$, representing an outcome where the universally quantified variable y_i that follows x_i is false, 2) $\frac{c_f}{2} + 1$ representing an outcome in which y_i is true, and 3) c_f , representing a "failure" or that the process is incomplete (because it was not allocated any computation time yet). As a shorthand, we denote the event $C(p_{x_i}^j) = \frac{c_f}{2}$ by $p_{x_i,0}^j$, the event $C(p_{x_i}^j) = \frac{c_f}{2} + 1$ by $p_{x_i,1}^j$, and $C(p_{x_i}^j) = c_f$ by $p_{x_i,I}^j$.

The remaining two processes, denoted by p_{default} and p_{success} are designed to be selected according to whether α is true. p_{default} always takes time $2n + 1$ to complete and delivers a solution with a known cost of 1. We show below that the optimal policy schedules the a action followed by the g action for p_{default} just when α is false.

When α is true, we show that it is optimal to schedule the $p_{x_i}^j$ corresponding to a satisfying assignment of α followed by p_{success} , which always takes time $n + 1$, has a

4. Theoretical Analysis of ACE2

cost distribution depending deterministically on the $p_{x_i}^j$, with cost 0 or c_f , depending on other costs, as defined below. If $p_{success}$ is run and delivers a solution of cost 0, the optimal policy then executes that solution (i.e. do the g action for $p_{success}$). The deadline for all processes is $D = 2n + 1$, so by construction one can either run $p_{default}$, or run n variable-setting processes and then $p_{success}$ before the deadline.

To test the truth of α , we have two tasks: enforcing ordered setting of the x_i , and letting the cost of $p_{success}$ be 0 if α is true. We now define the details of the probability distribution over the costs for the $p_{x_i}^j$ and $p_{success}$. First, we essentially force one of each $p_{x_i}^j$ pair to be allocated in order to make $p_{success}$ achieve cost 0. For $j \in \{f, t\}$ and $n \geq i \geq 2$,

$$P(p_{x_i, I}^j | p_{x_{i-1}, I}^f \wedge p_{x_{i-1}, I}^t) = 1$$

Note the dependence of the cost distribution on the observed costs of the preceding $p_{x_i}^j$ processes. The cost distribution for completed processes (for conciseness, we omit this conditioning event in the equations) is $P(p_{x_1, 0}^j) = P(p_{x_1, 1}^j) = 0.5$ and for $n \geq i \geq 2$, we have:

$$\begin{aligned} P(p_{x_i, 0}^j | \neg(p_{x_{i-1}, I}^f \wedge p_{x_{i-1}, I}^t)) \\ = P(p_{x_i, 1}^j | \neg(p_{x_{i-1}, I}^f \wedge p_{x_{i-1}, I}^t)) = 0.5 \end{aligned}$$

The cost of $p_{success}$ given the costs of the $p_{x_i}^j$ is defined as follows. If for any $1 \leq i \leq n$ we have $p_{x_i, I}^f \wedge p_{x_i, I}^t$, then $C(p_{success}) = c_f$ with probability 1. In all other cases we have exactly one of $C(p_{x_i}^t) < c_f$ or $C(p_{x_i}^f) < c_f$ for all i . Thus, the costs of all the $p_{x_i}^j$ encode a unique assignment A to all the Boolean variables. Let $C(p_{success}) = 0$ if assignment A satisfies Φ , and $C(p_{success}) = c_f$ otherwise. So the optimal policy is to choose $p_{default}$ (for a cost of 1) just when α is false. That is because in this case there is a probability of at least 2^{-n} of getting a non-satisfying assignment, thus a cost of at least $\frac{c_f}{2}$, and an expected cost $\geq \frac{2^{n+2}}{2^{n+1}} = 2$ if the default is not selected. However, if α is true, there exists a policy that always satisfies Φ , and thus the optimal scheduling policy begins with $p_{x_0}^f$ or $p_{x_0}^t$ (and concludes with $p_{success}$), delivering a cost of 0 with probability 1. \square

If the last possible deadline is polynomial in the problem description size, then finding the optimal first action is in PSPACE, since this is a ‘game against nature’

of polynomial length. Thus, under this restriction, the DACE2 problem is PSPACE-complete. Note that since the DACE2 problem is PSPACE-hard, it follows immediately that ACE2 is also PSPACE-hard, as the discrete problem is a special case of the more general model of ACE2.

Although of theoretical significance, membership in PSPACE is no help for practical solution of this scheduling problem, especially as it was conceived as a metareasoning problem, which must be solved in negligible computation time. Henceforth, we make the metareasoning assumption that all the random variables are jointly independent.

4.2 The Case of Known Costs

The special case of DACE2 where all plan costs are 0 and the cost of failure is 1 is equivalent to the discrete AE2 model of Shperberg et al. 2019 (denoted by SAE2). Further simplifying the SAE2 model to known deadlines, the problem was shown to be NP-hard, even though the optimal schedule is a linear sequence. However, with different costs the optimal schedule is not even necessarily linear.

Theorem D.2

Optimal solutions for DACE2 must sometimes consist of a contingent schedule (that depends on previous results delivered by processes), even for the restricted case of known deadlines and known costs.

Proof. Let $c_f = 100$ and consider processes $\{1, 2, 3\}$, with $C_1 = 0, C_2 = 10, C_3 = 15$, all with the same deadline $d_i = 2$. Let the completion-time distributions be: $m_1 = [0.1 : 1, 0.9 : 10], m_2 = [0.5 : 1, 0.5 : 10], m_3 = [1 : 1]$. That is, all processes have some probability of completing computation in one time unit, and otherwise do not complete within the deadline, with process 3 surely completing and delivering a plan costing 15. Thus, by construction, any optimal policy will run a process for at most one time unit, and only 2 processes can be scheduled to run before the (common) deadline.

The optimal policy P^* here is to run process 2 first (action a_2). If process 2 completes (thus delivering a cost 10 plan), then run process 1 (action a_1), resulting in an expected cost of $(10 \cdot 0.9 + 0 \cdot 0.1 = 9)$ in this branch, because if process 1 completes we have a plan with cost 0 (do action g_1), otherwise we have a plan with

cost 10 (do action g_2). If process 2 does not complete, the best option is a_3 to get a plan with cost 15 (and then do g_3 to execute it). Expected cost of this policy is thus $E[C(P^*)] = 0.5 \cdot 9 + 0.5 \cdot 15 = 12$. Note that this is a strictly conditional (i.e. non-linear), policy.

To see that P^* is strictly better than any other policy, consider starting with a_3 , which delivers a plan with cost 15. Now doing a_1 , we get a 0.1 probability of improving this to 0, so the expected cost is 13.5, while doing a_2 at this point we get a probability of 0.5 of improving the cost to 10, and thus a cost 12.5 in expectation, in both cases worse than P^* . Alternately, starting with a_1 , if process 1 completes we get cost 0, but otherwise the best we can do is a_3 to get a plan with cost 15. This is also suboptimal (expected cost of this policy is 13.5). \square

4.3 Simple Special Case: One Running Process

Looking for tractable cases, we consider the case where we are allowed to allocate time to only **one** incomplete process, denoted by i . However, we also posit $m - 1$ **completed plans** with known costs c_j and known deadlines d_j for $1 \leq j \leq m - 1$. These plans are results from processes that have completed computation in the past, if any. For convenience, we add an additional completed dummy plan m with $c_m = c_f$ and $d_m = \infty$, denoting the failure case. Without loss of generality, let $0 = d_1 < d_2 < \dots < d_m = \infty$ and also $c_1 < c_2 < \dots < c_m = c_f$, as a plan c_j that has a cost greater than or equal to c_{j+1} is dominated and can be removed. We can find an optimal stopping policy for the incomplete process i by considering all possible stopping points d_1, \dots, d_m .

Theorem D.3

If only one specific incomplete process can be scheduled, the optimal schedule (consisting of the optimal stopping time) can be computed in polynomial time.

Proof. Let policy $\pi_{i,k}(t_d)$ be to do no computation for delay time t_d (needed later on), then run process i until time d_k (if $d_k > t_d$). Finally, execute the cheapest timely plan available when i completes or "times out". Computing the expected cost of $\pi_{i,1}(t_d)$, denoted by $E_{\pi_{i,1}}(t_d)$, is easy because $d_1 = 0$, so no computation time is allowed for process i . The best timely plan at time t_d has cost c_l for the first l for which $d_l \geq t_d$, and thus $E_{\pi_{i,1}}(t_d) = c_l$. In particular, $E_{\pi_{i,1}}(0) = c_1$.

The value of $\pi_{i,2}(t_d)$ is more complex. Let us denote by $s_i(t, t_d)$ the probability that process i , starting at time t_d and running with no interruption, will complete its computation in at most t additional time units, and succeed in finding a timely solution. Likewise, denote by $f_i(t, t_d)$ the probability that process i will complete computing under the same conditions, and fail to find a solution (or find a solution, but discover that the deadline has passed). That is, we have:

$$s_i(t, t_d) = \sum_{t'=1}^t m_i(t')(1 - D_i(t_d + t' - 1))$$

$$f_i(t, t_d) = \sum_{t'=1}^t m_i(t')D_i(t_d + t' - 1)$$

Note that $f_i(t, t_d) + s_i(t, t_d) = M_i(t)$ because at any point in time, for a completed process, success and failure (in finding a timely solution) are mutually exclusive events.

If we decide to stop computing at d_2 , we get cost c_2 unless something better came up earlier. That is, we get cost c_2 either if process i is incomplete, or if it is complete but failed; however, if process i finds a timely solution, we get the better among the solution cost returned by process i and c_2 . Thus:

$$\begin{aligned} E\pi_{i,2}(t_d) &= (1 - M_i(d'_2)) \cdot c_2 + (f_i(d'_2, t_d) - f_i(d'_1, t_d)) \\ &\quad \cdot c_2 + (s_i(d'_2, t_d) - s_i(d'_1, t_d)) \cdot E[\min(C_i, c_2)] \end{aligned}$$

where $d'_j = d_j - t_d$. Note that we may actually get a result with expected cost less than c_1 even if $E[C_i] > c_1$.

Generalizing to compute the expected cost of $\pi_k(t_d)$: with probability $1 - M_i(t - t_d)$ process i does not complete before t , so we pick plan k with cost c_k . The probability that the computation stops with failure between d_{j-1} and d_j is: $f_i(d'_j, t_d) - f_i(d'_{j-1}, t_d)$, in which case we pick plan j with cost c_j . With probability $s_i(d'_j, t_d) - s_i(d'_{j-1}, t_d)$ the computation generates a timely plan after d_{j-1} but before d_j , resulting in expected cost $E[\min(C_i, c_j)]$. All in all, we get expected cost:

$$\begin{aligned} E\pi_{i,k}(t_d) &= (1 - M_i(d'_k)) \cdot c_k + \\ &\quad \sum_{j=2}^k ((f_i(d'_j, t_d) - f_i(d'_{j-1}, t_d)) \cdot c_j + \\ &\quad (s_i(d'_j, t_d) - s_i(d'_{j-1}, t_d)) \cdot E[\min(C_i, c_j)]) \end{aligned}$$

Thus in the case of one incomplete process to be scheduled, one can simply compute the values of all the $E_{\pi_{i,k}}(0)$, and select the optimal stopping point. \square

Unfortunately, this result cannot be easily extended to schedule more than one incomplete process. One can still solve the MDP in time that is ‘only’ exponential in the number of incomplete processes that may be scheduled. Obviously this is not reasonable for a large number of such processes, and certainly not for the case where processes actually stand for search nodes and must be scheduled in negligible time. Instead, we use the above ideas to generate a greedy rule that performs well in practice.

5 A Greedy Scheme With Costs

When considering only one incomplete process i for scheduling, there was no benefit in allocating runtime other than d_k (for some k), as we are not allowed to use time not allocated to process i towards other processes. However, when multiple processes may be scheduled, it is desirable to use as little time as necessary for the current process, as any remaining time can be used by other incomplete processes. Therefore, we generalize the notion of the value of a stopping policy π_k to stopping at d_k *subject to the constraint* that at most t time is allocated to the current process (after a delay of t_d). The expected value of the constrained policy, denoted by $\pi_k(t, t_d)$, is given by:

$$\begin{aligned} E_{\pi_{i,k}}(t, t_d) = & (1 - M_i(\min(t, d'_k))) \cdot c_k + \\ & \sum_{j=2}^k ((f_i(\min(t, d'_j), t_d) - f_i(\min(t, d'_{j-1}), t_d)) \cdot c_j + \\ & (s_i(\min(t, d'_j), t_d) - s_i(\min(t, d'_{j-1}), t_d))) \cdot \\ & E[\min(C_i, c_j)] \end{aligned}$$

Although $E_{\pi_{i,k}}(t, t_d)$ is monotonically decreasing in t , there is some t at which the expected returns (minus expected costs) per time unit is maximized, and this type of quantity is the useful basis of numerous greedy algorithms. Using this idea, we define the *most-effective reward gain (i.e. cost reduction) rate* for process i , relative to the current best valid plan cost c_c as:

$$ecr_i(t_d) = \max_{t,k} \frac{c_c - E_{\pi_{i,k}}(t, t_d)}{t} \quad (\text{D.2})$$

with $c_c = c_f$ if there is no currently valid plan. This definition resembles the idea behind the returns rate at the ‘most effective computation time’ (e_i) of Shperberg et al. 2019, but now in terms of expected cost, rather than logarithm of probability of failure.

It is common to use the value $ecr_i(0)$ (highest rate of returns for process i) to select the process i that has the highest returns rate among all processes. However, some processes are more time-critical than others, and this can be measured by how much the returns rate drops due to delaying the start of the processing for process i by t_d . The returns rate for a delayed process i is given by $ecr_i(t_d)$. Processes for which this decrease in returns rate $ecr_i(0) - ecr_i(t_d)$ is high should get priority. Trading off high returns rate with loss in returns rate due to delay, we get the following criterion:

$$Q_i(t_d) = ecr_i(0) - \gamma ecr_i(t_d) \quad (\text{D.3})$$

Note that γ is an empirically determined constant that can be used to balance between immediate reward (reward from allocating time to process i now) and future loss (due to delaying time allocation to process i by t_d). The value $\gamma = 0.5$ gives these factors equal weight, so one might expect a value not far from that to provide a good balance.

We defined a greedy scheme called Delay-Aware Greedy (DAG) based on Equation D.3. This scheme allocates time to a process i that maximizes $Q_i(t_d)$, we used $t_d = 1$ in the experiments. If for each i , the optimal individual policy for process i is to stop processing, then the algorithm terminates and executes the first valid plan.

Each iteration of DAG requires computing the ecr_i values of each process. Therefore, a naïve implementation would compute $E_{\pi_{i,k}}(t, t_d)$ for every completed plan, for every process, and for every $1 \leq t \leq d_{\max_i}$. Computing each $E_{\pi_{i,k}}(t, t_d)$ takes time $O(m \cdot d_{\max_i})$. Thus, a single iteration of DAG requires $O(n \cdot m^2 \cdot d_{\max}^2)$ time, where $d_{\max} = \max_{1 \leq i \leq n} d_{\max_i}$. Since there are at most d_{\max} iterations, and m is bounded by the number of processes, the execution of DAG takes $O(n^3 \cdot d_{\max}^3)$. However, DAG can be optimized. First, the values of s_i , f_i , $E_{\pi_{i,k}}$ and ecr_i can be pre-computed for every $1 \leq t, t_d \leq d_{\max}$ in $O(d_{\max}^2)$ time, and can be used across all iterations. Afterwards, each iteration would only take $O(n)$ time in order to obtain the process with the highest Q_i value. Hence, the overall runtime of DAG can be bounded by $O(n \cdot d_{\max}^2)$. Finally, if the M_i and D_i distributions are given implicitly, the computations of ecr_i ,

6. Empirical Results

Dist	# pr	MDP		MPP		RR		Random		P-Greedy		DAG(1/2)		DAG(1)	
		C	T	C	T	C	T	C	T	C	T	C	T	C	T
B	2	23.43	90,860	69.22	0	80.46	0	108.74	0	30.93	0	26.13	0	25.88	0
	5	19.23	7.9×10^7	40.78	0	63.73	0	75.78	0	29.07	0	26.61	0	20.91	0
	10			31.46	0	50.98	0	65.18	0	19.94	0	20.93	0	16.39	0
	100			26.15	0	37.89	0	43.57	0	14.81	10	12.21	80	12.07	80
N	2	41.16	107,730	104.82	0	166.98	0	205.69	0	68.17	0	63.67	0	53.19	0
	5	40.97	1.2×10^8	99.72	0	135	0	171.95	0	72.38	0	59.92	0	47.37	0
	10			81.1	0	130.63	0	151.47	0	62.87	0	38.47	0	40.89	0
	100			74.6	0	119.93	0	153.97	0	45.18	10	41.98	80	38.61	70
U	2	30.54	112,030	86.72	0	111.61	0	147.85	0	55.53	0	40.11	0	39.52	0
	5	35.18	1.3×10^8	84.21	0	120.48	0	152.39	0	50.93	0	37.55	0	38.26	0
	10			79.9	0	104.67	0	138.63	0	47.42	0	36.89	0	35.19	0
	100			68.19	0	93.04	0	121.39	0	44.14	10	33.65	90	30.74	80
P	2	193.55	328,960	456.32	0	628.26	0	750.53	0	299.39	0	210.44	0	200.01	0
	5			386.35	0	545.11	0	690.26	0	275.42	0	192.47	0	181.49	0
	10			369.35	0	462.62	0	606.33	0	228.59	10	169.87	80	160.78	80
	100			254.43	10	380.15	0	475.6	0	171.45	50	127.01	460	126.22	420
Average				144.58	0	201.97	0	253.71	0	94.76	10	71.12	50	66.72	50

Table D.1: Solution cost and metareasoning runtime (ms) of the algorithms on different types of benchmark problems.

s_i and f_i do not need to consider every $1 \leq t \leq d_{max}$, but rather specific points of interest. For example, if the distributions are piecewise linear, one may only need to examine transition points between segments and some other segment intersections.

6 Empirical Results

We tested the new DAG method on DACE2 problems whose performance profiles, cost distributions, and deadline distributions had a variety of forms. Following Shperberg et al. 2019, we used: Uniform (U), with minimal range value $a = 1$ and maximal range value b uniformly drawn from $\{[5, 10], [50, 100], [100, 200], [150, 300]\}$, we denote the set of possible $[a, b]$ ranges by R ; Boltzmann (B), truncated exponential distribution with the diminishing return property, using a $\lambda \in \{0.1, 1, 2\}$ and range drawn from R ; Truncated Normal Distribution (N) with $\mu \in \{5, 50, 100, 150\}$, $\sigma \in \{1, 5, 10\}$, and range drawn from R ; and Planner (P), which are distributions collected from search trees of the OPTIC planner when run on problems from the Robocup Logistics League (Niemueller et al. 2015) domain. To acquire the planner distributions, A* was executed from each node of a dumped search tree. The result of each of these searches provides the number $N(v)$ of expansions necessary to find the goal under a node v .

These numbers were binned separately for each $(h(v), g(v))$ pair. Then, a set V was formed by selecting nodes randomly from the tree, each one standing for a process. For each such $v \in V$, the list of numbers of expansion in the bin corresponding to $g(v)$ and $h(v)$ was treated as a distribution over completion times (in terms of number of expansions). Likewise for creating the latest start times for the resulting plan (the deadline distribution). When using our method as part of a planner, one would need to create such statistics on-the-fly.

We used the following scheduling algorithms as baselines for the evaluation: **(i) MDP:** solution computed using the Bellman equation (whenever computationally feasible). **(ii) P-Greedy:** the greedy log-probability of failure minimization scheme of Shperberg et al. 2019. **(iii) Random:** allocate time to a random process that has not already failed. **(iv) Most promising process (MPP):** allocate time to the process with the highest probability to find a timely solution. In case of failure, the algorithm chooses the next most promising process. **(v) Round-robin (RR):** allocate time units to each non-failed process in equal portions and in circular order. In addition to a scheduling scheme, we must specify a stopping scheme used by the candidate algorithms. The last three algorithms choose the best (least-cost) timely completed plan i available at any given time, but continue scheduling processes until the (now known) deadline for i arrives, in case a better plan is found. Once the deadline for i arrives, this plan is executed. We compared all the above algorithms to the DAG method using $\gamma \in \{0, 0.2, 0.5, 0.75, 1, 2, 10, 100\}$. However, the reported results include only 0.5 and 1, which were the best parameter values.

Evaluating the quality of a solution (policy) is not a trivial task, especially for adaptive policies that depend on the state to make a decision. In order to tackle this issue, we ran the algorithms on each setting for 500 attempts and reported the average cost over all runs. Since this policy evaluation process introduced noise, we have measured the standard deviation (std) of the solution quality (not reported in the table); the overall std was small (± 3.27), therefore, the introduced noise does not affect the trends reported below. The results are shown in Table E.1. The C column indicates the average cost achieved by the policy created by the algorithms and the T column is the metareasoning runtime in milliseconds. The last rows gives average solution cost and geometric mean of the runtimes. Generally, the DAG scheme achieved the lowest

costs among all algorithms (except for the MDP, which is optimal), and demonstrates a significant advantage over the naive schemes in terms of solution cost; $\gamma = 1$ seems to be the best balance between loss due to delay and reward slope. Although DAG was the slowest among all non-MDP algorithms, it is still several orders of magnitude faster than the MDP and required less than 1 second in all cases.

7 Conclusion

Situated temporal planning can benefit from metareasoning about the unknown deadlines and search process runtimes. An abstract deliberation scheduling scheme modeling such search processes, aimed at maximizing the *probability* of finding a timely solution, was developed by Shperberg et al. 2019, but it did not model the cost of the computed solution. This paper extends the latter scheme to handle plan costs.

An MDP model of the extended deliberation scheduling problem was defined and its complexity was analyzed. We showed that the incorporation of costs significantly complicates the metareasoning problem in several ways. First, even when everything except algorithm runtimes is known (deadlines, costs), the optimal schedule requires contingent policies, rather than just a linear schedule as in the case without costs. Second, the introduction of costs now necessitates a stopping policy, trading off execution of an already computed solution vs. attempting to find better solutions, at the risk of making the current solution(s) expire. Finally, with costs (and dependencies) we proved that finding even the first action in an optimal schedule is PSPACE-hard.

As the MDP has exponential size and is provably intractable, we examined special cases and approximations. We gave a polynomial-time optimal solution for a simple case in which there is only one running process (and a set of completed processes). We present a greedy algorithm (DAG) based on intuitions from this simple case, as well as the greedy algorithm described by Shperberg et al. 2019 (P-Greedy). Finally, we compared the new algorithm empirically against P-Greedy and other base-line algorithms; the results of DAG outperform the other methods. For very small instances, where the MDP solution was possible, the DAG scheme was near-optimal.

As more and more agents make plans that interact with the external world in all its temporal complexity, this work will help provide a foundation allowing situated

planning to see use in applications where costs play a significant role.

Acknowledgements

Partially supported by ISF grant #844/17, and by the Frankel center for CS at BGU. Project also funded by the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 821988 (ADE).

References of Paper D

- [BCC12] J. Benton, Amanda Jane Coles, and Andrew Coles. “Temporal Planning with Preferences and Time-Dependent Continuous Costs”. In: *ICAPS*. Atibaia, São Paulo, Brazil: AAAI Press, 2012, pp. 2–10.
- [BRD13] Ethan Burns, Wheeler Ruml, and Minh Binh Do. “Heuristic Search When Time Matters”. In: *J. Artif. Intell. Res.* 47 (2013), pp. 697–740. DOI: 10.1613/jair.4047. URL: <https://doi.org/10.1613/jair.4047>.
- [Cas+18] Michael Cashmore et al. “Temporal Planning while the Clock Ticks”. In: *ICAPS*. AAAI Press, 2018, pp. 39–46. URL: <https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17724>.
- [CRF17] Bence Cserna, Wheeler Ruml, and Jeremy Frank. “Planning Time to Think: Metareasoning for On-Line Planning with Durative Actions”. In: *ICAPS*. 2017, pp. 56–60. URL: <https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15753>.
- [Cse+18] Bence Cserna, William J. Doyle, et al. “Avoiding Dead Ends in Real-Time Heuristic Search”. In: *AAAI*. 2018, pp. 1306–1313. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17405>.
- [DTR11] Austin J. Dionne, Jordan Tyler Thayer, and Wheeler Ruml. “Deadline-Aware Search Using On-Line Measures of Behavior”. In: *Proceedings of the Symposium on Combinatorial Search (SoCS)*. 2011, pp. 39–46.
- [GJ79] M. R. Garey and D. S. Johnson. “Computers and Intractability, A Guide to the Theory of NP-completeness”. In: W. H. Freeman and Co., 1979, p. 190.
- [KS09] Sven Koenig and Xiaoxun Sun. “Comparing real-time and incremental heuristic search for real-time situated agents”. In: *Auton. Agents Multi Agent Syst.* 18.3 (June 2009), pp. 313–341. ISSN: 1573-7454. DOI: 10.1007/s10458-008-9061-x. URL: <https://doi.org/10.1007/s10458-008-9061-x>.

- [NLF15] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. “The RoboCup Logistics League as a Benchmark for Planning in Robotics”. In: *WS on Planning and Robotics (PlanRob) at Int. Conf. on Aut. Planning and Scheduling (ICAPS)*. 2015.
- [RW91] Stuart J. Russell and Eric Wefald. “Principles of Metareasoning”. In: *Artif. Intell.* 49.1-3 (1991), pp. 361–395. DOI: 10.1016/0004-3702(91)90015-C. URL: [http://dx.doi.org/10.1016/0004-3702\(91\)90015-C](http://dx.doi.org/10.1016/0004-3702(91)90015-C).
- [SFS14] Guni Sharon, Ariel Felner, and Nathan R. Sturtevant. “Exponential Deepening A* for Real-Time Agent-Centered Search”. In: *AAAI*. AAAI Press, 2014, pp. 871–877. URL: <http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8216>.
- [Shp+19] Shahaf S. Shperberg et al. “Allocating Planning Effort When Actions Expire”. In: *AAAI*. AAAI Press, 2019, pp. 2371–2378.

Paper E

Situated Temporal Planning Using Deadline-aware Metareasoning

Shahaf S. Shperberg, Andrew Coles, Erez Karpas, Wheeler Ruml,
Solomon E. Shimony

The paper has been published in the
*ICAPS '21: Proceedings of the The Thirty-First International Conference on Automated
Planning and Scheduling, 2021*

© 2021 ICAPS

The layout has been revised.

Abstract

We address the problem of situated temporal planning, in which an agent’s plan can depend on scheduled exogenous events, and thus it becomes important to take the passage of time into account during the planning process. Previous work on situated temporal planning has proposed simple pruning strategies, as well as complex schemes for a simplified version of the associated metareasoning problem. Although even the simplified version of the metareasoning problem is NP-hard, we provide a pseudo-polynomial time optimal solution to the case with known deadlines. We leverage intuitions emerging from this case to provide a fast greedy scheme that significantly improves upon previous schemes even for the case of unknown deadlines. Finally, we show how this new method can be applied inside a practical situated temporal planner. An empirical evaluation suggests that the new planner provides state-of-the-art results on problems where external deadlines play a significant role.

1 Introduction

This paper addresses the problem of situated temporal planning, where an agent plans online in the presence of external temporal constraints such as deadlines. For example, if a promising partial plan involves taking a particular train, then it might be worth ensuring that the planning process finishes soon enough that the agent can get to the station in time. In other words, a plan must be found quickly enough that it is possible to execute that plan after planning completes. In this setting, search decisions and temporal constraints interact in complex ways, as choosing to include some action in a plan can introduce a temporal constraint for the subtree of the search tree that includes that action; time spent searching within other subtrees affects the applicability of that action. This differs from other time-aware planning settings, such as real-time heuristic search (Korf 1990), in that each open search node might have a different deadline.

The first planner to address situated temporal planning (Cashmore et al. 2018) uses temporal reasoning (Dechter et al. 1991) to prune search nodes for which it is provably too late to start execution. It also uses estimates of remaining search time (Dionne et al. 2011) together with information from a temporal relaxed planning graph (A. J.

Coles et al. 2010) to estimate whether a search node is likely to be *timely*, i.e. likely to lead to a solution that will be executable when planning finishes. As these estimates are not admissible, it uses dual open lists: one only for timely nodes, and another for all nodes (including nodes for which it is likely too late to start execution). However, the planner still uses standard heuristic search (Weighted A*) with these open lists, while noting that this is the wrong thing to do; leaving for future work finding the right search strategy.

Inspired by the situated planning setting, Shperberg, A. Coles, et al. 2019 proposed a rational metareasoning (Russell and Wefald 1991) approach for a simplified version of the search problem faced by a situated planner. The problem was simplified in several ways: first, only an abstract version of the metareasoning problem was addressed, and second, distributions over the remaining search time and deadlines were assumed known. The metareasoning problem was formulated as an MDP with the objective of maximizing the probability of finding a timely plan. This was proved to be NP-hard, even when the deadlines are known. However, the reduction was from the Knapsack problem, suggesting the possibility of a pseudo-polynomial time optimal solution algorithm. Shperberg, A. Coles, et al. 2019 also suggested a greedy, and somewhat ad-hoc, decision rule (denoted hereafter as basic greedy), which worked well in an empirical evaluation with various types of distributions.

In this paper, we first show that indeed the known deadline case can be solved in pseudo-polynomial time through dynamic programming (DP). Despite being optimal when deadlines are known, the DP approach does not perform well with unknown deadlines. Our second contribution is an alternate greedy decision rule, called DDA, that is better justified theoretically than basic greedy; we show empirically that DDA delivers better results than the basic greedy scheme in the same abstract setting of the problem.

Finally, our third contribution is to integrate the new metareasoning scheme as the search strategy for the situated temporal planner of Cashmore et al. 2018. An empirical evaluation shows that the new approach leads to timely solutions for significantly more problems than using standard heuristic search, even with pruning late nodes and dual open lists. This is an important step in bringing situated temporal planning closer to practical utility.

2 Background

We start by reviewing formal models of situated temporal planning and the associated metareasoning problem. Although heuristic search with external temporal constraints can arise in many settings, we focus in this paper on the problem of situated domain-independent temporal planning.

2.1 Problem Statement

Following Cashmore et al. 2018, we formulate situated temporal planning as propositional temporal planning with Timed Initial Literals (TIL) (Cresswell and Coddington 2003; Edelkamp and Hoffmann 2004). Such problems are specified by a tuple $\Pi = \langle F, A, I, T, G \rangle$, where:

- F , a set of Bool. propositions describing the world state.
- A is a set of durative actions; each action $a \in A$ has:
 - Duration in the range $[dur_{\min}(a), dur_{\max}(a)]$
 - Start condition $cond_{+}(a)$, invariant condition $cond_{\leftrightarrow}(a)$, and end condition $cond_{-}(a)$, all of which are subsets of F , and
 - Start effect $eff_{+}(a)$ and end effect $eff_{-}(a)$, both of which specify which propositions in F become true or false when a starts or ends, respectively.
- $I \subseteq F$ is the initial state, and $G \subseteq F$ specifies the goal.
- T is a set of timed initial literals (TIL). Each TIL $l = \langle time(l), lit(l) \rangle \in T$ consists of a time $time(l)$ and a literal $lit(l) \in F$, specifying a proposition that becomes true (or false) at $time(l)$.

A solution to a situated temporal planning problem is a schedule σ : a sequence of triples $\langle a, t_a, d_a \rangle$, where $a \in A$ is an action, $t_a \in \mathbb{R}^{0+}$ is the time when action a is started, and $d_a \in [dur_{\min}(a), dur_{\max}(a)]$ is the duration chosen for a . To define a valid schedule, we view it as a set of instantaneous *happenings* (Fox and Long 2003) that occur when an action starts, when an action ends, and when a timed initial literal is triggered. For each triple $\langle a, t, d \rangle$ in σ , we have action a starting at time t (requiring

$cond_{\vdash}(a)$ to hold a small amount of time ϵ before t , and applying the effects $eff_{\vdash}(a)$ right at t), and ending at $t + d$ (requiring $cond_{\vdash}(a)$ to hold ϵ before $t + d$, and applying the effects $eff_{\vdash}(a)$ at $t + d$). For TIL l we have the effect specified by $lit(l)$ triggered at $time(l)$. We require the invariant condition $cond_{\leftrightarrow}(a)$ to hold over the open interval between t and $t + d$, and the goal G to hold after all happenings have occurred.

The difference from standard temporal planning is that here we interpret the TILs as encoding temporal constraints in absolute time since planning started. Thus, we require the schedule σ to start only after planning is completed. That is, if the planner started at time 0 and took t_p time for its planning, we require that $t_a \geq t_p$ for all $\langle a, t_a, d_a \rangle \in \sigma$.

2.2 Metareasoning in Situated Planning

The requirement $t_a \geq t_p$ implies that the plan must be fully generated before the minimum t_a , which may be unknown until planning completes. For a partial plan available at a search node i in the planner, this can be modeled by a random variable d_i , denoting the unknown deadline by which a potential plan expanded from node i must be generated. Thus, the planner faces the metareasoning problem of deciding which nodes on the open list to expand in order to maximize the chance of finding a plan before its deadline.

Shperberg, A. Coles, et al. 2019 propose a model of this problem called (AE)² ('allocating effort when actions expire') which abstracts away from the planning problem and merely assumes n independent processes. Each process attempts to solve the same problem under time constraints. In the context of situated temporal planning using heuristic search (which we explore further below), each process may represent a promising partial plan for the goal, implemented as a node on the open list eager to have its subtree explored. But the abstract problem may also be applicable to other settings, such as algorithm portfolios or scheduling candidates for job interviews. For simplicity, we assume a single processor, so the core of the metareasoning problem is to determine how to schedule the n processes on the single processor.

When process i terminates, it delivers a solution with probability P_i or, otherwise, indicates its failure to find one. For each process, there is a deadline defined in absolute wall clock time by which its computation must be completed in order for

2. Background

any solution it finds to be valid. The deadline may be uncertain and is provided as a probability distribution. For process i , let $D_i(t)$ be the CDF over wall clock times of the random variable denoting the deadline. The actual deadline for a process is only discovered with certainty when the process completes. This models the fact that a dependence on an external timed event might not become clear until the final action in a plan is added. If a process terminates with a solution before its deadline, we say that it is *timely*. Given $D_i(t)$, we assume w.l.o.g. that P_i is 1, otherwise one can adjust $D_i(t)$ to make the probability of a deadline that is in the past (thus forcing the plan to fail) equal to $1 - P_i$.

The processes have known search time distributions, (performance profiles (Zilberstein and Russell 1996)) described by CDFs $M_i(t)$, the probability that process i needs total computation time t or less to terminate. Although some of the algorithms we present can handle dependencies, we make the typical metareasoning assumption in our analysis that all random variables are independent. Given the $D_i(t)$ and $M_i(t)$ distributions, the objective of (AE)² is to schedule processing time between the n processes maximizing the probability of at least one process finding a timely solution.

A simplified discrete-time version of the problem, called S(AE)², can be cast as a Markov decision process. The MDP's actions are to assign (schedule) the next time unit to process i , denoted by a_i with $i \in [1, n]$. Action a_i is allowed only if process i has not already failed. A process is considered to have failed if it has terminated and discovered that its deadline has already passed, or if the current time is later than the last possible deadline for the process.

The state variables are the wall clock time T and one state variable T_i for each process, with domain $\mathbb{N} \cup \{F\}$, although in practice the time domains of T, T_i are bounded by the latest possible deadlines. T_i denotes the cumulative time assigned to each process i until the current state, or that the process failed (indicated by F). We also have special terminal states SUCCESS and FAIL. Thus the state space is:

$$\mathcal{S} = (\text{dom}(T) \times \bigtimes_{1 \leq i \leq n} \text{dom}(T_i)) \cup \{\text{SUCCESS}, \text{FAIL}\}$$

The initial state has $T = 0$, and $T_i = 0$ for all $1 \leq i \leq n$. The transition distribution is determined by which process i has last been scheduled (the action a_i), the M_i distribution (which determines whether currently scheduled process i has completed its computation), and D_i (which determines the revealed deadline for a completed

process, and thus whether it has succeeded or failed). If all processes fail, transition into FAIL (with probability 1). If some process is successful, transition into SUCCESS. The reward is 0 for all states except SUCCESS, for which the reward is 1.

The size of the state-space of the $S(AE)^2$ MDP is exponential in the number of processes, so it is impractical to fully specify the MDP explicitly or to solve it directly. Furthermore, the $S(AE)^2$ problem is NP-hard, even for known deadlines (denoted $KDS(AE)^2$) (Shperberg, A. Coles, et al. 2019).

2.3 Basic Greedy Scheme

As $S(AE)^2$ is NP-hard, Shperberg, A. Coles, et al. 2019 used insights from a diminishing returns result to develop a greedy scheme. They restrict their attention to linear contiguous allocation policies: schedules where the action taken at time t does not depend on the results of the previous actions, and where each process receives its allocated time contiguously. Using the p.m.f. $m_i(t') = M_i(t') - M_i(t' - 1)$, the probability that process i finds a timely plan when allocated t_i consecutive time units *beginning* at time t_{b_i} is:

$$s_i(t_i, t_{b_i}) = \sum_{t'=0}^{t_i} m_i(t')(1 - D_i(t' + t_{b_i})) \quad (E.1)$$

Example E.1

Let $m_1 \sim [0.5 : 2; 0.5 : 5]$, i.e. process 1 requires 2 time units or 5, equally likely; and $d_1 = 2$ with probability 1 (known deadline). Then $s_1(2, 0) = 0.5$, and $s_1(2, 1) = s_1(1, 0) = 0$. Thus, process 1 delivers a timely solution with probability 0.5 given 2 time units at $t_{b_1} = 0$; and is useless with $t_1 < 2$ or $t_{b_1} \geq 1$.

When considering linear contiguous policies, we need to allocate t_i, t_{b_i} pairs to all processes (with no allocation overlap). Note that overall a timely plan is found if at least one process succeeds, that is, overall failure occurs only if all processes fail. Thus, to maximize the probability of overall success P_s (over all possible linear contiguous allocations), we need to allocate t_i, t_{b_i} pairs so as to maximize:

$$P_s = 1 - \prod_i (1 - s_i(t_i, t_{b_i})) \quad (E.2)$$

2. Background

Using $LPF_i(\cdot)$ ('logarithm of probability of failure') as shorthand for $\log(1 - s_i(\cdot))$, we note that P_s is maximized if the sum of the $LPF_i(t_i, t_{b_i})$ is minimized and that $-LPF_i(t_i, t_{b_i})$ behaves like a utility that we need to maximize. For known deadlines, we can assume that no policy will allocate processing time after the respective deadline. We will use $LPF_i(t)$ as shorthand for $LPF_i(t, 0)$.

To bypass the problem of non-diminishing returns, the notion of *most effective computation time* for process i under the assumption that it begins at time t_b and runs for t time units was defined as:

$$e_i(t_b) = \operatorname{argmin}_t \frac{LPF_i(t, t_b)}{t} \quad (\text{E.3})$$

$e_i(t_b)$ is a generalization of e_i from Shperberg, A. Coles, et al. 2019 which equals $e_i(0)$ here. We use e_i to denote $e_i(0)$ below.

Example E.2

For process 1 from Example E.1 we have (log base 2): $LPF_1(t_1, t_{b_1}) = -1$ for all $t_1 \geq 2$ and $t_{b_1} = 0$, and $LPF_1(t_1, t_{b_1}) = 0$ for all other t_1 and t_{b_1} configurations; so $e_1(0) = 2$.

Since not all processes can start *now*, intuitions from diminishing returns are: to prefer process i that has the best utility per time unit, i.e. such that $-LPF_i(e_i)/e_i$ is greatest. Still, allocating time now to process i delays other processes, so it is also important to allocate the time now to processes with an early deadline. Shperberg, A. Coles, et al. 2019 thus suggested the following greedy algorithm: Iteratively allocate t_u units of computation time to process i maximizing:

$$Q(i) = \frac{\alpha}{E[D_i]} - \frac{LPF_i(e_i)}{e_i} \quad (\text{E.4})$$

where α and t_u are positive-valued parameters, and $E[D_i]$ is the expectation of the random variable with CDF D_i (a slight abuse of notation). α trades off between preferring earlier deadlines (large α) and better performance slopes (small α).

Example E.3

Add to Example E.2 process 2 with $d_2 = 4$ and $m_2 \sim [0.75 : 2; 0.25 : 20]$. Note that process 1 cannot be delayed as $d_1 = 2$, but process 2 can be delayed by 2 time units. Thus, the optimal policy (which results in $P_s = 7/8$) is to start with process 1 and then move to process 2. However, $LPF_2(2, t_{b_2}) = -2$ and $e_2(t_{b_2}) = 2$ for $t_{b_2} \leq 2$, with $-\frac{LPF_2(e_2)}{e_2} = 1$, better than $-\frac{LPF_1(e_1)}{e_1} = 0.5$. Nonetheless, with $\alpha = 10$, we have initially $Q(1) = 10/3 + 0.5 = 23/6$ and $Q(2) = 10/4 + 1 = 21/6$, so start with process 1, as required.

3 New Metareasoning Schemes

Our new results for the $S(AE)^2$ are a (pseudo) polynomial-time algorithm for the case of known deadlines (dropping the diminishing returns requirement), and a better justified greedy scheme that also works better in practice.

3.1 DP Solution for Known Deadlines

Although $S(AE)^2$ is NP-hard, Shperberg, A. Coles, et al. 2019 showed that under the additional restriction of diminishing returns (non-decreasing logarithm of probability of failure) an optimal schedule can be found in (pseudo) polynomial time. However, planning processes do not have diminishing returns. We therefore examine deliberation scheduling when the diminishing returns assumption is removed.

For $KDS(AE)^2$ (known deadlines $S(AE)^2$), it is sufficient to examine linear contiguous allocation policies (Shperberg, A. Coles, et al. 2019). We extend this result by showing that restricting the schedules to ones with processes sorted by an increasing order of deadlines is still optimal:

Theorem E.1

Given a $KDS(AE)^2$ problem, there exists a linear contiguous schedule with processes sorted by a non-decreasing order of deadlines that is optimal.

Proof. Given an optimal linear contiguous policy P , we show that it can be rearranged to have non-decreasing order of deadlines. Let consecutive processes $i, i + 1$ in P be

3. New Metareasoning Schemes

the first two processes with deadlines $d_{i+1} < d_i$. Let P' be the same as P but with the starting times of i and $i + 1$ exchanged. Since P' differs from P only in the order of i and $i + 1$, all processes allocated either before i or after $i + 1$ are unaffected, as they maintain the same starting time and duration allocations. In addition, since the time allocation for both processes is unchanged, the only way to decrease the solution quality is by violating the deadline of either i or $i + 1$. Making process $i + 1$ start earlier cannot cause it to violate its deadline. Furthermore, $d_i > d_{i+1}$ and d_{i+1} was not violated in P , therefore, d_i also cannot be violated in P' . Thus, the success probability for P' is not less than that of P . These steps can be repeated until an optimal schedule sorted by a non-decreasing order of deadlines is obtained. \square

Theorem E.1 can be used to obtain a DP scheme.

Theorem E.2

For known deadlines, an optimal schedule can be found in time polynomial in n, d_n using DP according to

$$OPT(t, l) = \max_{0 \leq j \leq d_l - t} (OPT(t + j, l + 1) - LPF_l(j)) \quad (E.5)$$

Proof outline. We show by induction that $OPT(t, l)$ is the utility of the optimal linear contiguous ordered (LCO) schedule for processes l through n for the 'remaining' time $d_n - t$. The base cases are when $l = n + 1$ (no processes to be assigned), thus $OPT(\cdot, n + 1) = 0$; or no time remaining, thus $OPT(d_n, \cdot) = 0$. Assume that $OPT(t', l')$ is the utility of the optimal LCO schedule for every $t' \geq t$ and $l' > l$. To compute the utility of the optimal LCO schedule for processes l through n for the "remaining" time $d_n - t$, we need to consider all time allocations j for process l . For each time allocation j , we add the reward resulting from the allocation, which is $-LPF_l(j)$, to the best utility over every possible optimal schedule of processes $l + 1$ through n , given that j time was already allocated. The latter is exactly $OPT(t + j, l + 1)$ according to the inductive hypothesis. Note that any allocation beyond the deadline of process l (d_l) is wasteful and cannot contribute to the utility. Thus, we can only consider time allocations j between 0 and $d_l - t$. This computation is exactly the right-hand side of Equation E.5.

Thus, $OPT(0, 1)$ is the maximal utility among all LCO schedules for processes from 1 to n starting at $t = 0$. Due to Theorem E.1, $OPT(0, 1)$ is the maximal utility among all schedules, indicating an optimal solution to $KDS(AE)^2$. \square

If the representation of the M_i is explicit, the algorithm evaluating Equation E.5 in descending order runs in polynomial time. Otherwise, it is pseudo-polynomial and can be approximated in polynomial time.

3.2 Delay-Damage Aware Greedy Scheme

Although the pseudo-polynomial algorithm for $KDS(AE)^2$ is reasonably fast, it is still too slow for metareasoning in a planner. Furthermore, it is not applicable when deadlines are unknown. Thus, we next examine a new greedy scheme, beginning with pointing out shortcomings of the greedy scheme from Shperberg, A. Coles, et al. 2019; namely, using the proxy $E[D_i]$ in the value $Q(i)$ is somewhat ad-hoc and fails when the deadline distribution has a large variance.

Example E.4

Modify Example E.3 so that process 2 deadline $d_2 \sim [0.75 : 2; 0.25 : 10]$, thus $E[D_2] = 4$ as before. Then $Q(1), Q(2)$ are unaffected and basic greedy behaves the same. However, now both process 1 and process 2 cannot be delayed. Therefore, the optimal policy now is to schedule only process 2 for $P_s = 9/16$.

A more principled scheme can use the utility per time unit as in $Q(i)$, but with a first term that is better justified theoretically. The first term of $Q(i)$ is used for prioritizing processes with early deadlines, as any delay might prevent them from completing their computation before their deadline, even if they *would* have been timely had they been scheduled for processing immediately. Therefore, instead of the first term, it makes sense to provide a measure of the ‘utility damage’ to a process i due to delaying its processing start time from time 0 to time t_d . Consider the case of two processes and allocating contiguous processing time to each of them, each equal to their most effective computation time e_i . For simplicity assume $e_1 = e_2 = e_{1,2}$. In this case, we can write down the ‘utility’ (negative logarithm of probability of failure) for first running process 1 and then process 2, as:

$$U(1,2) = -LPF_1(e_{1,2}, 0) - LPF_2(e_{1,2}, e_{1,2})$$

3. New Metareasoning Schemes

where the second term is for LPF for process 2 delayed to the end of the run of process 1. Likewise, for process 2 first:

$$U(2,1) = -LPF_2(e_{1,2},0) - LPF_1(e_{1,2},e_{1,2})$$

Since $e_1 = e_2 = e_{1,2}$, we can re-normalize the utility terms by adding $LPF_2(e_{1,2},e_{1,2}) + LPF_1(e_{1,2},e_{1,2})$ to get:

$$U'(1,2) = -LPF_1(e_{1,2},0) + LPF_1(e_{1,2},e_{1,2}) \quad (E.6)$$

$$U'(2,1) = -LPF_2(e_{1,2},0) + LPF_2(e_{1,2},e_{1,2})$$

U' is the difference in utility we could get for a process when it is run at time 0, minus the utility it can achieve if delayed by $e_{1,2}$ time units, and thus we call U' the ‘utility loss of delaying the process by $e_{1,2}$ time units’. The advantage of using U' as a measure of process i is that its value in equation E.6 depends only on process i . Although this optimality argument does not necessarily extend to more than 2 processes or to non-contiguous schedules, it has the advantage of correctly addressing the deadline distributions.

Example E.5

In Example E.3, $LPF_2(2,0) = LPF_2(2,2) = -2$ and $LPF_1(2,0) = -1$ but $LPF_1(2,2) = 0$. So $U'(1,2) = 3 > U'(2,1)$ and process 1 is first as needed. In Example 4, $LPF_2(2,2) = 0$, $LPF_2(2,0) = \log(7/16)$, so $U'(2,1) > U'(1,2)$ and process 2 is run as required.

In practice, a greedy scheme assigns some $t_u < e_i$ time units to process i at a time. As the most effective time e_i is not assigned in one chunk, it makes sense to assign time in order of the utility slope available if we allow process i to run now, rather than the total utility. In other words, we will use the utility slope as a proxy for the potential gain. This was done in the second term in Equation E.4 in the basic greedy scheme, and thus we still prefer a process i that maximizes:

$$slopenow_i = -LPF_i(e_i,0)/e_i$$

However, as suggested by Equation E.6, a measure of the urgency of process i w.r.t. the deadlines is in certain cases proportional to the utility loss due to delaying the process. The utility slope after delay of process i by t_u is:

$$slopelater_i = -LPF_i(e_i(t_u), t_u) / e_i(t_u)$$

The higher the slope after delay, the greater the gain, thus the smaller the loss, and the lower the urgency. Thus, this term is to be minimized.

Since the time t_u allocated at each round is not equal to e_i ; and since with more than two processes the delay a process suffers is unknown, the tradeoff between these terms is not necessarily equal. We use an empirically determined constant multiplier γ to balance between exploiting the current process reward from allocating time to process i now and the loss in reward due to delay. Thus, the delay-damage aware (DDA) greedy scheme is to assign, at each processing allocation round, t_u time to the process i that maximizes:

$$Q'(i) = \frac{\gamma \cdot LPF_i(e_i(t_u), t_u)}{e_i(t_u)} - \frac{LPF_i(e_i, 0)}{e_i} \quad (\text{E.7})$$

3.3 Evaluation on S(AE)²

We performed an empirical evaluation in order to assess the effectiveness of the different metareasoning schemes on the abstract problem, and to decide which scheme to integrate into an actual planner. Following Shperberg, A. Coles, et al. 2019, we generated problems with performance profiles and deadline distributions based on a variety of distributions: Uniform (U), with minimal range value $a = 1$ and maximal range value b uniformly drawn from $\{[5, 10], [50, 100], [100, 200], [150, 300]\}$, we denote the set of possible $[a, b]$ ranges by R ; Boltzmann (B), truncated exponential distribution with the diminishing return property, using a $\lambda \in \{0.1, 1, 2\}$ and range drawn from R ; Truncated Normal Distribution (N) with $\mu \in \{5, 50, 100, 150\}$, $\sigma \in \{1, 5, 10\}$, and range drawn from R ; and Planner (P), distributions collected from search trees of the Robocup Logistics League (Niemueller, Lakemeyer, et al. 2015) domain generated by the OPTIC planner. To acquire the planner distributions, A* was executed from each node of the dumped search tree. The result of each of these searches provides the

3. New Metareasoning Schemes

number $N(v)$ of expansions necessary to find the goal under a node v . These numbers were binned separately for each $(h(v), g(v))$ pair. Then, a set of nodes V was selected randomly from the trees, each node standing for a process. For each such $v \in V$, the list of numbers of expansions in the bin corresponding to $g(v)$ and $h(v)$ was treated as a distribution over completion times (in terms of number of expansions). Likewise for creating the latest start times for the resulting plan (the deadline distribution). Experiments were run with unknown deadlines, and with a known deadline randomly drawn from the corresponding distribution before execution.

The algorithms we evaluated are: the optimal MDP solution (whenever possible), computed using the Bellman equation in value-determination; the basic greedy scheme using $\alpha \in \{0, 0.2, 0.5, 1, 20\}$; the DDA scheme using $t_u = 1$ and $\gamma \in \{0, 0.2, 0.5, 0.75, 1, 2, 10, 100\}$; and the dynamic programming (DP) scheme, treating the expected deadline of each process as its true deadline when the deadlines are unknown. The reported results are only the best parameter values: $\alpha = 0$ for basic greedy and $\gamma = 1$ for DDA.

Evaluating the quality of a solution (policy) is not trivial, especially for adaptive policies. We ran the algorithms on each setting for 500 attempts and reported the fraction of successful runs out of the total number of attempts as the solution quality. Since this policy evaluation process introduced noise, we have measured the standard deviation (std) of the solution quality (not reported in the table); the overall std was small (± 0.02), therefore, the introduced noise does not affect the trends reported below. The results are shown in Table E.1, in which the top and bottom halves of the table contains results of the known and unknown deadlines cases respectively. Q indicates the solution quality (success probability achieved by the policy created by the algorithm); T is the metareasoning runtime in seconds. ‘Avg’ rows give average solution quality and geometric mean of the runtimes. DP is optimal (and therefore the best) when the deadlines are known; however, it performed poorly for unknown deadlines. For the unknown deadlines case, and on average across both cases, DDA achieved the best solution quality, outperforming the basic greedy scheme. However, DDA had the worst metareasoning runtime, except for the MDP, which we must consider when integrating it with a planner.

Dist	# pr	MDP		Basic		DDA		DP	
		Q	T	Q	T	Q	T	Q	T
B	2	0.72	0.1	0.63	0.00	0.66	0.00	0.72	0.00
	5			0.71	0.00	0.78	0.01	0.83	0.01
	10			0.63	0.01	0.75	0.08	0.88	0.09
	100			0.82	0.07	0.99	0.19	1.00	0.21
N	2	0.61	7.7	0.55	0.00	0.57	0.00	0.61	0.00
	5			0.84	0.00	0.84	0.01	0.84	0.01
	10			0.92	0.01	0.93	0.06	0.93	0.08
	100			1.00	0.02	1.00	0.23	1.00	0.28
U	2	0.66	1.5	0.62	0.00	0.64	0.01	0.66	0.02
	5			0.85	0.02	0.85	0.09	0.91	0.10
	10			0.97	0.03	0.98	0.22	0.98	0.31
	100			1.00	0.08	1.00	0.78	1.00	0.71
P	2	0.82	11.7	0.71	0.00	0.77	0.00	0.82	0.00
	5			0.77	0.00	0.82	0.02	0.83	0.03
	10			0.98	0.01	1.00	0.09	1.00	0.1
	100			1.00	0.05	1.00	0.27	1.00	0.24
Known, Avg.				0.81	0.02	0.85	0.13	0.87	0.14
B	2	0.69	188.1	0.62	0.01	0.65	0.03	0.50	0.03
	5			0.65	0.02	0.77	0.12	0.57	0.14
	10			0.71	0.06	0.75	0.48	0.72	0.55
	100			0.70	0.21	0.82	1.19	0.68	1.43
N	2	0.69	25.6	0.61	0.01	0.69	0.04	0.47	0.04
	5			0.70	0.02	0.87	0.10	0.66	0.13
	10			0.64	0.05	0.72	0.44	0.55	0.49
	100			0.76	0.19	0.84	2.03	0.73	1.95
U	2	0.73	112.8	0.67	0.04	0.73	0.25	0.73	0.26
	5			0.69	0.19	0.78	1.26	0.45	1.12
	10			0.79	0.22	0.91	2.25	0.84	2.31
	100			0.85	0.88	0.89	7.83	0.74	7.50
P	2	0.81	20.6	0.62	0.00	0.77	0.01	0.63	0.01
	5			0.89	0.00	0.93	0.03	0.81	0.06
	10			0.9	0.05	0.9	0.38	0.88	0.39
	100			0.86	0.21	0.95	2.21	0.75	2.23
Unknown, Avg.				0.73	0.14	0.81	1.17	0.67	1.17
Total, Avg				0.77	0.08	0.83	0.65	0.77	0.65

Table E.1: Solution quality and runtime for different settings

4 Integrating DDA into a Planner

DDA was the best performing algorithm in Section 3.3, while the basic greedy scheme (with $\alpha = 0$) had the best runtime. Since DDA with $\gamma = 0$ is equivalent to basic greedy with $\alpha = 0$, it suffices to implement DDA in the planner. In order to do so, several issues must be addressed. First, the non-trivial task of obtaining the distributions; second, how to use the DDA scheme as the basis for search.

4.1 Estimating the Distributions

DDA needs the D_i and M_i distributions as input. To estimate these distributions, we leverage estimates easily obtained in the situated temporal planner on which we build (Cashmore et al. 2018). The planner uses the temporal relaxed planning graph (TRPG) (A. J. Coles et al. 2010) to estimate $E[D_i]$, and the distance to go (also from the TRPG) to estimate remaining search time (Dionne et al. 2011), which we treat as an estimate of $E[M_i]$.

However, to use the DDA greedy rule we must also estimate the whole distribution, rather than just the expected values. For $D_i(t)$, we simply use a step function that goes from probability zero to one when t is *equal* to the deadline of the relaxed plan (our estimation of $E[D_i]$). In order to estimate M_i , we use an online temporal difference learning technique (Thayer et al. 2011). Note that the true distance-to-go from some state s , denoted $d^*(s)$ is equal to $d^*(bc(s)) + 1$, where $bc(s)$ is the *best child* of state s , that is, the successor that is on the best path to the goal. If d is a heuristic estimate of d^* , then the one-step error of d at s is defined as $\epsilon_d(s) = d(bc(s)) + 1 - d(s)$. The average error of d is then estimated by the average one-step error, either throughout the entire search space or along a specific path.

We extend these ideas to estimate M_i . First, as we have the one-step errors for all states observed so far, we can estimate the distribution of one-step errors of distance to go, denoted O_d . Since these errors accumulate along the path to the goal, to estimate the distribution of the distance to go from state s , we convolve O_d with itself $d(s)$ times. We then multiply this distribution by the average expansion delay divided by the expansion rate (Cashmore et al. 2018), which measures how much time passes, on average, between generating a state and expanding it.

One important implementation detail is that, when the expansion delay is small, the remaining-search-time estimate may unrealistically have zero probability of failure. Thus we add to the M_i distribution an infinite remaining search time outcome, with probability $pf_{\min} = 0.0001$.

4.2 Searching with DDA

For search, each state i on the open list can be treated as a process. Hence, the $Q'(i)$ value (Equation E.7) is maintained for each state i , and computation time is allocated to a state with highest $Q'(i)$. Recalling that the DDA scheme is based on allocating t_u units of computation time, thus we perform t_u expansions in the *subtree* rooted at i ; after t_u expansions, the non-expanded (frontier) nodes in this subtree are added to the open list, and another state is chosen according to Q' .

One important aspect of searching based on Q' values is that these values are not static: during search, the distribution O_d changes, and time passes, all of which change the $Q'(i)$ values. Hence, we recompute them every t_u expansions, i.e. whenever the next subtree to allocate time to is to be chosen. To reduce the metareasoning overhead (for instance, to keep it under some desired proportion of planner runtime), one could consider changing the frequency of this during search, but as we have not found the overheads to be excessive in our experiments, we leave this for future work.

Since DDA needs statistics, such as one-step error estimates, that are unavailable early in the search, we actually start off by using the baseline weighted A* search (Cashmore et al. 2018). The DDA ordering kicks in only after n_{exp} expansions (an algorithm parameter).

In some domains the distribution information is unhelpful, e.g. if the deadline is very far, in which case probability of success for many nodes is close to 1, both before and after delay. Alternately, the deadline may be very close, in which case the probability of success is close to 0, regardless of delay for many nodes. In such cases, many leading Q' values are identical, and our scheme may become erratic. In such cases we break ties or near-ties by preferring nodes with lower f value. This tie-breaking was implemented by actually expanding nodes with the highest $Q'(i) + \beta f(i)$, where a very small $\beta = -0.000001$ was used throughout.

One could also imagine a more sophisticated scheme in which the algorithm mon-

itors the distribution of Q' values to assess whether they contain useful information or suffer from substantial estimation error. This interesting issue remains for future work.

5 Empirical Evaluation

To evaluate the DDA metareasoning scheme, we implemented it on top of the situated temporal planner of Cashmore et al. 2018, which itself is implemented on top of OPTIC (Benton et al. 2012). As a baseline, we use the heuristic search scheme of Cashmore et al. 2018. We used the same set of benchmarks as this prior work, which includes all IPC domains with Timed Initial Literals (TILs), representing constraints on absolute time; as well as 200 instances from the Robocup Logistics League (RCLL) (Niemueller, Karpas, et al. 2016), a simulated robotic manufacturing setting – divided into instances with 1 or 2 robots, and a Turtlebot office delivery domain with tight deadlines from Cashmore et al. 2019. All experiments were run on a server with 72 Intel Xeon E5-2695 CPUs, using up to 64 processes in parallel (Tange 2011) and a 3GB memory limit.

Domain	baseline		DDA		DDA(nexp = 1)		DDA($\gamma = 0$)		DDA(hs)		DDA (dom tuned)	
airport	19.0	(19–19)	20.0	(20–20)	20.0	(20–20)	20.0	(20–20)	19.1	(19–20)	20.5	(19–21)
pw-nt	4.0	(3–4)	4.0	(3–5)	4.5	(3–5)	4.0	(3–4)	4.0	(4–4)	3.9	(3–5)
rcll 1	37.7	(37–40)	73.7	(53–92)	73.0	(65–91)	76.4	(69–81)	34.6	(15–75)	83.9	(59–99)
rcll 2	1.0	(1–1)	4.0	(2–23)	1.1	(0–14)	2.0	(2–2)	1.8	(1–7)	2.7	(0–13)
sat cmplx	5.0	(5–5)	5.0	(5–5)	4.8	(4–5)	5.0	(5–5)	5.0	(5–5)	3.8	(2–5)
sat tw	5.0	(5–5)	5.0	(5–5)	5.1	(5–6)	5.0	(5–5)	5.0	(5–5)	3.6	(3–5)
trucks	6.0	(6–6)	6.9	(6–9)	6.3	(6–7)	7.5	(7–8)	6.5	(6–7)	5.7	(5–8)
turtlebot	14.0	(14–14)	12.5	(10–13)	13.0	(13–13)	8.0	(8–8)	14.0	(14–14)	13.0	(13–13)
umts-flaw	4.1	(4–5)	5.1	(5–6)	0.0	(0–0)	0.1	(0–1)	0.4	(0–4)	5.0	(5–5)
umts	48.0	(48–48)	45.5	(42–49)	44.2	(41–48)	43.8	(41–48)	41.5	(41–42)	45.7	(44–49)
TOTAL	143.8	(142–147)	181.7	(151–227)	172.0	(157–209)	171.6	(160–182)	131.7	(110–183)	187.7	(153–223)

Table E.2: Number of problems solved by each planner, shown as: average (solved by all 20 runs - solved by at least one run).

5.1 Comparing DDA to the Baseline

We begin by comparing DDA, with the default parameter values ($t_u = 100, \gamma = 1$, and $nexp = 1000$), to the baseline. These values were the result of an initial guess,

which we verified empirically. In situated temporal planning, the time elapsed during planning affects search decisions. This introduces noise due to multiple processes sharing the same CPU, as well as features such as Intel Turbo Boost, which vary the speed of the CPU according to load. Thus, in this experiment we ran the planner 20 times on each planning problem. Each run was limited to 200 seconds of CPU time.

Table E.2 shows the (average) number of problems solved in each domain for each planner. Each entry shows the average number of problems solved in that domain by that planner (averaging over the 20 runs). Furthermore, in parentheses we give the number of problems solved by *all* 20 runs (of the given planner in the given domain) and the number of problems solved by *at least one* of the 20 runs (of the given planner in the given domain). These numbers serve as a confidence interval of sorts, as they indicate ease of replication, thus we denote them by low bar and high bar, respectively.

Looking only at the DDA and baseline columns for now, observe that DDA solves 38 more problems than the baseline. Interestingly, the low bar of 151 for DDA is higher than the high bar for the baseline – that is, there were 151 instances that were solved by all 20 runs of DDA, compared to 147 that were solved by at least one run of the baseline.

Also note that the “noise” (the difference between the high bar and the low bar) for DDA is much higher than the baseline. This is because the baseline only uses elapsed search time to prune search nodes, but otherwise keeps the same ordering between nodes based on f -values. On the other hand, DDA uses elapsed time to compute Q' , thus changing the ordering between nodes.

However, looking at the total coverage could be misleading, as the numbers of problems in each domain are different — and the number of solvable problem even more so (varying from about 5 to almost 100). Thus, we also count in how many domains DDA outperformed the baseline. DDA beats the baseline in 4 domains: airport, RCLL (with 1 and 2 robots), trucks, and UMTS flaw, and loses only in 2 domains: turtlebot and UMTS.

To further analyze this result, we exploited the fact that we have 20 different runs for each planner on each problem. Thus, we can perform a statistical significance test on the number of times each planner solved each problem, and check whether DDA is statistically significantly better than the baseline on each problem, whether

5. Empirical Evaluation

Domain			Domain	baseline				DDA			
	baseline	DDA		0.25	0.5	2	4	0.25	0.5	2	4
airport	0	1	airport	19.0	19.0	19.0	19.0	20.0	20.0	20.0	20.0
pw-nt	0	0	pw-nt	0.0	0.0	9.0	12.1	0.0	0.0	10.0	12.2
rc1 1	7	48	rc1 1	4.2	9.0	39.4	46.3	10.1	35.8	70.7	72.6
rc1 2	0	4	rc1 2	0.0	0.0	1.0	2.0	0.6	0.2	5.4	8.4
sat cmplx	0	0	sat cmplx	0.0	0.0	5.0	5.0	0.0	0.0	5.0	5.0
sat tw	0	0	sat tw	0.0	0.0	6.0	7.0	0.0	0.0	6.0	7.4
trucks	0	2	trucks	0.0	0.0	7.0	13.0	0.0	0.0	9.1	13.0
turtlebot	2	0	turtlebot	0.0	4.1	14.0	14.0	0.0	4.0	14.0	14.0
umts-flaw	0	1	umts-flaw	0.0	4.8	4.3	2.2	3.5	5.5	5.0	5.0
umts	7	1	umts	16.0	33.7	42.0	42.0	16.0	35.4	42.9	42.0
TOTAL	16	57	TOTAL	39.2	70.6	146.7	162.6	50.2	100.9	188.1	199.6

Table E.3: Statistically significant wins

Table E.4: Coverage with different TIL multipliers

the baseline is statistically significantly better than DDA, or whether there is no statistically significant difference. Specifically we used the Mann-Whitney U-test (Mann and Whitney 1947), a non-parametric test that checks if one variable is more likely to have a higher value than another. Table E.3 reports the number of problems in each domain in which the baseline or DDA were statistically significantly better than the other. The results here correspond well with the domains listed above, and show that in total DDA is statistically significantly better than the baseline on 3.5 times more problems.

5.2 DDA Ablation Studies

Having seen that DDA outperforms the previous work in situated temporal planning, we performed some ablation studies to test which of the features of DDA contribute the most to its success. We considered the following planner variants:

DDA(nexp = 1) immediately starts with DDA, instead of waiting 1000 node expansions for the estimates to stabilize.

DDA($\gamma = 0$) uses $\gamma = 0$ to compute $Q'(i)$, thus ignoring the slope later component of Q' .

DDA(hs) uses standard heuristic search based on Q' , instead of doing t_u expansions in the chosen subtree.

In all of these variants, the other parameters were kept the same as the default configuration. Table E.2 also shows the results for these variants. The results show that, indeed, every one of these features contributes somewhat to the success of DDA, with the focused search leading to many more solved problems. Interestingly, in the turtlebot domain, where DDA loses to the baseline, the pure heuristic search variant solves all the problems — the same as the baseline. This is likely because there are dead ends in the domain (due to deadlines expiring), and heuristic search does better at avoiding these.

5.3 Automated Parameter Tuning for DDA

The results we presented above were based on hand-chosen default parameter setting for DDA. On the one hand, this is only one point in a large space of possible parameter settings, and thus DDA has the potential for even better performance. On the other hand, there could be a concern that these parameter settings were chosen based on their performance on the problem domains being evaluated, and thus there is overfitting in the process.

Therefore, we also show that it is possible to automatically find good settings for each domain automatically, using SMAC – Sequential Model-Based Algorithm Configuration (Hutter et al. 2011; Lindauer et al. 2017). To avoid overfitting, we divided each domain into 2 folds – the evenly numbered problems and the odd numbered problems (this was done to try to maintain the same distribution of problem sizes in both folds). We then used SMAC — specifically, Bayesian Optimization and HyperBand (Falkner et al. 2018; Li et al. 2017) — to find the best configuration for each fold. This configuration was then used to evaluate the other fold. The results reported here use the configuration trained on the even problems to measure the performance of DDA on the odd problems, and vice versa — thus avoiding overfitting.

We gave SMAC 72 hours to find the best configuration. We fixed the values of \min_{pf} and β , as well as the decision to use the subtree-focused search. Thus we searched for values for the remaining parameters: γ was limited to values between -10 and 10, t_u to values between 10 and 1000 (on a logarithmic scale), and n_{exp} to values between 1 and 10,000 (also on a logarithmic scale). The score for each run was the total search time in seconds (if the problem was solved), or 2^{31} if the problem was

not solved. To overcome noise, each problem was run 3 times, and the average score was used.

Table E.2 also reports the results of DDA tuned for each domain (these are the combined results from both configurations in each domain). Overall, the domain-tuned version of DDA outperforms DDA in coverage. However, in a few domains, performance is actually worse. Notably, these are the domains where DDA solves very few problems to begin with (up to 7), and thus the signal for training is rather weak. On the other hand, in domains where DDA already solved 10 or more problems, the domain-specific parameter tuning helped improve performance. We believe this problem could be alleviated by using a random problem generator, but this is beyond the scope of this paper.

5.4 Evaluating the Impact of Tight Deadlines

We conclude the empirical evaluation by examining more closely when DDA outperforms the baseline. First, note that when the deadlines are very loose, there should be no difference between DDA and the baseline, as they will both have time to explore the search space. Second, when the deadlines are very tight, both approaches are likely to fail to find a solution before the deadline — in fact, this is the case in *pipesworld-no-tankage* (pw-nt). Thus, we expect DDA to outperform the baseline in the “Goldilocks” zone, where deadlines are tight, but not too tight.

To evaluate this claim empirically, we ran the baseline and DDA on the same problem instances as before, modified by multiplying the TILs in the problem by a factor. We tried factors of 0.25 and 0.5 (for tighter deadlines) and 2 and 4 (for looser deadlines). The number of problems solved by each planner in each domain is shown in Table E.4, where the number in each cell is the average of 10 runs.

First, observe that DDA outperforms the baseline overall for all multipliers. Second, looking at the *pipesworld* domain (pw-nt), we can see the phenomenon we described above. For multiplier of 0.25 and 0.5, both approaches solve 0 (and from Table E.2, for a multiplier of 1, both solve 4). When the multiplier is 2, DDA solves 1 more problem than the baseline, but when the multiplier is 4 (and deadlines are now very loose), the difference becomes only 0.1. A similar phenomenon occurs in *rc11* 2 and in *trucks*.

6 Discussion

We advanced a formal metareasoning approach for situated temporal planning. We note that optimizing the time allocation in an algorithm portfolio with known probabilistic performance profiles is a special case where there is a common deadline by which all processes must deliver the result, namely the (usually) known time limit per instance. Despite using rather crude estimates, the enhanced planner empirically outperforms previous work. This is an important step in making situated temporal planning practical. It also demonstrates the enduring power of metareasoning as a productive perspective on rational resource-bounded problem-solving.

We assumed here that the plan must be completed before execution. However, with very tight deadlines, it may be necessary to start execution before a complete plan is found. We intend to expand the metareasoning scheme to a (more complicated) model which supports execution of actions while still searching for a plan. Finally, we mean to enrich OPTIC with other metareasoning schemes (Shperberg, Felner, et al. 2020) tailored for the problem of minimizing expected cost (rather than trying to maximize the probability of finding a solution).

Acknowledgements

This research was supported by Grant No. 2019730 from the United States-Israel Binational Science Foundation (BSF) and by Grant No. 2008594 from the United States National Science Foundation (NSF). Project also funded by the European Union’s Horizon 2020 Research and Innovation programme under Grant Agreement No. 821988 (ADE), by a Grant from the GIF, the German-Israeli Foundation for Scientific Research and Development, by ISF grant #844/17, and by the Frankel center for CS at BGU.

References of Paper E

- [BCC12] J. Benton, Amanda Jane Coles, and Andrew Coles. “Temporal Planning with Preferences and Time-Dependent Continuous Costs”. In: *ICAPS*. Atibaia, São Paulo, Brazil: AAAI Press, 2012, pp. 2–10.
- [Cas+18] Michael Cashmore et al. “Temporal Planning while the Clock Ticks”. In: *ICAPS*. AAAI Press, 2018, pp. 39–46. URL: <https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17724>.
- [Cas+19] Michael Cashmore et al. “Replanning for Situated Robots”. In: *ICAPS*. AAAI Press, 2019, pp. 665–673.
- [CC03] Stephen Cresswell and Alexandra Coddington. “Planning with Timed Literals and Deadlines”. In: *Proceedings of 22nd Workshop of the UK Planning and Scheduling Special Interest Group*. 2003, pp. 23–35.
- [Col+10] Amanda Jane Coles et al. “Forward-Chaining Partial-Order Planning”. In: *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2010, pp. 42–49. URL: <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1421>.
- [DMP91] Rina Dechter, Itay Meiri, and Judea Pearl. “Temporal Constraint Networks”. In: *Artificial Intelligence* 49.1-3 (1991), pp. 61–95.
- [DTR11] Austin J. Dionne, Jordan Tyler Thayer, and Wheeler Ruml. “Deadline-Aware Search Using On-Line Measures of Behavior”. In: *Proceedings of the Symposium on Combinatorial Search (SoCS)*. 2011, pp. 39–46.
- [EH04] Stefan Edelkamp and Jörg Hoffmann. *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition*. Tech. rep. 195. University of Freiburg, 2004.
- [FKH18] Stefan Falkner, Aaron Klein, and Frank Hutter. “BOHB: Robust and Efficient Hyperparameter Optimization at Scale”. In: *ICML*. Vol. 80. *Proceedings of Machine Learning Research*. PMLR, 2018, pp. 1436–1445.

- [FL03] Maria Fox and Derek Long. “PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains”. In: *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), pp. 61–124. DOI: 10.1613/jair.1129. URL: <http://dx.doi.org/10.1613/jair.1129>.
- [HHL11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *LION*. Vol. 6683. Lecture Notes in Computer Science. Springer, 2011, pp. 507–523.
- [Kor90] Richard E. Korf. “Real-time Heuristic Search”. In: *Artificial Intelligence* 42.2-3 (1990), pp. 189–211.
- [Li+17] Lisha Li et al. “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization”. In: *J. Mach. Learn. Res.* 18 (2017), 185:1–185:52.
- [Lin+17] Marius Lindauer et al. *SMAC v3: Algorithm Configuration in Python*. <https://github.com/automl/SMAC3>. 2017.
- [MW47] H. B. Mann and D. R. Whitney. “On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other”. In: *Annals of Mathematical Statistics* 18.1 (1947), pp. 50–60.
- [Nie+16] Tim Niemueller, Erez Karpas, et al. “Planning Competition for Logistics Robots in Simulation”. In: *ICAPS Workshop on Planning and Robotics (PlanRob)*. London, UK, 2016.
- [NLF15] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. “The RoboCup Logistics League as a Benchmark for Planning in Robotics”. In: *WS on Planning and Robotics (PlanRob) at Int. Conf. on Aut. Planning and Scheduling (ICAPS)*. 2015.
- [RW91] Stuart J. Russell and Eric Wefald. “Principles of Metareasoning”. In: *Artif. Intell.* 49.1-3 (1991), pp. 361–395. DOI: 10.1016/0004-3702(91)90015-C. URL: [http://dx.doi.org/10.1016/0004-3702\(91\)90015-C](http://dx.doi.org/10.1016/0004-3702(91)90015-C).
- [Shp+19] Shahaf S. Shperberg, Andrew Coles, et al. “Allocating Planning Effort When Actions Expire”. In: *AAAI*. AAAI Press, 2019, pp. 2371–2378.

References of Paper E

- [Shp+20] Shahaf S. Shperberg, Ariel Felner, et al. “Bidirectional Heuristic Search: Expanding Nodes by a Lower Bound”. In: *IJCAI*. ijcai.org, 2020, pp. 4775–4779.
- [Tan11] O. Tange. “GNU Parallel - The Command-Line Power Tool”. In: *login: The USENIX Magazine* 36.1 (Feb. 2011), pp. 42–47. URL: <http://www.gnu.org/s/parallel>.
- [TDR11] Jordan Tyler Thayer, Austin J. Dionne, and Wheeler Ruml. “Learning Inadmissible Heuristics During Search”. In: *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*. 2011, pp. 250–257. URL: <http://aaai.org/ocs/index.php/ICAPS/ICAPS11/paper/view/2710>.
- [ZR96] Shlomo Zilberstein and Stuart J. Russell. “Optimal Composition of Real-Time Systems”. In: *Artif. Intell.* 82.1-2 (1996), pp. 181–213.

Paper F

Algorithm Selection in Optimization and Application to Angry Birds

Shahaf S. Shperberg, Solomon Eyal Shimony, Avinoam Yehezkel

The paper has been published in the
*ICAPS 2019: Proceedings of the Twenty-Ninth International Conference on Automated
Planning and Scheduling*
pp. 437–445, 2019.

© 2019 SoCS

The layout has been revised.

Abstract

Consider the MaxScore algorithm selection problem: given some optimization problem instances, a set of algorithms that solve them, and a time limit, what is the optimal policy for selecting (algorithm, instance) runs so as to maximize the sum of solution qualities for all problem instances?

We analyze the computational complexity of restrictions of MaxScore (NP-hard), and provide a dynamic programming approximation algorithm. This algorithm, as well as new greedy algorithms, are evaluated empirically on data from agent runs on Angry Birds problem instances. Results show a significant improvement over a hyper-agent greedy scheme from related work.

1 Introduction

Algorithm selection is of significant interest to researchers in AI, and other fields where more than one algorithm is available to solve problems under computational resource constraints (Rice 1976; Huberman et al. 1997; Xu et al. 2008). This paper examines a variant of algorithm selection ("MaxScore") where one needs to solve a set of optimization problems, with computational resource (assumed here to be time) limitation being over the entire set. This generalizes the standard setting handled in, e.g. SAT solver algorithm portfolios, where the time limit is separate for each individual problem instance.

Our original motivation for the MaxScore setting was combining multiple programs that compete in the AI Angry Birds (Copyright Rovio Entertainment) competition, on which we also base the empirical results of this paper. Angry birds is a physical simulation video game. In the Albirds competition, each agent program (or human) is presented with N previously unseen game levels (problem instances). The agents can select a level to play, where at each level the agent is presented with a screen-shot representing a physical simulation. The agent is supposed to kill off all the pigs with catapulted birds (shots), thereby completing (winning) the level. Points for completed levels are gained for destroying objects (pigs and blocks), and for using as few birds as possible. The agent may play any level as many times as desired, until

its overall allocated time (typically 10 or 30 minutes) expires. Level score is the maximum achieved in all attempts, with total score being the sum of level scores (typically 4 or 8 levels in past competitions).

Of the numerous AI techniques used in AI birds agent programs, to-date none have achieved near-human performance; each program has strengths and weaknesses in different areas of the game. Rather than attempting to create a better AI for Angry Birds, our goal here is one of meta-reasoning: use a portfolio of existing programs to play better, an idea suggested originally by (Stephenson and Renz 2017) with promising initial results. The focus of this paper is on how to do this combination optimally given the available information, in a decision-theoretic sense, rather than on the aspect of learning to fit the program to the problem instance. In addition to the research interest of this meta-reasoning problem, such an optimization might have an impact on algorithm portfolio optimization in general.

Informally (see Section 2 for the formal definition) we are given a set of problem instances (levels), to each of which we can apply any of a set of given algorithms (agents). Each such application uses up an unknown amount of time, and results in a score for the level that can be observed after each algorithm run terminates. Our meta-reasoning problem is to find a policy for selecting which agent to apply to which level, at any point in time, such that the sum of scores for all the levels at timeout is maximized. Note that in order to make this policy optimization well defined, one must specify some prior knowledge about scores and runtimes. In this paper we assume that these are specified by random variables with known distribution models.

Following the formal problem statement (Section 2), we analyze the computational complexity of restricted versions of our score maximization problem (Section 3.1: NP-hard even with *known* independent scores and runtimes). An approximation algorithm for one simple case is proposed, as well as faster greedy heuristic-based algorithms (Section 3.2). Empirical evaluation on scores and runtimes gathered from actual agent program runs show that using greedy expected improvement was near-optimal in practice (Section 4), and much better than the greedy scheme based on just expected score from (Stephenson and Renz 2017).

We then briefly examine the case of *unknown* independent distributions. These we treat as a distribution over performance profiles, i.e. a distribution over score

and runtime distributions (Section 5), with parameters that have to be learned from previously observed problem instances. This results in a model with induced dependencies, which makes the meta-level decision problem harder, in addition to the above learning problem. A naive learning method based on features from (Stephenson and Renz 2017; Tziortziotis et al. 2016) is proposed, and an empirical evaluation shows that our proposed greedy algorithm (coupled with belief updating) still performs well despite the low-quality learning.

2 Formal Problem Statement

This section describes the formal metareasoning MaxScore problem. In order to make the statement as general as possible, we abstract away from Angry Birds programs and levels, and present this as a sequential decision problem under uncertainty.

A MaxScore problem is a 4-tuple (I, A, T, P) : I is a set of problem instances to be optimized (game levels in Angry Birds); A is a set of algorithms (or agent programs); T is a time limit; and P is a known distribution model over problems in I , agents in A , that describes the (non-negative) score $S(a, l, i)$ achieved by agent $a \in A$ and the (positive valued) runtime $T(a, l, i)$ of a when applied to problem instance $l \in I$ during decision-making round i of the problem-solving task (or game play). Distribution P (also known as a *performance profile* (Zilberstein and Russell 1996)) can be defined in various ways. We consider the following cases for P :

1. P is deterministic.
2. P is a known distribution with independent variables.
3. P is a distribution with dependent variables (some of which are unobservable).

A policy π is a mapping from process histories to actions. The process is to select, at each round $i \geq 1$, an agent program $a(i)$ to apply to a problem instance (level) $l(i)$, given the past observations, according to π . The results $S(a(i), l(i), i)$ and $T(a(i), l(i), i)$ are observed after the selection at round i . Then i is incremented as we go to the next round. The process stops when we reach the time limit, i.e. at the first k such that:

$$\sum_{i=1}^k T(a(i), l(i), i) > T$$

The score of the process is the sum of maximal achieved scores for each problem instance, i.e.

$$S = \sum_{l \in I} \max_{i=1}^{k-1} S(a(i), l(i), i) \delta(l, l(i))$$

where $\delta(i, j)$ is the Kronecker delta (1 if $i = j$, 0 otherwise). The problem is: find a policy (mapping from process history, or alternately belief state and round number, to (agent, problem instance) pairs) that maximizes the expected value of S . This is the *sequential (adaptive)* decision making version of the problem.

We also consider, for computational complexity analysis, simpler *linear* settings of MaxScore, where the decision on which agents to run on which problem instances (and in which order) is decided only once, in advance. A policy in this setting is simply a sequence of (agent, problem instance) pairs. The linear setting is the same as the *batch* setting (also called *non-adaptive*) commonly used in the research literature (Shperberg and Shimony 2017), except that if the runtimes are not both known and deterministic, one must specify the ordering so as to have a well-defined policy (some of the agents may not get to run at all, due to runtimes uncertainty).

2.1 Performance Models

Assuming that the distribution model is Markov, the MaxScore problem is a POMDP with states defined by the current maximum scores vector cR and the play time elapsed. As the number of rounds is not known in advance, we define the problem as an indefinite horizon POMDP with terminal states being those where the sum of runtimes exceeds T ¹. The transition probabilities in this POMDP are trivially (and deterministically) defined given the score and runtime achieved in the current round (which in turn are defined by distribution P). The reward function is 0 except for transitions into terminal states. In general, POMDPs are intractable (PSPACE complete even if the belief space is finite). The actual complexity of MaxScore depends on the setting (sequential vs. linear/batch), on the performance profiles distribution model P , and on the size of sets I and A .

¹Because the timeout T is known, one could use a finite horizon POMDP with T time-slices, but this would necessitate many dummy transitions, for time points where an agent is running and no decision is to be made, which is inefficient.

3. Analysis: Known IID Case

A major issue is the performance profile (distribution model) P . Typically, exactly what scores and runtimes to expect is unknown, except by running the programs on the problem instances, which is too late to make the needed decision. However, we can run the programs on similar instances, collect statistics and learn a prediction model given instance features. Related work involved learning to predict the *expected score* of an agent in an unseen level (Stephenson and Renz 2017). However, as argued below, such information is insufficient for optimal choice: one may need to predict the *whole score distribution* (or equivalently, the expected improvement over each possible current score).

If the agent programs are effectively memoryless, i.e. attempt to solve the level from scratch each time they encounter it, then the order of the observed scores and runtimes is irrelevant. This behavior is reasonable for Angry Birds, as the game is effectively stochastic. Additionally (unlike search problems in most search domains), even if an agent knows the optimal play, it must still wait for the simulation to run its course, which usually takes on the order of one minute of real time per level attempt. Finally, the correlation coefficient between the actual score and time results measured over a few dozen instances was very small (≈ -0.015). Although this does not preclude a more complicated dependency between them, modeling score and time as independent is a reasonable approximation. We thus consider the agent scores and runtimes for a problem instance as i.i.d samples drawn from the distributions $P_S(a, l)$ and $P_T(a, l)$, respectively. If these distributions are known, the MaxScore problem is in fact an MDP, analyzed below.

3 Analysis: Known IID Case

We examine the computational complexity of some settings of the MaxScore problem. We begin with the fully deterministic case (scores and runtimes known in advance), and proceed to the independent case.

3.1 Complexity: Restricted Versions

We show that the MaxScore problem is NP hard even in the following extremely restricted cases:

$$OPT(rT, cR) = \max_{\substack{a \in A \\ l \in I}} \left(\sum_{\substack{r \in sp(P_S(a, l)) \\ \wedge t \leq rT}} \sum_{\substack{t \in sp(P_T(a, l)) \\ \wedge t \leq rT}} OPT(rT - t, R') \times P_T(a, l)[t] \times P_S(a, l)[r] + \sum_{\substack{t \in sp(P_T(a, l)) \\ \wedge t > rT}} \sum_{i=1}^m cR_i \times P_T(a, l)[t] \right)$$

Figure F.1: Optimal Solution to the MaxScore problem

1. Independent score distributions, deterministic runtimes, and only a single problem instance ($|I| = 1$).
2. Deterministic scores and runtimes, with $|A| = 1$ (but with $|I|$ unbounded).

We begin with the latter instance, as proving NP-hardness here is immediate through a straightforward mapping from the Knapsack problem. Simply map Knapsack item values to scores, item weights to runtimes, and the weight limit to T . Note that as the scores and runtimes are deterministic, in this case there is no difference between a linear setting, a batch setting, and a sequential setting of the problem.

With only one problem instance, we need to be more careful, but still get (see appendix for proof):

Theorem F.1

The linear setting of the MaxScore problem with independent score distributions, deterministic runtimes, and $|I| = 1$, is NP-hard.

We believe that the complexity of *sequential* setting with the same restrictions is at least as hard as the linear setting, but have not proved it. Also note that the *linear* setting MaxScore problem with $|I| = 1$ is non-trivial even if we further restrict it to *unit runtimes*. For example, using a natural greedy scheme that picks the agent with the best expected score can be suboptimal. Consider having agents A, B, C, with time limit $T=2$. Suppose A always scores 100, B scores 101 with probability 0.99 and 0 otherwise, and C scores 200 with probability 0.001, and 99 with probability 0.999. A greedy scheme would first pick A, as it has the certain value 100, higher than the expected scores of B and C. In fact the optimal policy is to pick B and C (expected score just over 101, whereas anything containing A achieves less than 101). The computational complexity of this setting of MaxScore is, as far as we know, an open problem.

3.2 Approximation Algorithms

Using dynamic programming schemes it is possible to achieve a pseudo polynomial algorithm for the case of (known distribution) independent scores and runtimes, and $|I|$ bounded by a constant, using the following scheme. (The assumption that $|I|$ is bounded by a constant is reasonable for e.g. Albirds competitions, where $|I|$ is 4 or 8.) Additionally, we are assuming that score items (e.g. score for killing a pig in Angry Birds) and runtimes are integer valued, and that the time span and score items have a unary representation in the input. The following dynamic programming value determination scheme (a variant of the Bellman equation) computes the optimal policy, and has a time complexity linear in the time span and the maximum score, and exponential in $|I|$.

Let $OPT(rT, cR)$ be the optimal solution value to the MaxScore problem with rT remaining time, and current maximum score vector $cR = \langle cR_1, \dots, cR_m \rangle$. The value determination recursive equation for $OPT(rT, cR)$ appears in Figure F.1, where $R' = \langle cR_1, \dots, \max\{cR_l, r\}, \dots, cR_m \rangle$, and $sp(D)$ is the support of distribution D . The value of $OPT(T, \langle 0, \dots, 0 \rangle)$ is that of the optimal policy at the initial state.

If the score distributions are continuous, or have too many values, we can round them into bins, achieving a $(1 - \epsilon)$ -approximation to the optimal policy. Likewise discretizing the runtime distributions is possible, but here near-optimality is not guaranteed. Although the dynamic programming approximation scheme can be computed in pseudo-polynomial time, it is still too computationally demanding to be practical. We would thus like to use a greedy scheme in practice, and the one that comes to mind immediately is to use the agent that has the best expected score, as essentially done in (Stephenson and Renz 2017). A slightly better scheme is to take into account the runtime, and use the ratio of expected score over expected runtime. However, it is easy to show that these schemes are far from optimal (as verified by the empirical results).

For example, suppose we have only one problem instance and two agents. We have already achieved a score of 10,000, and have time for exactly one more run. Suppose the first agent always scores 10,000. The second agent scores 100,000 with probability 0.05, and otherwise fails and scores 0, thus its expected score is 5,000. The above greedy schemes would select the first agent and always get 10,000, while the optimal

policy would obviously select the second agent, to possibly end up with 100,000 (with expected final score 10,500). An improved greedy scheme instead looks at the *expected improvement* to the score over the current score, i.e. the value:

$$E[S(a, l, i) - \max_{a \in A} \max_{j=1}^{i-1} S(a, l, j)] \quad (\text{F.1})$$

rather than just the expected score $E[S(a, l, i)]$. (Consider $S(a, l, j)$ to be 0 if agent a was not selected in round j to be run on problem instance l .) Although the improved greedy scheme is suboptimal even for unit runtimes, in practice it does well (Section 4).

4 Experiments: Known IID

Quality/runtime tradeoffs for the independent model were examined for score and runtime distributions based on runs of AIBirds algorithms on original Angry Birds game levels. Each algorithm was run 10 times on each level to obtain the empirical distributions, which were then treated as if they were the true distributions. Levels that caused issues with the agent’s vision module were filtered out.

We applied the meta-agent to the open source versions of the following five existing agents: Naive, AngryBER, ihsev, Eagle’s Wing and planA, which competed in past AIBirds competitions. Note that these versions are not necessarily identical to the versions submitted for the competition. All tests were conducted on Windows 10 using a machine with Intel(R) i7-4700HQ 2.40GHz processor and 12 GB RAM. The evaluation process was performed using different numbers of levels and time budgets as described in Algorithm 1.

We also evaluated some of the algorithms in AIBirds competition settings: 8 levels with $T = 1800$ seconds.

The following optimization algorithms were compared:

1. **Dynamic Programming (optimal)**: compute the recurrence relation in Figure F.1, using memoization of the results from all recursive calls.
2. **Binned Dynamic Programming(X, Y)**: same as the optimal solution, with scores rounded up to the next multiple of X , and times rounded to the next multiple

4. Experiments: Known IID

Algorithm 1: Evaluation process

```

1 for 1 to 50 do
2   for Every number of levels and time budget  $T$  configuration in
      $\{2, 3, 4\} \times \{200, 400, 600, 800, 1000\}$  do
3     Draw random levels uniformly from the level pool.
4     for 1 to 10,000 do
5       while Time budget was not exceeded do
6         Compute policy and choose a move using collected statistics as
           true distribution.
7         Execute the selected move (agent and level) by drawing score
           and time according to the statistics.

```

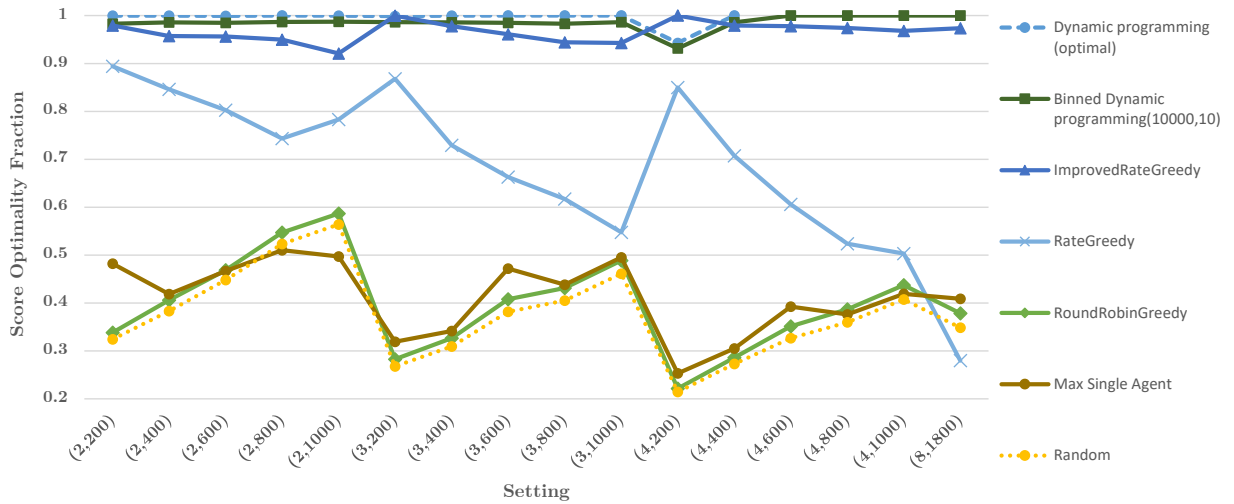


Figure F.2: Optimization Algorithms Score Evaluation

of Y . We used this algorithmic scheme with $X \in \{1, 1000, 10000\}$ and $Y \in \{1, 10, 25\}$.

3. **Score Greedy**: choose the agent and level that maximize the expected score.
4. **Rate Greedy**: choose the agent and level that maximize the expected score divided by the expected time given that the time is less than or equal to the remaining time, multiplied by the probability that the time is less or equal to the remaining time.
5. **Improved Score/Rate Greedy**: same as the score/rate greedy except for considering expected score/rate improvement instead of expected score.
6. **Round Robin Score Greedy**: the algorithm used by the hyper-agent from (Stephenson and Renz 2017), which selects a level using round-robin, and chooses the agent that maximizes the expected score for that level, preferring agents not selected in previous attempts.
7. **Play Single Agent(A)**: select a level using round-robin, always with agent A . This scheme was evaluated for each of the 4 possible agents.
8. **Random**: draw a pair of level and agent uniformly.

The results appear in Figure F.2. The scores in the plot are normalized to the highest score for each setting. For clarity, we show only a subset of the algorithms. In Play Single Agent policies, we show only the maximum value among them. The score greedy and improved score greedy were dominated by the rate greedy and improved rate greedy respectively, and are not shown. Finally, we showed the Binned Dynamic Programming(10000,10) as a sole representative of its category, since it achieved the best balance between runtime and score. The optimal policy did not always result in the best score, as the process is stochastic and thus exhibits measurement noise. The results indicate that the optimal policy is indeed the best in terms of scores. However, the binned version and the improved rate greedy achieved near-optimal results. When considering runtime and space usage, the optimal solution could not solve instances greater than 4 levels with $T = 400$ time budget. The binned version was able to solve all instances, with a maximal overhead of 13.7 seconds and a maximal memory

usage of 52 MB across all instances with 4 levels or less. However, it required an average of 335 seconds and 270 MB of memory to handle the full competition setting (8 levels with 1800 seconds time budget). All the other algorithms ran in negligible time (several milliseconds) and memory. This makes the improved rate greedy the best algorithm in terms of balance between score and resources. The large gap in scores between schemes that selected instances either randomly or in round-robin fashion and those that attempted to optimize instance selection (improved greedy and dynamic programming) suggest that the multi-instance setting is very different from the single instance setting, and that a good instance selection scheme is crucial. The apparent increase in the gap as the number of instances grows further supports this observation (Figure F.2). Note that in the experiments we did not include the optimization runtime in the total available runtime T ; but in a competition it must be. Also, the simulation runtime of the agents averaged roughly 90 seconds, so 1800 seconds consists of about 20 rounds (agent runs).

5 Unknown Distributions

A major point of competitions like Albirds (as well as other competitions, such as IPC, SAT-solving, etc.) is that they are done with *previously unseen* problem instances, so the score and runtime distributions are unknown. The latter issue then becomes a learning problem, which can be modeled by treating the agent performance quality as hidden random variables, with some assumed prior distributions based on observing similar problem instances. We adopt a naive learning scheme (described briefly below for independent scores and runtimes for simplicity, as implemented for the Albirds meta-agent).

5.1 Unknown IID Score and Runtime

In our naive learning scheme, we are assuming that agent performance profiles of a previously unseen problem instance are (almost) equal to their performance profile on some problem instance(s) for which performance statistics were already collected. Hence, we are essentially taking a case-based reasoning (CBR) approach to predicting the performance profile for an unknown instance. However, we do not assume knowl-

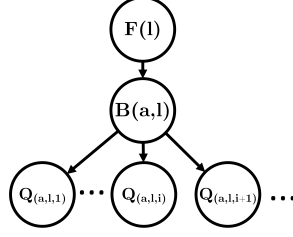


Figure F.3: Dependency model (Bayes net fragment)

edge of *which* of the previously seen performance statistics fits the current instance.

Therefore, if we need to predict an agent performance $Q(a, l, i)$ for round i of the current problem instance l (which we model as the score and runtime iid $P_S(a, l)$ and $P_T(a, l)$) it makes sense to condition on the (unknown) agent performance profile $Q(a, l)$ for the current problem instance. Essentially, what we need is a mapping from features to agent performance (i.e. *distributions* over score and runtime distributions), as we do in this paper.

The mapping we adopt here is simply a smoothed version of the performance profiles of the K most similar instances. (Smoothing is done in by assigning scores and runtimes into bins.) That is, for every problem instance l in the training set, and every agent program a , collect and store the score and runtime distributions estimate as $Q(a, l)$ indexed by the feature values (vector $F(l)$) for problem instance l .

When a new problem instance l is encountered online, compute its feature vector values $F(l)$, and find the K most similar instances l_1, \dots, l_K according to some appropriately defined similarity measure $s(l_i, l)$. Now we assume that the agent performance for instance l has a distribution over performance profiles, and that it is equal to the (smoothed version) of its performance profile for some instance l_i , with probability proportional to $s(l_i, l)$. That is, denote by $B(a, l)$ a K -valued random variable, with integer values denoting the respective $Q(a, l_i)$ profile. Then we have:

$$P(B(a, l) = i) = \frac{s(l_i, l)}{\sum_{j=1}^K s(l_j, l)}.$$

The performance profiles describe both score and runtime distributions, and additionally we assumed that these are drawn i.i.d. given the value of $B(a, l)$. The distribution model topology is summarized in Figure F.3, for each problem instance l (shown for one agent program). We have an observable feature vector variable $F(l)$. Belief updating for this conditionally i.i.d. model is straightforward, as this is a naive Bayes model.

5. Unknown Distributions

Algorithm	Number Of Levels															AVG
	2					3					4					
	200	400	600	800	1000	200	400	600	800	1000	200	400	600	800	1000	
IRG	0.71	0.72	0.70	0.75	0.78	0.66	0.71	0.63	0.67	0.69	0.55	0.63	0.62	0.63	0.67	0.67
BIRG	0.69	0.76	0.81	0.84	0.85	0.67	0.75	0.73	0.76	0.83	0.54	0.68	0.77	0.75	0.79	0.75
IRG (1)	0.60	0.60	0.64	0.63	0.59	0.57	0.54	0.52	0.52	0.54	0.54	0.53	0.46	0.44	0.47	0.55
IRG (50)	0.70	0.73	0.74	0.75	0.78	0.72	0.73	0.65	0.70	0.72	0.57	0.69	0.64	0.70	0.71	0.70
IRG (100)	0.70	0.70	0.71	0.75	0.76	0.68	0.69	0.64	0.67	0.70	0.59	0.65	0.62	0.63	0.68	0.68
Random	0.38	0.50	0.56	0.64	0.68	0.31	0.40	0.45	0.52	0.59	0.24	0.34	0.36	0.47	0.50	0.46
RRG	0.45	0.56	0.63	0.71	0.75	0.32	0.48	0.52	0.60	0.66	0.28	0.40	0.42	0.55	0.57	0.53
MSA	0.53(p)	0.80(e)	0.66(p)	0.72(i)	0.75(i)	0.48(p)	0.57(i)	0.60(p)	0.66(p)	0.65(i)	0.38(e)	0.60(p)	0.61(p)	0.64(p)	0.63(i)	0.62

Table F.1: Solution qualities as a fraction of the solution quality obtained by the "omniscient" improved rate greedy scheme

In the conditionally i.i.d. model, since $B(a, l)$ is unobservable, but its current distribution (thus belief state) changes given new observations of $T(a, l, i)$ and $S(a, l, i)$, we now have a POMDP that we cannot hope solve optimally, especially in real time. Instead, we can solve an MDP where the $T(a, l, i)$ and $S(a, l, i)$ are assumed to be i.i.d. as before, but based on the current belief state of $B(a, l)$.

That is, we can do the belief updating given the new observed scores and runtimes, but in the policy computation act as if future updates are not observed. Then one can re-compute the MDP policy after each observation and belief update. However, the MDP solution was also quite computationally intensive, and re-computation makes it even more so. As the improved greedy scheme performed almost as well as the MDP solution w.r.t. optimality, we no longer considered using the MDP solution, for practical reasons.

Obtaining a reasonable $s(l, l')$ is a learning problem, which was tackled by normalizing all feature values to $[0, 1]$, and taking the inverse of the Euclidean distance as the similarity. Despite the naive nature of our method for defining prior probability of unobserved levels, our algorithms, provided with such priors, showed a major improvement over existing methods according to the results presented below.

5.2 Experiments: Unknown IID

We normalized each level's score by $maxScore_l$, an upper bound on achievable score in each level, that can be computed using the features. We used the following subset of features, described in (Stephenson and Renz 2017; Tziortziotis et al. 2016): #Blocks, targetWidth, targetHeight, closestObjDist, farthestObjDist, density, #Objects, iceOb-

jects, woodObjects, stoneObjects, #Pigs, helmetPigs, noHelmetPigs, #Birds, #RedBirds, #YellowBirds, #BlueBirds, #BlackBirds, #WhiteBirds, varietyOfBirds, feasibleObjects, feasiblePigs, roundObjectsNotPigs, icedTerritory, woodenTerritory, stonedTerritory, averagePigsInBlocks, blocksWithPigs and #TNTs.

We tested the optimization algorithms using the same process as in Section 4, where the algorithms had to rely on the predicted distribution based on the naive learning scheme. We incorporated the resulting distribution in the following algorithms: (1) the improved rate greedy defined in Section 4, using the distribution of distributions with K neighbors (denoted by IRG(K)) without belief updating; (2) a binned version of the improved rate greedy with Bayesian belief updating, using a zero value bin and 10 additional bins uniform in $(i \cdot \maxScore_l, i + 0.1 \cdot \maxScore_l]$, $0 \leq i \leq 9$ (denoted as BIRG). We used $K = 128$ in the learning process for both algorithms as a default value. We also tested IRG with other K values, specified in parenthesis.

Table F.1 shows the solution quality achieved by the different algorithms, relative to the quality achieved by an improved rate greedy algorithm acting on *known* distributions (denoted "omniscient"). The projected standard deviation σ' of the results was at most ± 0.013 , small enough to maintain the performance ordering between the algorithms as presented below.²

In most cases, IRG (using *unknown* distributions) shows a major improvement over the baseline methods: choosing agent and level at random; choosing the post-facto best performing agent (denoted MSA, with the first letter of agent achieved that score in parenthesis); and the round robin greedy algorithm (denoted as RRG) with *known* expectation. BIRG (using *unknown* distributions), further improved the results, achieving an average solution quality of 0.75 despite using a naive learning scheme. Note that when BIRG had sufficient time to perform updates (600 seconds and above) it achieved the best score out of all tested algorithms. We also tracked the improvement due to Bayesian updates by comparing the Wasserstein distance (also known as "earth mover's distance" (EMD)) between the predicted and true distributions. As expected, results improved as the above EMD decreased, which also explains the improved results for BIRG in the longer sequences. With Bayesian updating, the EMD

²The projection σ' was based on the standard deviation σ measured over averages of sets of 35 random runs. To project to the 1000 runs per instance actually used we have $\sigma' = \frac{\sigma}{\sqrt{\frac{1000}{35}}}$.

has improved from the beginning to the end of each test setting, with an average improvement of 17.2% (not shown).

In IRG performance seems to improve with increased K , up to a point where this trend peters off and even reverses. We believe that including too few cases is insufficient to predict the performance profile well, but too many cases leads to overfitting. This phenomenon does not occur in BIRG due to the Bayesian updating which quickly disregards irrelevant cases, and improves monotonically with K .

5.3 Experiments: Angry Birds Game

After experimenting using data collected from the game, we implemented a full version of the meta-agent, which interfaces with the AIBirds server and plays the actual game. Our meta-agent implementation starts by collecting information on all levels using the provided vision module. Based on this, the meta-agent constructs an objects-tree for each level, extracts features from the objects-trees, and predicts a performance profile for each level and agent pair. Then, the meta-agent applies the BIRG scheme to select a pair of agent and a level to play. The meta-agent sends the selection to the server and observes the results of the run. The observations are used for belief updating. The select-and-play process repeats until the time limit is reached.

Our evaluation was based on past competition levels (between 2014 and 2016), a total of 72 levels (8 at a time with a 30 minutes time budget). The results are shown in table F.2. The improved-greedy based meta-agent achieved an average score of 441,752, compared to PlanA (357,468), ihsev (321,280), AngryBER (303,166) and Eagle’s Wing (323,099). Note that the above 4 agents were the ones actually used by the meta-agent, and all of them contributed to its score. The hyper-agent of (Stephenson and Renz 2017) achieved an impressive average score of 424,740; However, the authors, which are the organizers of the AI-Birds competition, had access to a total of 8 agents as opposed to our 4 open-sourced agents. We strongly believe that using the full set of 8 agents would have further improved the performance of our algorithm, as in auxiliary experiments (not shown) our scheme was relatively robust to adding agents (including dummy, useless agents). This belief is further supported by Stephenson and Renz 2017 which showed that each individual agent actually contributed to the score of the hyper-agent, meaning that the agents to which we had no

Round	Naive	PlanA	ihsev	A-BER	E-Wing	Hyper	Meta
Q 2014	187,180	314,540	109,920	188,710	282,110	332,270	418,210
S 2014	400,980	541,220	439,520	429,680	442,800	524,400	602,050
F 2014	209,130	193,110	257,410	90,110	250,970	338,330	231,480
Q 2015	68,020	316,850	163,790	367,000	346,760	351,300	372,260
S 2015	145,910	288,870	166,750	143,270	299,220	375,670	369,130
F 2015	131,660	452,860	458,030	392,420	191,970	483,610	490,050
Q 2016	251,080	313,440	444,560	310,600	252,100	336,840	426,570
S 2016	436,870	372,330	562,820	445,030	420,170	610,280	593,020
F 2016	390,050	423,990	288,720	361,670	421,790	469,960	471,380
Average	246,764	357,468	321,280	303,166	323,099	424,740	441,572

Table F.2: Actual game results using past competitions setting

access were actually *not* useless.

6 Discussion

In this paper we defined the MaxScore optimization problem, analyzed its computational complexity (NP-hard even under extreme restrictions), and suggested approximation algorithms for known independent distributions. In practice, based on empirical evaluation on AI birds, it turns out that a greedy algorithm based on expected improvement is near-optimal. Despite the latter having no theoretical guarantees, it currently seems to be the only viable alternative for real-time computation. Applying these results to unknown distributions requires learning performance profiles given problem instance features. A naive learning scheme applicable to the AIbirds application was proposed. This results in imperfect predicted distributions, which degrades the meta-reasoning results. Nevertheless, the greedy algorithm is still the better option, especially if the distribution model is updated using scores and run-times observed during the run.

The MaxScore problem is closely related to *algorithm selection*, as originally defined by Rice in 1976 (Rice 1976). *Algorithm portfolios* (Gomes and Selman 2001; Huberman

et al. 1997) are a natural and popular extension of the idea of algorithm selection. Such techniques are based on minimizing risk in economics. This approach defines a collection of algorithms (a portfolio) and establish a resource allocation to the algorithms in the portfolio in order to solve a given problem instance (instead of choosing a single algorithm for a given problem instance). This field has been studied extensively in the last decades, including works on different computational settings (parallel, sequential or in-between), many applications with outstanding results (Xu et al. 2008; Hoos et al. 2014; Kadioglu, Malitsky, Sellmann, et al. 2010; Kadioglu, Malitsky, Sabharwal, et al. 2011) and even meta-level techniques for choosing a selector (Lindauer et al. 2015). Most common settings of algorithm portfolios focus on finding a solution to a single given problem instance. Our setting generalizes the meta-level decision problem solved in algorithm portfolios to choosing which problem instance to work on, as well as selecting algorithms to use at any given time. A paper on dynamic restart policies (Kautz et al. 2002) proposes an optimal restart scheme in a decision-theoretic sense, similar to that defined in our paper, and with a scheme for learning a runtime distribution. Since our setting allows non-binary scores, where it is important to get a good score on a problem instance, rather than just solve it, the optimization scheme used in the restart policies paper is not directly applicable here. The scheme they use to learn runtime distributions may be applicable to our setting, but must be extended to predict score distributions as well before it can be used here. Maximizing the *number* of instances solved is also mentioned in (Kautz et al. 2002), but their instances are drawn randomly and independently, so there seems to be no allowance for the capability of choosing to return to a previously run instance as in our setting, in addition to there being no notion of instance score in (Kautz et al. 2002).

The MaxScore problem is also loosely related to multi-armed bandit (MAB) problems (Auer et al. 2002). Much of the related work on MABs does not assume a known distribution, or even a distribution over distributions as done in this paper. Rather, bounds on regret are analyzed, both asymptotic and finite. However, the fact that the reward in MaxScore is the maximum rather than the sum makes it unclear how such techniques might carry over. Additionally, in the motivating application of Albirds, the number of rounds is small, further complicating such attempts. In fact, if we tried to apply an MAB scheme directly, we would get a random selection of problem in-

stance, against which we did compare in the Angry Birds domain (random did poorly, as expected).

A significant part of the research on algorithm portfolios and multi-armed bandits focuses on learning issues. E.g. in (Kotthoff 2016), the focus is on analyzing problem features and applying different varieties of machine learning techniques in order to find scheduling policies for the portfolios. In this paper we achieved good results despite using a rather naive learning scheme to obtain a mapping from features to score and runtime distributions. Note that the relative performances in Table 1 still leave much room for improvement by better predicting the distributions: these performance figures are still well below the 1.0 value obtained by the an "omniscient" rate-greedy scheme that has access to the true distributions. Introducing better learning schemes for better prediction of the distributions should thus result in better performance.

Another issue for future work is learning the distribution models with time-score and inter-round dependencies, thus extending MaxScore solutions to more general settings of algorithm portfolios over optimization problems. Fully testing such generalized scenarios would require changing the rules of the competitions to maximizing total score over a global time limit, rather than the current setting where the time limits are per-instance.

Acknowledgements

Supported by ISF grant 417/13 and BGU Frankel Center. Meta-agent implemented by: Lior Schachter, Dor Bareket, Ori Zviran.

Appendix: Proof of Theorem F.1

Theorem 1. *The linear setting of the MaxScore problem with independent score distributions, deterministic runtimes, and $|I| = 1$, is NP-hard.*

Proof: by reduction from the optimization version of knapsack ((Garey and Johnson 1979), problem number [MP9]), re-stated below. Given a set of items $\mathcal{S} = \{s_1, \dots, s_n\}$, each with a positive integer weight w_i and a positive integer value v_i , a weight limit W , find a sub-multiset S of \mathcal{S} with a maximal total value, subject to: total

6. Discussion

weight of S at most W .

In the reduction, each agent represents an item in the Knapsack problem, where $P_T(a_i, l) = [1 : w_i]$ and $P_S(a_i, l) = [\varepsilon : v_i, 1 - \varepsilon : 0]$. As this is a simple one-to-one mapping, we abuse the notation and treat the agents as if they are actually the respective elements from \mathcal{S} in the Knapsack problem. In the MaxScore problem, let:

$$T = W, \quad H = \max_{s_i \in \mathcal{S}} v_i, \quad M = \frac{W}{\min_{s_i \in \mathcal{S}} w_i}, \quad \varepsilon = \frac{1}{M^2 H + 1}$$

Let S be a candidate solution to the MaxScore problem, with $m = |S| \leq n$. Assume w.l.o.g. that $S = \{s_1, \dots, s_m\}$ and that the items are sorted in non-descending order of values v_i . Denote by $P(S)$ expected value from selecting the items in the sequence S as a policy. Then:

$$P(S) = \sum_{i=1}^m v_i \varepsilon (1 - \varepsilon)^{m-i} \leq \sum_{i=1}^m v_i \varepsilon$$

On the other hand, we have:

$$\begin{aligned} P(S) &= \sum_{i=1}^m v_i \varepsilon (1 - \varepsilon)^{m-i} \geq \sum_{i=1}^m v_i \varepsilon (1 - \varepsilon)^m \\ &\geq \sum_{i=1}^m v_i \varepsilon (1 - \varepsilon)^M \end{aligned}$$

From Bernoulli's inequality, we have:

$$(1 - \varepsilon)^M \geq 1 - M\varepsilon = 1 - \frac{M}{M^2 H + 1} > 1 - \frac{1}{MH}$$

Therefore:

$$\begin{aligned} P(S) &> \sum_{i=1}^m v_i \varepsilon \left(1 - \frac{1}{MH}\right) = \sum_{i=1}^m v_i \varepsilon - \frac{\sum_{i=1}^m v_i \varepsilon}{MH} \\ &\geq \sum_{i=1}^m v_i \varepsilon - \varepsilon = \varepsilon \left(\sum_{i=1}^m v_i - 1\right) \end{aligned}$$

Now let S be an *optimal* solution to the MaxScore problem. Since S satisfies the time constraint, we have $\sum_{i=1}^m w_i \leq T = W$, so S satisfies the weight constraint in the Knapsack problem and is thus a solution therein. Assume in contradiction that there exists a legal solution S' to Knapsack s.t. $\sum_{s_i \in S'} v_i > \sum_{s_i \in S} v_i$. Since the values of the items in knapsack are integers, we know that $\sum_{s_i \in S'} v_i \geq (\sum_{s_i \in S} v_i) + 1$. Thus, as $|S'| \leq \frac{1}{\varepsilon}$:

$$P(S') > \varepsilon \left(\sum_{s_i \in S'} v_i - 1\right) \geq \varepsilon \left(\sum_{s_i \in S} v_i\right) \geq P(S)$$

As S' satisfies the timing budget in the MaxScore problem, it is a solution better than S , a contradiction. So S is also an optimal solution to the Knapsack problem. \square

References of Paper F

- [ACF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. “Finite-time Analysis of the Multiarmed Bandit Problem”. In: *Mach. Learn.* 47 (2-3 May 2002), pp. 235–256. ISSN: 0885-6125.
- [GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [GS01] Carla P. Gomes and Bart Selman. “Algorithm portfolios”. In: *Artif. Intell.* 126.1-2 (2001), pp. 43–62. DOI: 10.1016/S0004-3702(00)00081-3. URL: [https://doi.org/10.1016/S0004-3702\(00\)00081-3](https://doi.org/10.1016/S0004-3702(00)00081-3).
- [HLH97] Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. “An Economics Approach to Hard Computational Problems”. In: *Science* 275.5296 (1997), pp. 51–54.
- [HLS14] Holger Hoos, Marius Thomas Lindauer, and Torsten Schaub. “claspfolio 2: Advances in Algorithm Selection for Answer Set Programming”. In: *TPLP* 14.4-5 (2014), pp. 569–585.
- [Kad+10] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, et al. “ISAC –Instance-Specific Algorithm Configuration”. In: *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2010, pp. 751–756.
- [Kad+11] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, et al. “Algorithm Selection and Scheduling”. In: *CP (LNCS6876)*. Jan. 2011, pp. 454–469.
- [Kau+02] Henry A. Kautz et al. “Dynamic Restart Policies”. In: *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*. Ed. by Rina Dechter, Michael J. Kearns, and Richard S. Sutton. AAAI Press / The MIT Press, 2002, pp. 674–681. URL: <http://www.aaai.org/Library/AAAI/2002/aaai02-101.php>.

- [Kot16] Lars Kotthoff. “Algorithm Selection for Combinatorial Search Problems: A Survey”. In: *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*. Ed. by Christian Bessiere et al. Vol. 10101. Lecture Notes in Computer Science. Springer, 2016, pp. 149–190.
- [Lin+15] Marius Thomas Lindauer et al. “AutoFolio: An Automatically Configured Algorithm Selector”. In: *J. Artif. Intell. Res.* 53 (2015), pp. 745–778.
- [Ric76] John R. Rice. “The Algorithm Selection Problem”. In: *Advances in Computers* 15 (1976). Ed. by Morris Rubinoff and Marshall C. Yovits, pp. 65–118.
- [SR17] Matthew Stephenson and Jochen Renz. “Creating a Hyper-Agent for Solving Angry Birds Levels”. In: *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17), October 5-9, 2017, Snowbird, Little Cottonwood Canyon, Utah, USA*. Ed. by Brian Magerko and Jonathan P. Rowe. AAAI Press, 2017, pp. 234–240.
- [SS17] Shahaf S. Shperberg and Solomon Eyal Shimony. “Some Properties of Batch Value of Information in the Selection Problem”. In: *J. Artif. Intell. Res.* 58 (2017), pp. 777–796. DOI: 10.1613/jair.5288. URL: <https://doi.org/10.1613/jair.5288>.
- [TPB16] Nikolaos Tziortziotis, Georgios Papagiannis, and Konstantinos Blekas. “A Bayesian Ensemble Regression Framework on the Angry Birds Game”. In: *IEEE Trans. Comput. Intellig. and AI in Games* 8.2 (2016), pp. 104–115.
- [Xu+08] Lin Xu et al. “SATzilla: Portfolio-based Algorithm Selection for SAT”. In: *J. Artif. Intell. Res.* 32 (2008), pp. 565–606.
- [ZR96] Shlomo Zilberstein and Stuart J. Russell. “Optimal Composition of Real-Time Systems”. In: *Artif. Intell.* 82.1-2 (1996), pp. 181–213.

Paper G

Enriching Non-parametric Bidirectional Search Algorithms

Shahaf S. Shperberg, Ariel Felner, Nathan R. Sturtevant, Solomon Eyal
Shimony, Avi Hayoun

The paper has been published in the
AAAI '19: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, 2019
pp. 2379–2386

© 2019 AAI

The layout has been revised.

Abstract

NBS is a non-parametric bidirectional search algorithm proven to expand at most twice the number of node expansions required to verify the optimality of a solution. We introduce new variants of NBS that are aimed at finding all optimal solutions. We then introduce an algorithmic framework that includes NBS as a special case. Finally, we introduce DVCBS, a new algorithm in this framework that aims to further reduce the number of expansions. Unlike NBS, DVCBS does not have any worst-case bound guarantees, but in practice it outperforms NBS in verifying the optimality of solutions.

1 Introduction and Overview

Given a graph G , the shortest-path problem is to find the least-cost path from state s to state g in G . Bidirectional heuristic search algorithms (denoted henceforth by Bi-HS) interleave two separate searches, a search forward from s and a search backward from g . Recent research (Eckerle et al. 2017) defined conditions on the node expansions required by Bi-HS algorithms to guarantee solutions optimality. Following work reformulated these conditions as a *must-expand graph* (G_{MX}), showing that the *Minimum Vertex Cover* (MVC) of G_{MX} corresponds to the minimal number of expansions (Chen et al. 2017) required to prove optimality. Finally, Shaham et al. (2017; 2018) studied the G_{MX} structure and its extension, $G_{MX\epsilon}$, that exploits knowledge of the minimal edge cost (ϵ), to characterize properties of the MVC.

Bi-HS algorithms can be classified as *parametric* or as *non-parametric*. Two parametric algorithms were recently developed. *Fractional MM* ($fMM(p)$) (Shaham, Felner, Chen, et al. 2017) generalizes the MM algorithm (Holte et al. 2017) by controlling the fraction p of the optimal path at which the forward and backward frontiers meet. There exists an optimal fraction p^* for which $fMM(p^*)$ will expand exactly an MVC of G_{MX} , but p^* is not known a priori. Another parametric algorithm is GBFHS (Barley et al. 2018), which iteratively increases the depth of the search. It is parametric in a pre-defined *split function* that determines how deep to search on each side at each iteration. GBFHS with an optimal split function also converges to an MVC of G_{MX} . However, such a split function is not known a priori. Without knowledge of the optimal

parameter values, both algorithms may expand many more nodes than an MVC of G_{MX} .

In this paper we focus on non-parametric Bi-HS algorithms. NBS (Chen et al. 2017) is a robust state-of-the-art non-parametric algorithm that computes a *vertex cover* (VC) of G_{MX} whose size is at most $2|MVC|$. We enrich this line of research and introduce new settings and new algorithms that aim to find a VC of G_{MX} . In particular, we make the following contributions:

- (1) We describe and motivate the problem of finding *all* optimal solutions, and introduce two new versions of G_{MX} (with/without ϵ) that are suited for such settings. This results in four different problem settings, each with its own G_{MX} .
- (2) We introduce a 2-level framework for non-parametric Bi-HS algorithms and reformulate NBS as a special case.
- (3) Utilizing our framework, we adapt NBS to the four settings, while maintaining the $2|MVC|$ guarantee.
- (4) We introduce a new algorithm *Dynamic Vertex Cover Bidirectional Search* (DVCBS). It uses the same high-level framework we developed, but unlike NBS, always tries to expand a VC of a dynamic G_{MX} graph which is also introduced. Here too, four versions are possible.
- (5) Our experimental results show that the new variants of NBS, as well as DVCBS, outperform previous variants of NBS for finding both the *first* and *all* optimal solutions, expanding significantly fewer nodes in many cases.

2 Definitions and Background

Let $d(x, y)$ denote the shortest distance between x and y , $C^* = d(s, g)$, and let f_F , g_F and h_F indicate f -, g -, and h -costs in the forward search, and likewise f_B , g_B and h_B in the backward search. The *forward heuristic* h_F is *admissible* iff $h_F(u) \leq d(u, g)$ for every state $u \in G$ and is *consistent* iff $h_F(u) \leq d(u, u') + h_F(u')$ for all $u, u' \in G$. The *backward heuristic* h_B is defined analogously. *Front-to-end* Bi-HS algorithms use these two heuristic functions and in this paper we assume that both are admissible and consistent. *Front-to-front* Bi-HS algorithms use heuristics between pairs of states on opposite frontiers, and are outside the focus of this paper; see Holte et al. 2017 for a survey.

2.1 Guaranteeing Solution Optimality

Unidirectional search algorithms must expand all nodes n with $f(n) < C^*$ in order to guarantee the optimality of solutions (Dechter and Pearl 1985).

Eckerle et al. 2017 generalized this to Bi-HS by examining pairs of nodes $\langle u, v \rangle$ such that u is in the forward frontier and v is in the backward frontier. They defined conditions for when such pairs should be expanded:

1. $f_F(u) < C^*$
2. $f_B(v) < C^*$
3. $g_F(u) + g_B(v) < C^*$

If u and v meet the three conditions, then to guarantee solution optimality every algorithm must expand at least one of u or v in order to ensure that there is no path from s to g passing through u and v of cost $< C^*$.

Definition G.1

For each pair of states (u, v) let $lb(u, v) = \max\{f_F(u), f_B(v), g_F(u) + g_B(v)\}$

In Bi-HS, a pair of states $\langle u, v \rangle$ is called a *must-expand pair* (MEP) if $lb(u, v) < C^*$. The MEP definition is equivalent to the above conditions; for each MEP only *one* of u or v *must* be expanded. In the special case of unidirectional search, algorithms expand all the nodes with $f_F < C^*$, which is equivalent to expanding the forward node of every MEP. Bi-HS algorithms may expand nodes from either side, potentially covering all the MEPs with fewer expansions.

Shaham, Felner, Sturtevant, et al. 2018 generalized the three conditions to handle the case where a lower bound ϵ on the edge costs is available. In unit edge-cost domains $\epsilon = 1$, while in other domains one might iterate over all action costs and set ϵ to their minimum. We denote this case by ϵ -case, as opposed to the base-case, where no knowledge of ϵ is available. For ϵ -case, Condition 3 is changed to:

3. $g_F(u) + g_B(v) + \epsilon < C^*$

Consequently, the lower bound is changed to:

$$lb(u, v) = \max\{f_F(u), f_B(v), g_F(u) + g_B(v) + \epsilon\}$$

and an MEP is defined according to the new lb .

2.2 The Must-Expand Graph (G_{MX})

The problem of selecting the minimal set of nodes that cover all MEPs can be restated as finding an MVC on the *must-expand graph* (Chen et al. 2017).

Definition G.2

The Must-Expand Graph (G_{MX}) of a problem instance is an undirected, unweighted bipartite graph. For each state $u \in G$ there is a left vertex u_F and a right vertex u_B . G_{MX} has an edge between a left vertex u_F and a right vertex v_B if and only if (u, v) is an MEP.

It follows that Bi-HS algorithms must expand a vertex cover (VC) of the induced G_{MX} when solving a problem instance. The MVC is thus a lower bound on the number of expansions. Another version of G_{MX} , denoted by G_{MX_ϵ} , can be constructed for ϵ -case (Shaham, Felner, Sturtevant, et al. 2018).

Figure G.1 illustrates different versions of G_{MX} for the problem instance in Figure G.1(a), in which $C^* = 3$. Figure G.1(b) shows the corresponding G_{MX} . The left (right) vertices are ordered by increasing (decreasing) g_F -costs (g_B -costs). Additionally, vertices with identical g_F (or g_B) are merged into a single *weighted vertex*, denoted as a *cluster*. For example the *cluster* with $g_F = 1$ includes both A and X and its weight is 2. Similarly, an edge that connects clusters represents all possible edges between them (the product of their weights), e.g., 6 edges connect the cluster with $g_F = 1$ to the one with $g_B = 1$. Figure G.1(c) shows G_{MX_ϵ} ($\epsilon = 1$). Due to the addition of ϵ , some edges that exist in G_{MX} no longer exist in G_{MX_ϵ} . For example, the left cluster (vertex) with $g_F = 1$ is connected to all right clusters with $g_B \leq 1$ in G_{MX} but is only connected to the right cluster with $g_B = 0$ in G_{MX_ϵ} .

2.3 The Minimum Vertex-Cover of G_{MX}

Shaham, Felner, Chen, et al. 2017 introduced $\text{CalculateWVC}()$ (see their Section 6.5), an algorithm for finding an MVC of a G_{MX} . This algorithm relies on the fact that all such MVCs are *contiguous* and *restrained* in both directions. That is, there exist thresholds $t_F, t_B \in \mathbb{R}$ such that $t_F + t_B = C^*$ ($t_F + t_B + \epsilon = C^*$ for ϵ -case) for which a vertex u in direction D is in the MVC if and only if $g_D(u) < t_D$.

2. Definitions and Background

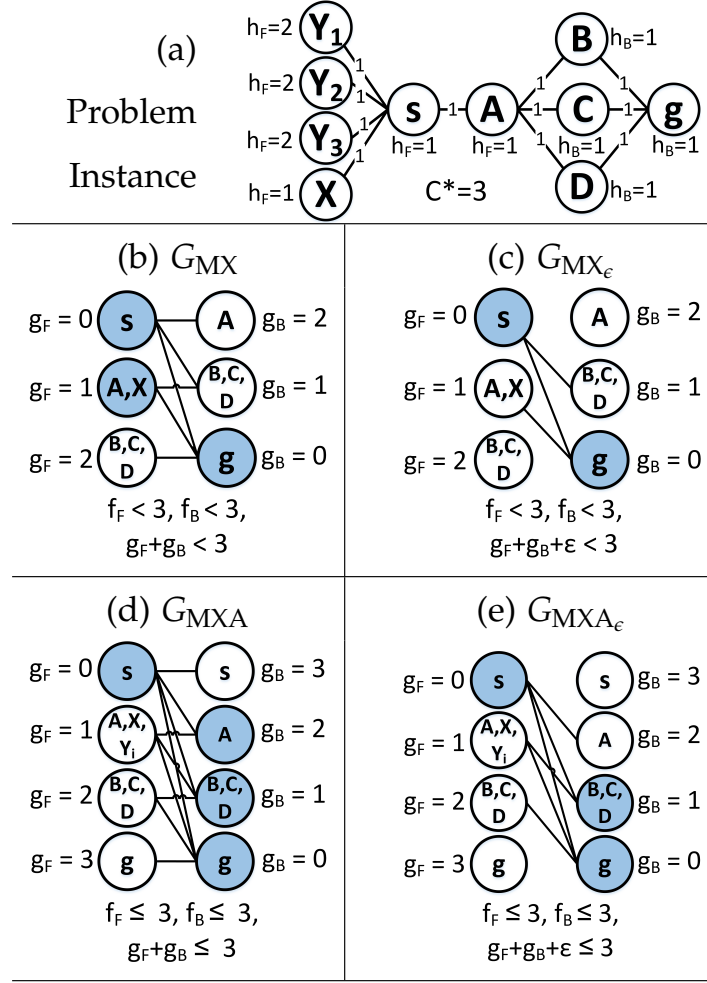


Figure G.1: Case Study: Different versions of G_{MX}

$\text{CalculateWVC}()$ iterates over all relevant pairs of values for which $t_F + t_B = C^*$ and finds the pair which induces the MVC. For example, in Figure G.1(b) the MVC (colored blue) is induced by $\langle t_F, t_B \rangle = \langle 2, 1 \rangle$ and includes only four nodes (s, A, X, g). $\text{CalculateWVC}()$ runs in time linear in number of clusters ($O(C^*)$) but assumes that G_{MX} and C^* are given as input. Thus, it can only run post-priori, after C^* was found and the entire G_{MX} was fully built (e.g., by running A* from both sides). Such information is not available to any Bi-HS algorithm during execution. Therefore, Bi-HS algorithms cannot guarantee that the VC they find is minimal. Hence, a main challenge of Bi-HS is to approximate an MVC by using only information available during the search.

3 Finding All Optimal Solutions

A common practice in the heuristic search literature is to halt the search once the *first* optimal solution is found and verified. This problem comprises two tasks: (1) finding a solution of cost C^* and (2) verifying that there are no solutions with cost $< C^*$. Most search algorithms interleave these tasks, completing them in an arbitrary order. The G_{MX} analysis above only handles the second task. Therefore, an MVC of G_{MX} may not capture the extra work needed to complete the first task of finding a solution (but $|MVC|$ is still a lower bound on the entire search). This is similar to only counting nodes with $f < C^*$ as necessary expansions in unidirectional search, and omitting nodes with $f = C^*$ that are expanded to find the goal (Dechter and Pearl 1985).

In many cases, *all optimal solutions* are required. For example, if not all the problem constraints can be encoded due to privacy issues, competing objectives, partial knowledge, etc. then an external decision maker is needed to choose a solution from the set of all optimal solutions (Byers and Waterman 1984; Arthur et al. 1997; Mahadevan and Schilling 2003). In other cases a solution may become invalid and an alternative solution needs to be obtained quickly (Siegmund et al. 2012; Isermann 1977). We denote these problem spaces by ϵ -ALL-case when knowledge of ϵ exists, and base-ALL-case otherwise.

Finding *all* optimal solutions only consists of a single compound task: verifying that there are no undiscovered solutions with cost $\leq C^*$ (as this includes the task of finding solutions with cost C^*). Thus, we can generalize the analysis in Section 2.2 to the case of finding *all* solutions in a way that allows us to bound the number of expansions required for the *entire* search. In addition, we show below that using this formalization also helps in finding a *first* solution faster.

3.1 G_{MX} for Finding All Optimal Solutions

The first step in generalizing the analysis for the task of finding all solutions is to re-define MEPs to use \leq instead of $<$ in the three conditions. Let u and v be nodes in the forward and backward frontiers, respectively. There can be an optimal path (of cost C^*) that goes from s to u to v to g , if:

1. $f_F(u) \leq C^*$

3. Finding All Optimal Solutions

2. $f_B(v) \leq C^*$
3. $g_F(u) + g_B(v) \leq C^*$

Likewise, $g_F(u) + g_B(v) + \epsilon \leq C^*$ is used in the ϵ -ALL-case. We define a pair of states (u, v) to be an MEP for the *all* cases (we call such pairs *must-expand-all* pairs, or MEAPs) if $lb(u, v) \leq C^*$, where $lb(u, v)$ is again the maximum of the three terms.

Theorem G.1

Let $I = \langle G(V, E), s, g \rangle$. A Bi-HS algorithm B will find all optimal paths in I if and only if B expands at least one state from every MEAP.¹

proof. If Case: Assume that B found all optimal paths but there is an MEAP $\langle u, v \rangle$ where neither u nor v were expanded by B . Consider the two paths: U from s to u with a cost of $g_F(u)$; and V from v to g with the cost of $g_B(v)$. Let $I' = \langle G'(V, E), h \rangle$ be a problem instance where $\langle u, v \rangle$ is an edge with cost ϵ . Therefore, there is a path $P = U \cdot V$ from s to g in G' . Since $\langle u, v \rangle$ is an MEAP, the cost of P is $g_F(u) + d(u, v) + g_B(v) = g_F(u) + g_B(v) + \epsilon \leq C^*$. However, $B(I') = B(I) \not\ni P$, contradicting the assumption that all optimal paths from s to g were found by B .

Only-If Case: Assume that B expanded at least one state from every MEAP, and there exists an optimal solution $P = \langle s = p_0, \dots, p_k = g \rangle$ that was not found. Since the heuristics are admissible, for all $0 \leq i \leq k$, $f_F(p_i) \leq C^*$, $f_B(p_i) \leq C^*$. Since P was not found, there exist nodes $p_i, p_j \in P$, $p_i \neq p_j$, in the forward frontier and backward frontiers of B respectively, when the search terminates. P is an optimal path, thus, $g_F(p_i) + g_B(p_j) + d(p_i, p_j) = C^*$. Since ϵ is a lower bound on the distance between nodes, $g_F(p_i) + g_B(p_j) + \epsilon \leq g_F(p_i) + g_B(p_j) + d(p_i, p_j) = C^*$. Hence $\langle p_i, p_j \rangle$ is an MEAP, contradicting the assumption that B expanded at least one state from every MEAP.

Note that the proof holds in base-ALL-case if $\epsilon = 0$.

We use the new *must-expand-all* conditions to define two new graphs: G_{MXA} for base-ALL-case, and G_{MXA_ϵ} for ϵ -ALL-case, in a manner similar to G_{MX} and G_{MX_ϵ} respectively, but with the \leq conditions. Importantly, $|MVC|$ of G_{MXA} and G_{MXA_ϵ} is a lower bound on the number of nodes that must be expanded to complete the joint task of finding *all* optimal solutions and verifying that there are no cheaper solutions.

¹We assume B is DXBB (See (Eckerle et al. 2017)). We also assume B maintains a frontier of all unexpanded discovered nodes, from which nodes are removed only upon expansion.

By contrast, $|MVC|$ of G_{MX} and G_{MX_e} only bounds the minimal number of expansions to complete the (second) task of verifying that no solution with cost $< C^*$ exists.

G_{MXA} and G_{MXA_e} for the example in Figure G.1(a) are shown in Figures G.1(d) and G.1(e), respectively. As can be seen, each vertex has more neighbors due to the use of \leq instead of $<$ in condition 3. For example, the cluster with $g_F = 1$ is now also connected to the cluster with $g_B = 2$. Furthermore, since conditions 1 and 2 now also have \leq , G_{MXA} contains additional clusters (e.g., with $g_F = 0$) and existing clusters may now be composed of additional states (e.g., y_i with $g_F = 1$ are included in G_{MXA} but not in G_{MX}).

Since G_{MXA} includes more edges than G_{MX} , the contiguous partition of their MVCs may be different, as demonstrated in Figure G.1. The MVC of G_{MX} (Figure G.1(b)) is composed of the vertices $\{s, A, X\}$ in the forward direction and $\{g\}$ in the backward direction. The MVC of G_{MXA} (Figure G.1(d)) is composed of vertex s in the forward direction and $\{g, D, C, B, A\}$ in the backward direction. Note that X is part of the MVC of G_{MX} but not a part of the MVC of G_{MXA} .

As a result, existing Bi-HS algorithms that consider G_{MX} when aiming to find a first solution should be modified to consider G_{MXA} when trying to find all optimal solutions. For example, the optimal fraction of $f_{MM}(p)$ for finding all solutions ($\frac{1}{4}$ for Figure G.1(a)) is different from the optimal fraction for finding a first solution ($\frac{2}{3}$). Furthermore, in section 4.2 we demonstrate that algorithms which consider G_{MXA} may be even better at finding the first solution.

4 A General Framework Encompassing NBS

Near-Optimal Bidirectional Search (NBS) (Chen et al. 2017) is a robust state-of-the-art non-parametric algorithm that is guaranteed to expand a VC of G_{MX} whose size is at most $2|MVC|$. In this section, we introduce a generalization of NBS: a two-level framework which we call the Lower-Bound-Framework (LBF). NBS is a specific implementation of the low level of LBF. We then introduce additional algorithms in this family which differ in their decisions at the low level of LBF.

LBF has two levels. The high level (Algorithm 1) maintains and dynamically increases a global lower bound (LB) on the cost of an optimal solution. It keeps track of

Algorithm 1: LBF high-level

```

1  $C \leftarrow \infty$ 
2  $LB \leftarrow \min\{h_F(s), h_B(g)\}$ 
3 while  $LB < C$  do
4    $C = \text{ExpandLevel}(LB, C)$ 
5   Increase  $LB$  to the next value
6 return  $C$ 

```

all states in the frontiers (OPEN lists) of the two directions of the search. For each node pair $\langle u, v \rangle$, $lb(u, v)$ is defined according to Definition H.2 above, depending of course, on the exact case (base-case, ϵ -case etc.). The global lower bound LB is set to be the minimal lb among all pairs.² The low level of LBF then needs to select valid nodes for expansion, i.e., nodes that may be part of paths of cost $\leq LB$. All the algorithms in the LBF family discussed in this paper use the same high level, but differ in the low-level selection policy.

4.1 The Low-Level Expansion Policy of NBS

The low-level policy of NBS is based on an approximate VC algorithm (Papadimitriou and Steiglitz 1982) which repeatedly chooses an edge and adds both its endpoints to the VC. Therefore, NBS repeatedly finds a pair $\langle u, v \rangle$ for which $lb(u, v) \leq LB$ and expands *both* u and v . The implementation details of NBS, as done by the original authors (outlined in Algorithm 2) are as follows. The frontier for each direction D is split into two separate queues: $waiting_D$ (sorted by f -value), which serves as a gateway to $ready_D$ (sorted by g -value). Nodes with a minimal f -value are moved from $waiting_D$ to $ready_D$, and only nodes from $ready_D$ are expanded. In the pseudo codes, every line which includes D is repeated twice, once for each direction. First (Lines 2–3), all nodes for which $f_D(u) < LB$ are moved to $ready_D$. Next (Lines 6–7), NBS selects a pair of nodes $u \in ready_F$ and $v \in ready_B$ for which $g_F(u) + g_B(v) \leq LB$, and expands *both* u and v . If no such pair is found, NBS repeatedly moves a pair of nodes for which

²Other Bi-HS algorithms also maintain and increase a global lower bound on the optimal solution, e.g., C in MM and $fLim$ in GBFHS. These bounds use less information than LB of LBF which directly depends on current knowledge on MEP as defined by the G_{MX} theory and therefore is tighter.

Algorithm 2: NBS Expand Level (LB, C)

```

1 while true do
2   while  $\min f \text{ in } \text{waiting}_D < LB$  do
3     move best node from  $\text{waiting}_D$  to  $\text{ready}_D$ 
4   if  $\text{ready}_D \cup \text{waiting}_D$  empty then
5     Terminate search - no solution was found
6   if  $\text{ready}_F.g + \text{ready}_B.g \leq LB$  then
7     ExpandD(C) node with min  $g_D$ -value in  $\text{ready}_D$ 
8   else
9     if  $\text{waiting}_D.f \leq LB$  then
10      move best node from  $\text{waiting}_D$  to  $\text{ready}_D$ 
11     else
12      return C

```

$f_F(u) \leq LB$ and $f_F(u) \leq LB$ from waiting_D into ready_D (Line 10) and continues to look for a pair for which $g_F(u) + g_B(v) \leq LB$. If such a pair is still not found, the low level reports back to the high level that no valid pairs were found, causing LB to be incremented.

Chen et al. 2017 proved three properties of NBS: (1) It is guaranteed to find an optimal solution. (2) It expands at most $2|MVC|$ states while finding a VC in G_{MX} . (3) No other Bi-HS algorithm can have better worst-case performance.

4.2 Finding All Optimal Solutions with NBS

The original low level used for NBS by Chen et al. 2017 is based on the properties of MEPs which use $< C^*$ in all three conditions. Therefore, NBS first considers nodes with f_F and f_B which are *strictly less* than LB (Line 2). Nodes with f_F and f_B that equal LB are only added *lazily* later (Lines 9–10 of Algorithm 2). We use NBS_F and $NBS_{F\epsilon}$ (F for *first* solution) to denote the original versions (Algorithm 2) for the base-case and ϵ -case, respectively.

In order to be better suited for finding *all* solutions we adapt the low-level

expansion policy of NBS to be based on MEAPs which have \leq in the three conditions. Specifically, we modify the NBS_F expansion policy to immediately consider all nodes for which $f_D(u) \leq LB$ by changing the $<$ condition in Line 2 of Algorithm 2 to be \leq . This change also eliminates Lines 9–11, as such nodes are handled *eagerly* in Line 2. We use NBS_A and $\text{NBS}_{A\epsilon}$ (A for *all* solutions) to denote these new versions which use the modified expansion policy (with \leq in Line 2) and aim to find a vertex cover of G_{MXA} and $G_{MXA\epsilon}$ respectively.

Note that there are many possible ways to implement the low level of NBS in terms of how to move nodes from waiting_D to ready_D . NBS_F and NBS_A are special cases directly inspired by G_{MX} and G_{MXA} .

4.3 Finding a First Solution with NBS_A

An interesting phenomenon is that although NBS_A is designed to find *all* solutions, it may expand fewer nodes than NBS_F , even when finding the first solution. The explanation for this is as follows. The low level of NBS_A utilizes more information about G_{MX} when making a decision. In an iteration where $LB < C^*$, nodes with $f = LB$ are part of G_{MX} , and considering them earlier helps in increasing LB faster, thus finding an MVC faster. In iterations where $LB = C^*$, a VC of G_{MX} has already been found, and nodes with $f = LB$ can lead to a solution if one was not yet discovered.

An example of this phenomenon is presented in Figure G.2, where $C^* = 4$ (the edge between s and g). Both NBS_F and NBS_A begin by expanding $\langle s, g \rangle$ ($LB = 1$), followed by $\langle A, H \rangle$ ($LB = 2$), at which point LB is incremented to 3. For NBS_F ready_B will contain only K ($f_F(K) = 2$ while $f = LB = 3$ for all other nodes). NBS_F will expand $\langle B, K \rangle$ before moving other nodes to ready_B (using Lines 10–11). Next, it will expand $\langle C, E \rangle$, $\langle D, F \rangle$ and terminate after expanding 10 nodes. By contrast, in NBS_A after setting $LB = 3$ ready_B will contain E, F, I, J and K (all with $f \leq LB = 3$). Since nodes with lower g -values are expanded first, NBS_A will expand $\langle B, E \rangle$ and $\langle C, F \rangle$, terminating with 8 node expansions, without expanding nodes K and D (since $g_F(D) + g_B(K) = 4 > 3 = LB$). Our experiments below suggest that this phenomenon is rather common in practice.

Note that every pair expanded by NBS_A in every iteration where $LB < C^*$ is an edge of G_{MX} . Thus, NBS_A retains the $2|\text{MVC}|$ bound until finding a VC of G_{MX} .

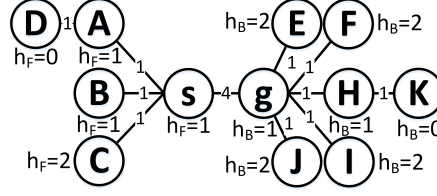


Figure G.2: Comparing NBS_A and NBS_F

5 Bidirectional Search using Dynamic VC

We now introduce a new family of algorithms called Dynamic Vertex Cover Bidirectional Search (DVCBS). It uses the high level of LBF but conceptually differs from the NBS family in its low-level expansion policy. While NBS always expands both nodes of a chosen MEP, DVCBS works by maintaining a *dynamic* version of G_{MX} (DG_{MX}) and greedily expanding an MVC of the DG_{MX} at each step.

DG_{MX} is defined as follows. Its structure resembles G_{MX} , with two main differences: (1) The full G_{MX} is not available during the search. Instead, DG_{MX} contains only nodes in the forward frontier (generated not expanded) for constructing left vertices, and only nodes from the backward frontier for constructing right vertices. (2) The value of C^* is not known during the search, thus edges of DG_{MX} are defined on pairs $\langle u, v \rangle$ such that $lb(u, v) < LB$. Since $LB \leq C^*$, all such pairs are in fact MEPs of G_{MX} .

Note that DG_{MX} shares all the interesting properties of the full G_{MX} . Thus, vertices with the same g -value can be merged to form a weighted vertex (cluster). More importantly, $\text{CalculateWVC}()$ can be directly applied to DG_{MX} in time linear in the number of its clusters. This is done in all low-level variants of DVCBS presented next.

5.1 Low-Level Expansion Policy in DVCBS

There are many possible low-level expansion policies based on DG_{MX} and on its MVC. Every node expansion deletes vertices and may add new vertices to DG_{MX} , invalidating the most recently computed MVC. However, computing the MVC every time DG_{MX} changes incurs extra overhead (albeit linear in the number of clusters in DG_{MX}). Thus, an efficient expansion policy should balance between expanding many nodes

and maintaining the most up-to-date DG_{MX} and MVC. We experimented with multiple expansion policy variants, and found that an efficient balance between these two extremes is to expand a single cluster (containing all nodes with the same g_F - or g_B -value) in every iteration of the high level. This results in a manageable amount of MVC computations, while working on reasonably up-to-date information. Furthermore, since all vertices in a cluster have the same g -value, LB may increase only after expanding an entire cluster but never before. We only report experimental results for this variant.

DVCBS contains several other decision points. First, there can be several possible MVCs for a given DG_{MX} . Additionally, as mentioned above, one cluster from MVC should be chosen and expanded. Finally, the way we order nodes within the cluster for expansion may affect the number of expansions before reaching a solution when $LB = C^*$. We have experimented with many possible decision choices but report the results in Section 6 using the best variant as follows. Select the cluster with the smallest number of nodes among the clusters with minimal g_F - and g_B -values, among all MVCs. Tie breaking for specific node expansion within a cluster orders nodes according to their order of discovery.

Pseudo code of the low level of DVCBS appears in Algorithm 3. The life cycle of DVCBS includes the following steps: (1) initialize DG_{MX} , (2) `CalculateWVC()`, (3) choose the cluster of nodes to expand from the MVC, and (4) update DG_{MX} . Steps 2-4 are repeated until either an optimal solution is found or no possible solution exists. To execute efficiently, DVCBS uses data structures denoted as $Cwaiting_D$ and $Cready_D$, which are similar to the $waiting_D$ and $ready_D$ queues of NBS, modified to use clusters.

5.2 Variants of DVCBS

Like NBS, DVCBS also has four variants corresponding to the four versions of G_{MX} . The variants that use G_{MX} and G_{MX_e} are denoted by $DVCBS_F$ and $DVCBS_{F_e}$ which *lazily* move nodes with $f_D = LB$ from $Cwaiting_D$ to $Cready_D$. Likewise, variants that use DG_{MXA} (a dynamic graph based on G_{MXA} , i.e., based on the conditions of MEAPs) can be derived by adapting the low-level expansion policy to G_{MXA} and G_{MXA_e} . Specifically, as was done for NBS, we modify the DVCBS expansion policy to *immediately* consider all nodes for which $f_D(u) \leq LB$ by changing the $<$ condition in Line 2 of Algorithm 3 to be

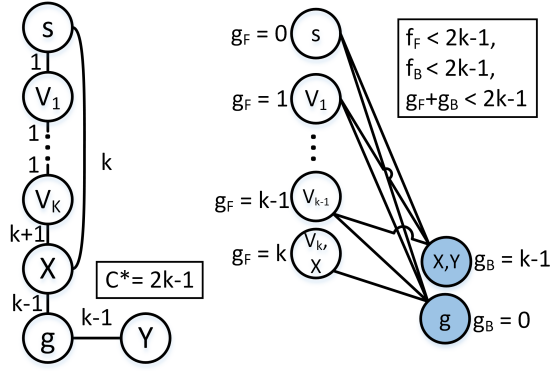


Figure G.3: An example for unbounded behavior of DVCBS

\leq . This change also eliminates Lines 11–13, as we handle such nodes immediately in Line 2. These variants are called DVCBS_A and DVCBS_{Ac} .

Here too, DVCBS_A can also be used to find a first solution, sometimes faster than DVCBS_F , as we demonstrate using Figure G.2. Initially, $LB = 1$. Since no nodes have $f_D < LB$, $DG_{MX} = DG_{MXA} = \{U_F = \{s\}, V_B = \{g\}, E = \{\langle s, g \rangle\}\}$. Assume that both DVCBS_F and DVCBS_A selected s for expansion and so $\{A, B, C\}$ are added to $waiting_F$. Their minimal f -value is 2 (A and B) so $LB = 2$. There are no clusters in $waiting_F$ with $f_F < LB$, thus, $\{A, B\}$ are moved to $ready_F$ and $DG_{MX} = DG_{MXA} = \{U_F = \{A, B\}, V_B = \{g\}, E = \{\langle A, g \rangle, \langle B, g \rangle\}\}$. Therefore, $\{g\}$ is the MVC, and both algorithms expand g and add $\{E, F, H, I, J\}$ to $waiting_B$. Next (LB is still 2), H is added to $ready_B$ and since H is the MVC, it is expanded and K is added to $waiting_B$. Now, $\{K\}$ is the only cluster in $waiting_B$ with $f_B \leq LB$. Since $g_B(K) = 2$ and $gmin_F = 1$ ($\{A, B\}$) LB is incremented to 3. At this point the algorithms diverge. DG_{MXA} moves C to $ready_F$ and $\{E, F, I, J, K\}$ to $ready_B$. Thus, DG_{MXA} includes 3 clusters with $f_D \leq LB = 3$: $\{A, B, C\}$ with $g_F = 1$ in $ready_F$, and two clusters in $ready_B$: $\{E, J, F, I\}$ with $g_B = 1$, and $\{K\}$ with $g_B = 2$. Thus, DVCBS_A expands cluster $\{A, B, C\}$ (it is the MVC), then, D is generated and expanded and DVCBS_A terminates after expanding a total of 7 nodes (s, g, H, A, B, C and D). By contrast, when $LB = 3$, DG_{MX} contains only two clusters with $f_D < LB = 3$: $\{A, B\}$ (with $g_F = 1$) in $ready_F$ and $\{K\}$ (with $g_F = 2$) in $ready_B$. Thus, DVCBS_F expands K (node C , as well as $\{E, J, F, I\}$ are added to $ready_D$, with $f_D = LB = 3$). Then it expands cluster $\{A, B, C\}$. Next it expands D and terminates, after expanding a total of 8 nodes (s, g, H, K, A, B, C and D). Recall that NBS_F expands 10 nodes and NBS_A expands 8 on this example.

Algorithm 3: DVCBS Expand a Level

```

1 while true do
2   while  $\min f \text{ in } C_{\text{waiting}_D} < LB$  do
3     Move best cluster from  $C_{\text{waiting}_D}$  to  $C_{\text{ready}_D}$ 
4   if  $C_{\text{ready}_D} \cup C_{\text{waiting}_D}$  empty then
5     Terminate search - no solution was found
6    $DG_{MX} \leftarrow \text{Build}DG_{MX}(C_{\text{ready}_D})$ 
7   if  $DG_{MX}$  is not empty then
8      $MVC \leftarrow \text{find}MVC(DG_{MX})$ 
9     Choose and Expand a cluster from MVC of  $DG_{MX}$ .
10  else
11    if  $C_{\text{waiting}_D}.f \leq LB$  then
12      Move best cluster from  $C_{\text{waiting}_D}$  to  $C_{\text{ready}_D}$ 
13    else
14      return true

```

5.3 No Upper Bound Guarantees for DVCBS

The most important property of NBS is the $2\times$ bound guarantee. While DVCBS outperforms NBS on average (see experiments below), DVCBS is not bounded in its worst case. A synthetic example and its G_{MX} demonstrate this in Figure G.3. The optimal path is $\langle s, X, g \rangle$ of cost $k + (k - 1) = 2k - 1$. Note that there is a longer path to X via the v_i nodes of cost $2k + 1$. In this example, the MVC of G_{MX} includes three nodes (g , X and Y in the backward direction, all colored blue). We next show that DVCBS never expands Y , and therefore has to expand at least $k + 2$ nodes — all connected to Y in G_{MX} . To expand Y , an algorithm needs to generate it by expanding g . If at any point DVCBS chooses to expand g then DG_{MX} will have two nodes in the backward side ($\{X, Y\}$) and a single node in the forward side (s or one of the V_i nodes). Thus, the MVC of DG_{MX} is always in the forward direction (choosing the V_i node), and DVCBS has to expand all of s, V_1, \dots, V_{k-1} before converging to the size $k + 1$ VC of G_{MX} . Otherwise, if g is never chosen for expansion, DVCBS always chooses to expand nodes in the for-

ward direction and it has to expand $k + 2$ nodes (s , X and all of the V_i s) in order to find a VC. In both cases, DVCBS expands more than k nodes. Since k can be arbitrarily large, DVCBS is not bounded by a constant factor of the MVC.

5. Bidirectional Search using Dynamic VC

Domain	Heuristic	Algorithm	base-case			ϵ -case		
			VC: G_{MX}	first	all: G_{MXA}	VC: G_{MX_ϵ}	first	all: G_{MXA_ϵ}
14 Pancake	GAP	A*	32 (1.22)	57	941 (1.17)	32 (1.23)	57	941 (1.24)
		NBS _F	49 (1.88)	163	1,338 (1.67)	47 (1.83)	147	1,224 (1.61)
		NBS _A	44 (1.70)	258	1,106 (1.38)	41 (1.57)	310	932 (1.23)
		DVCBS _F	31 (1.18)	106	880 (1.10)	30 (1.14)	121	832 (1.09)
		DVCBS _A	32 (1.24)	191	901 (1.12)	31 (1.18)	284	793 (1.04)
	GAP-1	A*	6,410 (1.39)	6,412	81,705 (1.56)	6,404 (1.73)	6,416	81,694 (2.11)
		NBS _F	7,184 (1.55)	7,226	80,192 (1.53)	5,870 (1.59)	5,915	62,374 (1.61)
		NBS _A	5,656 (1.22)	5,705	61,699 (1.18)	4,332 (1.17)	4,527	45,746 (1.18)
		DVCBS _F	5,319 (1.15)	5,341	61,278 (1.17)	4,321 (1.17)	4,344	45,206 (1.17)
		DVCBS _A	4,818 (1.04)	4,886	52,747 (1.01)	3,750 (1.01)	9,955	38,819 (1.00)
	GAP-2	A*	322,299 (2.65)	322,378	2,659,657 (3.33)	322,099 (4.15)	322,938	2,659,326 (5.61)
		NBS _F	208,648 (1.71)	209,723	1,393,062 (1.74)	137,295 (1.77)	137,719	842,947 (1.78)
		NBS _A	151,616 (1.24)	152,046	991,354 (1.24)	96,774 (1.25)	99,773	614,320 (1.30)
		DVCBS _F	141,111 (1.16)	141,669	864,611 (1.08)	86,292 (1.11)	87,012	493,288 (1.04)
		DVCBS _A	122,054 (1.00)	122,587	800,105 (1.00)	77,595 (1.00)	168,176	474,315 (1.00)
15 Puzzle	MD	NBS _F	13,542,536 (N/A)	13,587,955	28,117,879 (N/A)	12,709,517 (N/A)	12,748,107	26,162,236 (N/A)
		NBS _A	12,696,359 (N/A)	12,817,989	24,649,233 (N/A)	11,739,393 (N/A)	12,556,299	22,648,690 (N/A)
		DVCBS _F	11,863,100 (N/A)	11,940,791	25,717,691 (N/A)	11,589,837 (N/A)	11,669,720	24,088,398 (N/A)
		DVCBS _A	11,253,941 (N/A)	11,449,406	23,276,239 (N/A)	10,659,744 (N/A)	11,933,791	21,619,261 (N/A)
Grids DAO	Octile	A*	5,322 (1.25)	5,406	5,758 (1.20)	5,322 (1.25)	5,406	5,758 (1.20)
		NBS _F	6,569 (1.54)	6,686	6,952 (1.45)	6,561 (1.54)	6,677	6,942 (1.44)
		NBS _A	6,555 (1.54)	6,888	6,932 (1.44)	6,547 (1.53)	6,880	6,919 (1.44)
		DVCBS _F	5,158 (1.21)	5,546	5,594 (1.16)	5,158 (1.21)	5,545	5,593 (1.16)
		DVCBS _A	5,154 (1.21)	5,547	5,590 (1.16)	5,152 (1.21)	5,546	5,586 (1.16)
TOH4	10+2	A*	276,081 (2.25)	276,089	353,130 (2.28)	276,081 (2.25)	276,089	353,130 (2.28)
		NBS _F	234,165 (1.91)	234,165	291,195 (1.88)	232,509 (1.90)	232,509	288,177 (1.86)
		NBS _A	232,268 (1.89)	232,268	288,583 (1.86)	230,108 (1.88)	230,108	285,073 (1.84)
		DVCBS _F	225,910 (1.84)	225,910	273,210 (1.76)	224,233 (1.83)	224,249	270,715 (1.74)
		DVCBS _A	218,820 (1.78)	218,820	280,800 (1.81)	217,247 (1.77)	219,022	278,286 (1.79)
	6+6	A*	3,239,287 (4.75)	3,268,093	3,674,518 (4.89)	3,239,287 (5.19)	3,268,093	3,674,518 (5.34)
		NBS _F	731,446 (1.07)	731,522	796,289 (1.06)	663,136 (1.06)	681,995	732,638 (1.07)
		NBS _A	730,562 (1.07)	730,597	795,564 (1.06)	662,424 (1.06)	681,989	732,303 (1.06)
		DVCBS _F	704,213 (1.03)	707,679	766,722 (1.02)	636,375 (1.02)	664,469	695,950 (1.01)
		DVCBS _A	690,389 (1.01)	691,159	757,484 (1.01)	627,983 (1.01)	660,555	690,348 (1.00)

Table G.1: Experimental results of average node expansions across domains

6 Experimental Evaluation

We ran experiments on four domains: **(1)** 50 **14-Pancake Puzzle** instances with the GAP heuristic (Helmert 2010). To get a range of heuristic strengths, we also used the GAP- n heuristics (for $n = 1 \dots 3$) where the n smallest pancakes are left out of the heuristic computation. **(2)** The standard 100 instances of the **15 Puzzle** problem (Korf 1985) using the Manhattan Distance heuristic. **(3)** **Grid-based** pathfinding: 156 maps from Dragon Age Origins (DAO) (Sturtevant 2012), each with different start and goal points (a total of 3150 instances); **(4)** 50 instances of the 12-disk **4-peg Towers of Hanoi** (TOH4) problem with (10+2), (8+4) and (6+6) additive PDBs (Felner et al. 2004).

Table G.1 presents results averaged over all instances for a representative set of the heuristics we used. The same trends were observed for other heuristics. The left side of the table is for the base-case while the right side is for the ϵ -case. Four low-level expansion policies were executed until all optimal solutions were found: NBS_F , NBS_A , $DVCBS_F$ and $DVCBS_A$. For comparison reasons we also added A^* as a baseline. We report the number of nodes expanded at three different points of the execution, each in a different column, as follows. **(1)** The VC column presents the number of nodes expanded until the algorithm reached a VC of the corresponding G_{MX} . The number reported in parenthesis is the *ratio* (i.e., the relative size) of the discovered VC compared to an oracle (Shaham, Felner, Chen, et al. 2017), that built the entire G_{MX} (by running A^* in both directions) and found its exact MVC. Numbers close to 1 indicate nearly optimal VCs. Due to memory limits, some MVCs could not be computed (N/A). **(2)** The *first* column shows the number of nodes expanded until the first solution was found and verified. **(3)** The *all* column gives the number of nodes expanded until *all* optimal solutions were found (i.e., exactly when a VC of G_{MXA}/G_{MXA_ϵ} is found). Here, the ratio relative to the optimal MVC of G_{MXA}/G_{MXA_ϵ} is reported.

Runtime results are reported in Table G.2. The node expansion rates of all variants were similar, with very low variance. Therefore, we use the number of node expansions as the measure in the following analysis of the results.

Previous research (Chen et al. 2017; Sturtevant and Felner 2018) reported that NBS tends to outperform and is more robust than A^* and other related Bi-HS algorithms (e.g., MM). Table G.1 confirms that A^* is not as robust as the LBF family. In some cases,

6. Experimental Evaluation

e.g., the 15 puzzle, A^* failed to solve all instances because memory was exhausted. Except for cases where the heuristic is very good (where MVC might be unidirectional), A^* 's performance is much worse than the LBF family in all three measures. See (Shaham, Felner, Chen, et al. 2017) for a deeper study on the relation between A^* and MVC.

Since NBS has a 2x bound guarantee, any other algorithm will expand no fewer than half the nodes of NBS, leaving little leeway. Yet, our new algorithms managed to improve upon NBS and the following trends are evident. First, within the NBS family, NBS_A and $NBS_{A\epsilon}$ outperform NBS_F and $NBS_{F\epsilon}$, respectively, in terms of finding a VC of G_{MX} and of G_{MXA} . Moreover, they found the first solution faster than $NBS_F/NBS_{F\epsilon}$ in all cases except GAP and DAO.

Second, both DVCBS variants always outperformed the NBS variants in all three measures in the base-case, with $DVCBS_A$ almost always being best. In the ϵ -case, $DVCBS_F$ outperformed NBS_F in all three measures, while $DVCBS_A$ outperformed NBS_A in VC and *all*. We note that the VCs discovered by the DVCBS variants were often much closer (e.g., GAP-1; 55% vs. 4%, a factor of 14) to being optimal compared to the VCs discovered by the NBS variants. In fact, in some cases, with a weak heuristic, $DVCBS_A$ managed to find the exact MVC(!) of G_{MX} (a ratio of 1).

Finally, an interesting anomaly occurs with $DVCBS_{A\epsilon}$. It was the fastest to reach a VC of $G_{MX\epsilon}$ but was rarely the fastest to find a first solution; in such cases DVCBS was best among all algorithms. For example, for GAP-2, $DVCBS_{A\epsilon}$ expanded 77,595 nodes to find a VC of $G_{MX\epsilon}$ while DVCBS found a VC after 86,292 expansions. However, $DVCBS_{A\epsilon}$ expanded 90,581 more nodes (totaling 168,176) before discovering a first solution, while $DVCBS_{F\epsilon}$ expanded only 720 additional nodes (totaling 87,012). We conjecture that the reason is that in the ϵ -case, the frontiers may not be connected (i.e., same node in both frontiers) when a VC is found, and $DVCBS_{A\epsilon}$ must perform many additional node expansions before connecting the frontiers and finding a solution. However, other algorithms seem to perform more expansions before finding a VC, but they are able to connect the frontiers during this process. We intend to study this behavior further in future work.

To summarize, $DVCBS_A$ is clearly the algorithm of choice (among all 4) when all optimal solutions are needed. When only a first solution is needed, $DVCBS_A$ is the best

Alg	14 Pancake	15 Puzzle	Grids DAO	TOH4
A*	92,697	N/A	1,821,205	380,325
NBS _F	93,176	250,518	1,567,131	402,616
NBS _A	98,868	233,166	1,604,500	408,415
DVCBS _F	98,448	235,621	1,417,141	418,944
DVCBS _A	86,339	259,756	1,457,497	460,368

Table G.2: Average node expansions per second

Domain	BS*	MM ϵ	DVCBS _{Fϵ}	A*
GAP-0	183	149	121	57
GAP-1	5,262	5,048	4,344	6,416
GAP-2	266,442	119,310	87,012	322,938
10+2	174,936	303,189	224,249	276,089
6+6	1,599,018	1,120,392	664,469	3,268,093
MD	12,001,024	13,162,312	11,669,720	N/A
Octile	6,200	7,396	5,545	5,406

Table G.3: Average expansions for first solution (ϵ -case)

in the base-case, while DVCBS_{F ϵ} is the best in ϵ -case. Both always outperform any of the NBS variants, despite not having any theoretical guarantees.

We have also compared DVCBS_{F ϵ} (which is our best variant for finding a first solution in the ϵ -case) to A* as well as to MM ϵ (Holte et al. 2017) and BS* (Kwa 1989) which are benchmark Bi-HS algorithms. Table G.3 presents the average number of node expansions for finding a first solution in the ϵ -case. As can be seen, DVCBS_{F ϵ} tends to outperform all others, and is certainly the most robust to weaker heuristic.

7 Conclusions and Future Research

We have enriched the family of non-parametric Bi-HS algorithms as well as the family of G_{MX} graphs while also focusing on the problem of finding *all optimal solutions*. We have shown that our new algorithms outperform existing ones. We aim to look deeper in these directions in the future, and study additional variants and their relative performance.

Acknowledgements

This work was supported by Israel Science Foundation (ISF) grant #844/17 to Ariel Felner and Eyal Shimony, by BSF grant #2017692, by NSF grant #1815660 and by the Frankel center for CS at BGU.

References of Paper G

- [Art+97] Jeffrey L Arthur et al. “Finding all optimal solutions to the reserve site selection problem: formulation and computational analysis”. In: *Environmental and Ecological Statistics* 4.2 (1997), pp. 153–165.
- [Bar+18] Michael W. Barley et al. “GBFHS: A Generalized Breadth-First Heuristic Search Algorithm”. In: *SoCS*. 2018, pp. 28–36.
- [BW84] Thomas H Byers and Michael S Waterman. “Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming”. In: *Operations Research* 32.6 (1984), pp. 1381–1384.
- [Che+17] Jingwei Chen et al. “Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions”. In: *IJCAI*. ijcai.org, 2017, pp. 489–495.
- [DP85] Rina Dechter and Judea Pearl. “Generalized Best-First Search Strategies and the Optimality of A*”. In: *J. ACM* 32.3 (1985), pp. 505–536.
- [Eck+17] Jurgen Eckerle et al. “Sufficient Conditions for Node Expansion in Bidirectional Heuristic Search”. In: *ICAPS*. 2017, pp. 79–87.
- [FKH04] Ariel Felner, Richard E. Korf, and Sarit Hanan. “Additive Pattern Database Heuristics”. In: *J. Artif. Intell. Res.* 22 (2004), pp. 279–318. DOI: 10.1613/jair.1480. URL: <https://doi.org/10.1613/jair.1480>.
- [Hel10] Malte Helmert. “Landmark Heuristics for the Pancake Problem”. In: *SoCS*. 2010.
- [Hol+17] Robert C. Holte et al. “MM: A bidirectional search algorithm that is guaranteed to meet in the middle”. In: *Artif. Intell.* 252 (2017), pp. 232–266.
- [Ise77] Heinz Isermann. “The enumeration of the set of all efficient solutions for a linear multiple objective program”. In: *Journal of the Operational Research Society* 28.3 (1977), pp. 711–725.
- [Kor85] Richard E. Korf. “Depth-First Iterative-Deepening: An Optimal Admissible Tree Search”. In: *Artif. Intell.* 27.1 (1985), pp. 97–109. DOI: 10.1016/0004-3702(85)90084-0. URL: [https://doi.org/10.1016/0004-3702\(85\)90084-0](https://doi.org/10.1016/0004-3702(85)90084-0).

References of Paper G

- [Kwa89] James B. H. Kwa. "BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm". In: *Artif. Intell.* 38.1 (1989), pp. 95–109.
- [MS03] R Mahadevan and CH Schilling. "The effects of alternate optimal solutions in constraint-based genome-scale metabolic models". In: *Metabolic engineering* 5.4 (2003), pp. 264–276.
- [PS82] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., 1982.
- [SF18] Nathan R. Sturtevant and Ariel Felner. "A Brief History and Recent Achievements in Bidirectional Search". In: *AAAI*. 2018.
- [Sha+17] Eshed Shaham, Ariel Felner, Jingwei Chen, et al. "The Minimal Set of States that Must Be Expanded in a Front-to-End Bidirectional Search". In: *SoCS*. 2017, pp. 82–90.
- [Sha+18] Eshed Shaham, Ariel Felner, Nathan R. Sturtevant, et al. "Minimizing Node Expansions in Bidirectional Search with Consistent Heuristics". In: *SOCS*. 2018, pp. 81–98.
- [SND12] Florian Siegmund, Amos H. C. Ng, and Kalyanmoy Deb. "Finding a preferred diverse set of Pareto-optimal solutions for a limited number of function calls". In: *IEEE World Congress on Computational Intelligence (CEC)*. 2012, pp. 2417–2424.
- [Stu12] Nathan R. Sturtevant. "Benchmarks for Grid-Based Pathfinding". In: *IEEE Trans. Comput. Intellig. and AI in Games* 4.2 (2012), pp. 144–148. DOI: 10.1109/TCIAIG.2012.2197681. URL: <https://doi.org/10.1109/TCIAIG.2012.2197681>.

Paper H

Improving Bidirectional Heuristic Search by Bounds Propagation

Shahaf S. Shperberg, Ariel Felner, Avi Hayoun, Solomon Eyal
Shimony, Nathan R. Sturtevant

The paper has been published in the
SoCS 2019: The Twelfth Annual Symposium on Combinatorial Search
pp. 106–114, 2019.

This paper has received the best paper award of the conference.

© 2019 SoCS

The layout has been revised.

Abstract

Recent research on bidirectional search describes anomalies, or cases in which improved heuristics lead to more node expansions. Aiming to avoid such anomalies, this paper characterizes desirable properties for bidirectional search algorithms, and studies conditions for obtaining these properties. The characterization is based on a recently developed theory for bidirectional search, which has formulated conditions on pairs of nodes such that at least one node from every pair meeting these conditions must be expanded. Moreover, based on this must-expand-pairs theory, we introduce a method for enhancing heuristics by propagating lower bounds (lb-propagation) between frontiers. This lb-propagation can bestow the desirable properties on some existing algorithms (e.g., the MM family) while avoiding the above anomaly altogether. Empirical results show that lb-propagation reduces the number of node expansions in many cases.

1 Introduction

Bidirectional heuristic search (Bi-HS) algorithms interleave two separate searches: a search forward from *start*, and a search backward from *goal*. Recently, a new line of research into Bi-HS was spawned. Eckerle et al. 2017 defined three conditions on the node expansions required by Bi-HS algorithms to guarantee solution optimality. Following work reformulated these conditions as a *must-expand graph* (G_{MX}). It was shown that the *Minimum Vertex Cover* (MVC) of G_{MX} corresponds to the minimal number of expansions required to prove optimality (Chen et al. 2017). Finally, a number of algorithms were introduced. NBS (Chen et al. 2017) and DVCBS (Shperberg et al. 2019) are non-parametric G_{MX} -based Bi-HS algorithms that aim to find a vertex cover of G_{MX} quickly, but in different ways. *Fractional MM* ($fMM(p)$) (Shaham, Felner, Chen, et al. 2017) is a parametric algorithm that generalizes the MM algorithm (Holte et al. 2017) by controlling the fraction p of the optimal path at which the forward and backward frontiers meet. Another parametric algorithm, GBFHS (Barley et al. 2018), iteratively increases the depth of the search by using a *split function* to determine how deep to search on each side at each iteration.

Holte et al. 2017 observed an *anomaly* where improving a heuristic caused the

MM algorithm to expand more nodes. Aiming to generalize this anomaly beyond MM, Barley et al. 2018 defined that an algorithm is *well-behaved* if using a better heuristic will never hurt its performance; otherwise, it is *ill-behaved*. In this paper we expand this line of work on Bi-HS in several ways.

First, we study and develop desirable properties for Bi-HS algorithms by re-formalizing the *well-behaved* property and providing a definition which is even more general than that of Barley et al. 2018. We also introduce the *reasonable* property which guarantees that an algorithm will never expand nodes if the lower-bound associated with them is greater than the current global lower bound (LB) on the optimal solution. We then introduce and prove sufficient conditions required to fulfill each property.

Second, building on the conditions of Eckerle et al. 2017, we introduce *lb-propagation*, a method for propagating the best lower-bound between the two search frontiers, thereby improving heuristics and the f -values in each frontier. *lb-propagation* can be used on top of any Bi-HS algorithm; it is already used implicitly in G_{MX} -based algorithms such as NBS and DVCBS. We show that *lb-propagation* causes the MM family to become well-behaved and reasonable, thereby avoiding the anomaly, although some algorithms, such as BS^* , cannot be fixed in this way.

Third, we perform a study on a number of algorithms, characterizing those that are inherently well-behaved and reasonable, as well as whether or not *lb-propagation* bestows these properties on the algorithms. Finally, we show experimentally that *lb-propagation* reduces the number of node expansions for non- G_{MX} -based algorithms.

1.1 Definitions and Background

A shortest-path problem, P , is defined as a tuple $(G = \{V, E\}, start, goal)$ in which G is a graph and $start, goal \in V$. The aim of such problems is to find the least-cost path between $start$ and $goal$. Let $d(x, y)$ denote the shortest distance between x and y and let $C^* = d(start, goal)$. In some cases, the minimal edge-cost is known beforehand; this minimal cost is denoted by ϵ .

Most Bi-HS algorithms maintain two open lists: $Open_F$ for the forward search and $Open_B$ for the backward search. There are two types of heuristics in bidirectional search. *Front-to-front* heuristics (Champeaux 1983; Champeaux and Sint 1977) estimate the distance between any two nodes in the search space, while *front-to-end* heuristics

(Kaindl and Kainz 1997) estimate the distance from any node and the *start* or *goal*. Front-to-front heuristics may be more computationally expensive, and efficient data structures for front-to-front algorithms do not exist. This paper only considers front-to-end heuristics.

Given a direction D (either forward or backward) We use f_D , g_D and h_D to indicate f -, g -, and h -values in direction D . In addition, $fmin_D$ and $gmin_D$ represent the minimal f - and g -values in that direction.

The *forward heuristic* h_F is *admissible* iff $h_F(u) \leq d(u, g)$ for every state $u \in G$ and is *consistent* iff $h_F(u) \leq d(u, u') + h_F(u')$ for all $u, u' \in G$. The *backward heuristic* h_B is defined analogously. A pair of forward and backward heuristic functions is *bi-admissible* if both heuristics are admissible. Likewise, such a pair is *bi-consistent* if both heuristics are consistent. A search algorithm is admissible if it is guaranteed to find an optimal solution whenever its heuristic is admissible. Finally, a heuristic h_1 is said to *dominate* another heuristic h_2 if and only if for every node $n \in G$, $h_1(n) \geq h_2(n)$ (Russell and Norvig 2016). We limit the discussion in this paper to admissible deterministic black-box expansion-based algorithms (called *DXBB* by Eckerle et al. 2017) used with bi-admissible and bi-consistent heuristics.

1.2 Fractional MM

We use the MM family of algorithms as a case study, therefore briefly describe them next. MM is a Bi-HS algorithm that *meets in the middle* (Holte et al. 2017), i.e. it is guaranteed to never expand a node whose g -value exceeds $C^*/2$. Fractional MM ($fMM(p)$) is a generalization of MM that never expands a node in the forward direction whose g -value exceeds C^*/p , and never expands a node in the backward direction whose g -value exceeds $C^*/(1-p)$. For a given fraction $0 < p < 1$, $fMM(p)$ chooses a node for expansion according to the following priority functions:

$$\begin{aligned} pr_F(u) &= \max\{g_F(u) + h_F(u), \frac{g_F(u)}{p} + \epsilon\} \\ pr_B(v) &= \max\{g_B(v) + h_B(v), \frac{g_B(v)}{1-p} + \epsilon\} \end{aligned}$$

A node with minimal priority in either direction is chosen for expansion.¹ fMM

¹For $p = 1$ or $p = 0$ fMM runs forward- or backward A*. Additionally, the original definition of fMM and MM did not include ϵ , which was introduced in later versions of the algorithms: $MM\epsilon$ (Sharon et al. 2016) and $fMM\epsilon$ (Shaham, Felner, Sturtevant, et al. 2018).

terminates when one of the following conditions is met:

- One of $Open_F$ or $Open_B$ is empty.
- There exists a node v in both open lists with $C = g_F(v) + g_B(v)$ s.t. either:
 - $fmin_F \geq C$;
 - $fmin_B \geq C$;
 - $gmin_F + gmin_B + \epsilon \geq C$; or
 - $\min\{\min_{u \in Open_F} pr_F(u), \min_{v \in Open_B} pr_B(v)\} \geq C$.

Note that MM is a special case of $fMM(p)$ with $p = 1/2$.

Shaham, Felner, Chen, et al. 2017 showed that for every problem instance, there exists a fraction p^* such that $fMM(p^*)$ is optimally efficient and will expand the minimal number of nodes required to guarantee the optimality of its solution. However, p^* is not known a priori since it depends on the search-tree structure and the value of C^* .

1.3 GBFHS

GBFHS (general breadth-first heuristic search) (Barley et al. 2018) is a prominent bidirectional heuristic search algorithm that iteratively increases the depth of the search. For each depth, denoted by $fLim$, GBFHS uses a pre-defined *split function* (a "parameter" of the algorithm) that determines how deep to search on each side. The split function splits $fLim$ to $gLim_F$ and $gLim_B$, such that $fLim = gLim_F + gLim_B + \epsilon - 1$ (in unit edge cost domains $\epsilon - 1 = 0$). For a given iteration (i.e., a given value of $fLim$) all nodes with $f_D(n) \leq fLim$ and $g_D(n) < gLim_D$ are called *expandable*. GBFHS expands all *expandable* nodes from both directions. GBFHS terminates as soon as a solution with $cost = fLim$ is found. Specifically, GBFHS stops when there exists a node n in both open lists with $g_F(n) + g_B(n) \leq fLim$. If a solution is not found after expanding all *expandable* nodes, $fLim$ is incremented (adds 1), and as a result the split function updates $gLim_F$ or $gLim_B$ (such that $fLim = gLim_F + gLim_B + \epsilon - 1$). Then, a new iteration begins. The split function must update the g -limits ($gLim_F$ and $gLim_B$) in a consistent way, i.e., the values it returns must be *larger than or equal to* the previous values. This

means that when $fLim$ is incremented then one of $gLim_F$ and $gLim_B$ is incremented too.

GBFHS was shown to have some desirable properties when given a problem instance from I_{AD} : **(1)** it returns an optimal solution when the edges are non-negative integers; **(2)** in unit cost domains, the first solution GBFHS finds is guaranteed to be optimal; **(3)** Its frontiers can be made to meet anywhere using a proper split function; and **(4)** it is both *well-behaved* and *reasonable*.

Since GBFHS is guaranteed to return an optimal solution only when the edge costs are (non-negative) integers, we will assume such edge costs for the sake of the analysis. Nevertheless, we conjecture that GBFHS can be slightly modified in a way that would guarantee optimal solutions for any non-negative edge costs, while retaining all of its original properties. Investigating this conjecture is left for future work.

2 The Well-Behavedness Property

If h_1 and h_2 are consistent heuristics and $h_1(s) \geq h_2(s)$ for all non-goal nodes (i.e., h_1 dominates h_2), then every node expanded by A^* using h_1 will also be expanded by A^* using h_2 up to tie-breaking in the last f -layer (Holte 2010). Holte et al. 2017 describe an anomaly that may occur in Bi-HS algorithms such that a similar property does not hold. An example is provided in which MM using a global zero-heuristic (denoted henceforth by h_0 and the MM variant using it by MM_0) expands a subset of nodes that are expanded by MM that uses a stronger heuristic. Barley et al. 2018 also refer to the above anomaly, calling algorithms *well-behaved* if switching to a stronger heuristic does not lead to the expansion of any additional nodes, and *ill-behaved* otherwise. Well-behavedness has not been formally defined in a general manner; Holte et al. 2017 did not formally define the anomaly and Barley et al. 2018 defined it using terms that are specific to the GBFHS algorithm. We introduce a general definition of the *well-behavedness* property below and show that the anomaly results from a combination of (1) different tie-breaking, and (2) not using the theoretical lower-bound conditions for guiding the expansion process.

Many heuristic search algorithms do not fully specify which single node to expand at any given point in the search. For example, A^* may choose any node in

OPEN with a minimal f -value, and fMM can choose any node in either open list with minimal priority. Instead, these algorithms specify a set of nodes from the open lists (denoted henceforth by *allowable-set*) from which the next node must be expanded. An additional *tie-breaking* scheme is used to select a single node from the allowable-set. Tie-breaking is often specific to a given implementation, and in most cases is not part of the published algorithm definition. For example, A^* must expand nodes with the smallest f -value. There are many possible tie-breaking rules to decide how to break ties among nodes with the same f -value (e.g., smallest h or smallest g , generation order etc.). However, all of these tie-breaking functions are low-level details of A^* implementations.

We use $\mathcal{A}_h(I, t)$ to denote the sequence of nodes expanded by running algorithm \mathcal{A} using heuristic h on problem instance I with a tie-breaking function t , and by $S(\mathcal{A}_h(I, t))$ the (unordered) set of nodes induced by the expansion performed by $\mathcal{A}_h(I, t)$.

Definition H.1

Let h_1, h_2 be bi-admissible bi-consistent heuristics, such that h_1 dominates h_2 . Algorithm \mathcal{A} is said to be *well-behaved* if for every tie-breaking policy t and problem instance I , there exists a tie-breaking policy t' such that $S(\mathcal{A}_{h_1}(I, t')) \subseteq S(\mathcal{A}_{h_2}(I, t))$.

This is a general definition that can be used with any Bi-HS algorithm. To date, only A^* and GBFHS have been proven to be well-behaved, while MM has been shown to be ill-behaved (see below). This property has not been studied in other algorithms. We define conditions that enable the classification of algorithm as either well- or ill-behaved, covering a wider family of algorithms.

2.1 Example of the Anomaly for fMM

In order to explore algorithms that are ill-behaved, we borrow an example from Holte et al. 2017, depicted in Figure H.1. In this example $\epsilon = 1$ and the values inside nodes are h -values in the direction indicated by the arrow. We henceforth denote this bi-consistent heuristic by h_{fig} . MM_0 expands nodes by their g -value. Thus, MM_0 starts by expanding *start* and *goal* (priority of 0), after which nodes S_1, G_1, A , and C have a priority of 1. There exists a tie-breaking policy t in which MM_0 expands A, C , and S_1 and then terminates, since it finds a solution of cost 4 and $gmin_F + gmin_B + \epsilon = 4$.

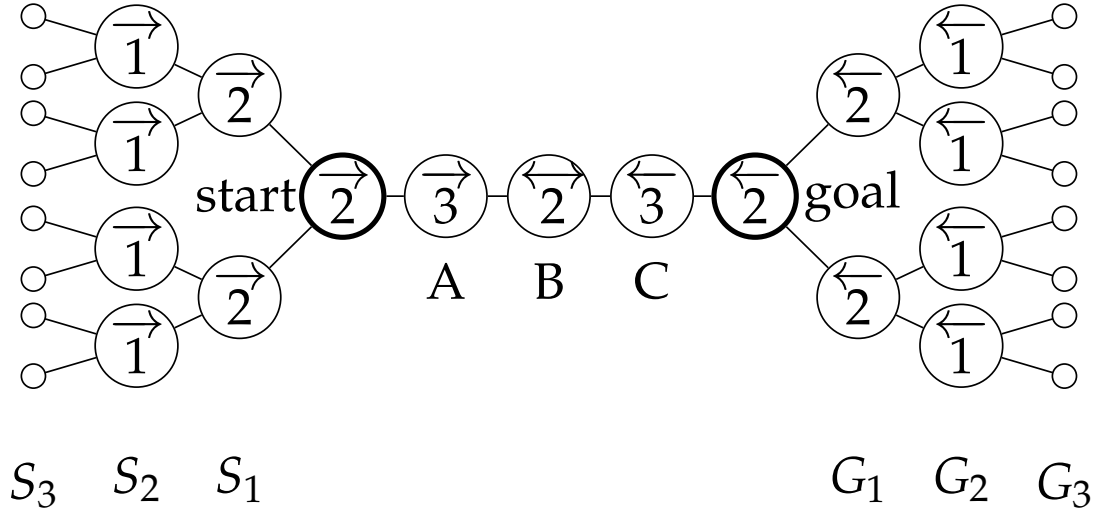


Figure H.1: An example in which the anomaly manifests

In contrast, MM must expand both S_1 and G_1 after expanding *start* and *goal* before expanding A and C , since S_1 and G_1 have a priority of $g + h = 2g + \epsilon = 3$, while A and C have a priority of 4 ($g + h = 4$). Consequently, there exists a tie-breaking policy t for MM_0 such that for every a tie-breaking policy t' the set of nodes expanded by running MM on this instance using t' is **not** a subset of the set of nodes expanded by MM_0 using t . Thus, MM is ill-behaved.

To understand why MM is ill-behaved, consider the situation after MM expanded *start*, *goal*, and S_1 . At this point, $Open_F$ contains S_2 ($g_F = 2, h_F = 1, pr_F = 5$), and A ($g_F = 1, h_F = 3, pr_F = 4$); $Open_B$ contains G_1 ($g_B = 1, h_B = 2, pr_B = 3$), and C ($g_B = 1, h_B = 3, pr_B = 4$). Thus, if the optimal solution goes through G_1 , it must go through either S_2 or A . If the optimal path goes through G_1 and S_2 , its cost would be at least $g_F(S_2) + g_B(G_1) + \epsilon = 4$. Similarly, if the optimal path goes through G_1 and A then its cost would be at least $f_F(A) = 4$. Hence, every path that goes through G_1 must have a cost of at least 4. The priority of G_1 ($pr_F(G_1) = 3$) doesn't reflect knowledge available in the search, which causes MM to be ill-behaved. This observation suggests that the sufficient conditions for node expansions (Eckerle et al. 2017) may be connected to whether an algorithm is well-behaved.

2.2 Guaranteeing Solution Optimality

Unidirectional search algorithms must expand all nodes n with $f(n) < C^*$ in order to guarantee the optimality of solutions (Dechter and Pearl 1985).

Eckerle et al. 2017 generalized this to Bi-HS by examining pairs of nodes $\langle u, v \rangle$ such that $u \in \text{Open}_F$ and $v \in \text{Open}_B$. Let ϵ be the minimal edge cost in G^2 . If u and v meet the following conditions, then every algorithm must expand at least one of u or v in order to ensure that there is no path from s to g passing through u and v of cost $< C^*$.

1. $f_F(u) < C^*$
2. $f_B(v) < C^*$
3. $g_F(u) + g_B(v) + \epsilon < C^*$

Definition H.2

For each pair of nodes (u, v) let $lb(u, v) = \max\{f_F(u), f_B(v), g_F(u) + g_B(v) + \epsilon\}$

In Bi-HS, a pair of nodes $\langle u, v \rangle$ is called a *must-expand pair* (MEP) if $lb(u, v) < C^*$. The MEP definition is equivalent to Eckerle’s conditions; for each MEP only *one* of u or v *must* be expanded. In the special case of unidirectional search, algorithms expand all the nodes with $f_F < C^*$, which is equivalent to expanding the forward node of every MEP. Bi-HS algorithms may expand nodes from either side, potentially covering all the MEPs with fewer expansions.

However, to address the ill-behavedness property we wish to bound the minimal solution cost that can pass through each node u in our open lists. To do so, we use the bound $lb(u, v)$ and apply it to every node v on the opposite frontier and take the minimum among these values. Formally, for every node u in Open_D let

$$lb(u) = \min_{v \in \text{open}_{\bar{D}}} \{lb(u, v)\}$$

where \bar{D} denotes the opposite direction from D . Then, $lb(u)$ is a lower bound on the cost of any solution that passes through u . Finally, we define the global lower bound LB to be the minimal $lb(u)$ among all nodes. This is identical to the minimal $lb(u, v)$ among all pairs. LB was used in the high-level pseudocode (described below) of the NBS and DVCBS algorithms. Note that the search begins with $LB = lb(\text{start}, \text{goal})$, after which LB increases iteratively until $LB = C^*$. We can now use these definitions to show whether a family of algorithms is well-behaved.

²Strictly speaking the ϵ term was added by Shaham, Felner, Sturtevant, et al. 2018 as a generalization of the inequalities, since $\epsilon = 0$ is always a lower-bound to edge cost.

2.3 Conditions for Being Well-Behaved

We first introduce three sufficient conditions for an admissible Bi-HS algorithm \mathcal{A} to be well-behaved:

Condition C1: Algorithm \mathcal{A} chooses a node u for expansion **only if** $lb(u) = LB$.

Condition C2: Algorithm \mathcal{A} terminates as soon as a solution with cost $C \leq LB$ is found.

Condition C3: The allowable-set of algorithm \mathcal{A} contains **every** node u with $lb(u) = LB$.

Theorem H.1

An admissible Bi-HS algorithm \mathcal{A} that satisfies conditions C1, C2, and C3 is well-behaved.

Proof. Let h_1, h_2 be bi-admissible bi-consistent heuristics, s.t. h_1 dominates h_2 , and let t_2 be an arbitrary tie-breaking policy. We will show that there exists a tie-breaking policy t_1 s.t. $S_1 = S(\mathcal{A}_{h_1}(I, t_1)) \subseteq S(\mathcal{A}_{h_2}(I, t_2)) = S_2$.

To do this, we examine the execution of $\mathcal{A}_{h_1}(I, t_1 = t_2)$ and show how to modify t_1 to make $S_1 \subseteq S_2$. Let n be the first node expanded in the trace of the execution s.t. $n \notin S_2$ (if no such node exists then $S_1 \subseteq S_2$ and we are done). In addition, let D be the direction in which n was expanded. We now have two cases:

Case 1: There exists a node n' in one of the frontiers s.t. $lb(n') = lb(n) = LB$ and $n' \in S_2$. given C3, we can modify t_1 to choose n' instead of n .

Case 2: All of the nodes n' in the frontiers with $lb(n') = lb(n) = LB$ are not in S_2 . We will show that this case is not possible. Note that since n was chosen for expansion, all other nodes m in the frontiers have $lb(m) \geq lb(n) = LB$. Let v be a node in $Open_{\overline{D}}$ s.t. $lb(n) = lb(n, v)$. Since $lb(v) \leq lb(n, v)$ and $lb(v) \geq lb(n)$ then $lb(n) = lb(v)$, and therefore $v \notin S_2$. Since both n and v are in OPEN using h_1 when n is chosen for expansion, and because n is the first to not be in S_2 , then all other nodes that were expanded by h_1 were expanded by h_2 . Thus, at some point in $\mathcal{A}_{h_2}(I, t_2)$, both n and v are in OPEN with a g -value less than or equal to that found in $\mathcal{A}_{h_1}(I, t_1)$. We can therefore refer to $lb(n, v)$ in $\mathcal{A}_{h_2}(I, t_2)$. Henceforth, $lb_i(n, v)$ refers to the value $lb(n, v)$ in $\mathcal{A}_{h_i}(I, t_i)$. Since h_1 dominates h_2 , and since the g -values cannot be larger

when using h_2 , $lb_2(n, v) \leq lb_1(n, v)$. We proceed by examining the possible values of $lb_1(n, v)$, and show that any value leads to a contradiction. There are three possible values of $lb_1(n, v)$ to consider:

- (i) If $lb_1(n, v) < C^*$, then $lb_2(n, v) < C^*$ as well. Therefore, $\langle n, v \rangle$ is an MEP and any admissible algorithm must expand one of them. The fact that $n, v \notin S_2$ contradicts the admissibility of \mathcal{A} .
- (ii) If $lb_1(n, v) = C^*$, then since $\mathcal{A}_{h_1}(I, t_1)$ expanded n , it did not yet find any solution of cost C^* . Since LB is a lower-bound on the optimal solution, a solution with cost C^* must pass through some node m with $lb(m) = lb(n) = C^*$ that was not yet expanded. Under the assumption of case 2, there are no nodes m with $lb(m) = lb(n) = LB$ in one of the frontiers that is also in S_2 . Therefore, $\mathcal{A}_{h_2}(I, t_2)$ will never find a solution of cost C^* . This is a contradiction to the assumption that \mathcal{A} is admissible.
- (iii) If $lb_1(n, v) > C^*$, then since $\mathcal{A}_{h_1}(I, t_1)$ expanded n and \mathcal{A} satisfies C2, it did not find any solution of cost C^* . Additionally, when n was chosen for expansion, the lb between every pair of nodes in the open lists is greater than C^* . Thus, a solution of cost C^* does not exist by contradiction to the definition of C^* . \square

These conditions are sufficient, but not necessary. In the next section we explore another desirable property.

3 The Reasonableness Property

While being well-behaved is an interesting property, some well-behaved algorithms do not behave sensibly. For example, an algorithm that completely ignores heuristic values and expands nodes according to their g -value is clearly well-behaved because a stronger heuristic will not change the behavior of the algorithm. However, such an algorithm might expand nodes n with $f(n) > C^*$ whose $g(n) \leq C^*$. Gilon et al. 2016 denoted algorithms as *reasonable* if they have a best-first structure (i.e. an open list and an expansion rule), and they prune any node n with $f(n) > C$, where C an upper bound on the cost. We generalize this notion as follows:

Definition H.3

A Bi-HS algorithm is *reasonable* if for every tie-breaking policy it does not expand a node v if either $lb(v) > C^*$, or if $lb(v) = C^*$ and a solution of cost C^* has already been

found.

Note that since $f(n) \leq lb(n)$ and $C^* \leq C$, the redefinition of the reasonable property is tighter than the original definition of Gilon et al. 2016.

Theorem H.2

Any admissible Algorithm \mathcal{A} that satisfies C1 and C2 is reasonable.

Proof. Let \mathcal{A} be an algorithm that always expands a node u with $lb(u) = LB$ and terminates as soon as a solution with a cost $c \leq LB$ is found. Assume by contradiction that $lb(u) > C^*$. Since $lb(u)$ is minimal ($lb(u) = LB$) then every solution that passes through every node in the open lists has a cost $> C^*$. Since C2 dictates that \mathcal{A} terminates when a solution with a cost $c = LB$ is found, no solution with cost C^* could have been found. Therefore, there is no possible solution with a cost of C^* , by contradiction to the definition of C^* . \square

To summarize both theorems, an algorithm that satisfies conditions C1 and C2 is reasonable, and one that also satisfies C3 is well-behaved. In both cases, the conditions are *sufficient* but not *necessary*.

4 Improving Heuristics by lb -propagation

We next introduce several methods that improve the heuristic value of a node by utilizing information gathered during the search in both frontiers. The strongest method which propagates lb -values causes some ill-behaved algorithms to become well-behaved (e.g., the MM family). In addition, algorithms that satisfy conditions C1 and C2 with respect to f instead of lb which use this method become reasonable. Note that this improvement is achieved by modifying only the heuristic, without any other changes to the algorithms.

4.1 Propagating g - and f -values

A simple observation on the nature of bidirectional search yields that the minimum $g_{\bar{D}}$ -value with the addition of ϵ is an admissible heuristic for any node in $Open_D$. Furthermore, we can propagate the minimal f -value from the opposite frontier because

it is a lower bound on any possible solution. Formally, let $gmin_D = \min_{v \in open_D} \{g(v)\}$ and let $fmin_{\bar{D}} = \min_{v \in open_{\bar{D}}} \{f_{\bar{D}}(v)\}$. We can improve the heuristic of node n in direction D to be:

$$h'_D(n) = \max\{h_D(n), gmin_{\bar{D}} + \epsilon, fmin_{\bar{D}} - g_D(n)\}$$

h' clearly dominates h and is easy to implement. One only needs to keep track of $gmin_D$ and $fmin_D$ for both directions. Nevertheless, h' does not solve the anomaly; MM using h' on Figure H.1 behaves identically to MM using the original heuristic, as described in Section 2.1

4.2 *lb*-propagation heuristic

The next heuristic exploits knowledge from *lb*-values. Let $h_{lb}(n) = lb(n) - g_D(n)$ denote the new heuristic function for nodes in direction D . Consider the following key observations: **(1)** h_{lb} is a dynamic heuristic that takes into account information generated by the search in the opposite direction. Therefore, its value for a node may change as the search proceeds. **(2)** Since $lb(n) \geq f_D(n)$, $h_{lb}(n) \geq h_D(n)$ for every node in both directions. **(3)** h_{lb} maintains the bi-consistency and bi-admissibility properties of h .

The heuristic h_{lb} dominates h' because h' looks at the global values of $gmin_{\bar{D}}$ and $fmin_{\bar{D}}$, while h_{lb} considers each pair of nodes in isolation. Despite the fact that h_{lb} dominates h' , using *lb*-propagation depends on the ability to efficiently compute the *lb* of nodes in OPEN. This task is certainly more difficult than applying the other propagation, which simply requires maintaining the minimal *f*- and *g*-values in each direction. In some algorithms the *lb*-propagation can be applied to a limited subset of OPEN, possibly enabling an efficient implementation (similar to NBS). In other cases, the *lb* of every node is required. This leads to a potentially less efficient implementation, using *g-h* buckets (Burns et al. 2012); this solution would work if the number of possible *g*-values (and therefore *h*-values) is relatively small, which is the case in many common domains.

An important property of h_{lb} is that it changes the *f*-values of nodes to be their *lb*-value, and therefore makes some existing algorithms well-behaved and reasonable as we show in the next section.

5. Classification of Existing Algorithms

Algorithm	Without lb -p		With lb -p	
	R	WB	R	WB
BHPA	×	✓	✓	✓
BS*	×	×	✓	×
fMM	×	×	✓	✓
GBFSH	✓	✓	✓	✓
NBS, DVCBS	✓	×	✓	×

Table H.1: Algorithm properties summary. R columns denote reasonableness, WB columns denote well-behavedness.

5 Classification of Existing Algorithms

As mentioned, lb -propagation makes the f -values of nodes identical to their lb -values. Therefore, any algorithm which chooses to expand nodes based on f -values and applies lb -propagation will now satisfy condition C1. However, in order to be provably reasonable it should also satisfy condition C2, and to be well-behaved, condition C3 is also needed. In this section, we review several Bi-HS algorithms and analyze how lb -propagation affects them. For any algorithm \mathcal{A} we henceforth denote by \mathcal{A}_{lb} a version of \mathcal{A} that applies lb -propagation. Table H.1 summarizes the results of this section, for algorithms with and without lb -propagation (lb -p).

5.1 BHPA

We begin with BHPA (Pohl 1971), a simple algorithm that first selects a direction and chooses to expand a node with minimal f -value in that direction. BHPA terminates when the minimal f -value is *greater than or equal to* C^* .

Lemma H.1

BHPA is well-behaved.

Proof. Let I be a problem instance, h_1 and h_2 be heuristics that are bi-admissible and bi-consistent on I s.t. h_1 dominates h_2 , and let t_2 be a tie-breaking policy. Let S_2 denote $S(BHPA_{h_2}(I, t_2))$ and let S_1 denote $S(BHPA_{h_1}(I, t_1 = t_2))$. Let u be the first node expanded in the trace of $BHPA_{h_1}(I, t_1)$ s.t. $u \notin S_2$. If u does not exist, we are done. Otherwise, we want to fix t_1 . If there exists a node $n \in S_2$ that has a minimal

f -value in either direction of $BHPA_{h_1}(I, t_1)$ when u was selected for expansion, we can alter t_1 to select n instead of u . Otherwise, there exists a node u' in the opposite direction of u with minimal f -value, and we could modify t_1 to select u' instead of u for expansion. We know that for every node v , $f_D^{h_1}(v) \geq f_D^{h_2}(v)$, thus $f_D^{h_1}(u) \geq f_D^{h_2}(u)$ and $f_D^{h_1}(u') \geq f_D^{h_2}(u')$. Therefore, if $f_D^{h_1}(u) < C^*$ and $f_D^{h_1}(u') < C^*$, we know that $f_D^{h_2}(u) < C^*$ and $f_D^{h_2}(u') < C^*$, hence S_2 must contain either u or u' , so that $BHPA_{h_2}(I, t_2)$ could terminate by contradiction to the fact that $u, u' \notin S_2$. Otherwise, $f_D^{h_1}(u) = C^*$ or $f_D^{h_1}(u') = C^*$. In this case, since $BHPA_{h_1}(I, t_1)$ is admissible and must find an optimal solution, there must be some other node $v \in S_2$ in the open lists when u was chosen for expansion s.t. $f_D^{h_1}(v) = C^*$ by contradiction to the case assumption. \square

Lemma H.2

BHPA is unreasonable.

Proof. Consider the problem instance I in Figure H.1 assuming that $h_F(S_3) = h_B(G_3) = 0$. Since for all $i \in \{1, 2, 3\}$, $f_F(S_i) = f_B(G_i) = 3$, while $f_F(A) = f_B(C) = 4$, running $BHPA_{h_{fig}}$ on I with any tie-breaking policy must expand either $\{S_1, S_2, S_3\}$, $\{G_1, G_2, G_3\}$, or both, before being able to expand A or C . Furthermore, there exists a tie-breaking in which $BHPA_{h_{fig}}$ expands *start* and *goal* followed by $\{S_1, S_2, S_3\}$. Since $lb(S_3) = g_F(S_3) + g_B(C) + \epsilon = 5 > C^* = 4$, BHPA is unreasonable. \square

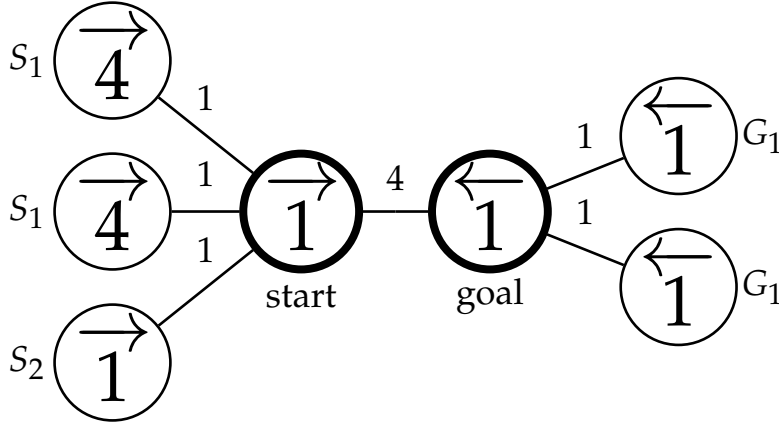
Lemma H.3

$BHPA_{lb}$ is reasonable and well-behaved.

Proof. Since after the propagation the f -value of a node equals its lb , $BHPA_{lb}$ always expands nodes with minimal lb (C1). In addition, $BHPA_{lb}$ terminate as soon as a solution with a cost $C \leq fmin_D = LB$ is found (C2). Finally, the allowable-set of $BHPA_{lb}$ contains all nodes with minimal lb since they all have the same f -value (C3). \square

5.2 BS*

BS* (Kwa 1989) expands a node with a minimal f -value from the smallest open-list (Pohl's cardinality criterion (Pohl 1971)) and terminates when the minimal f -value is *greater than or equal to* C^* . In addition, BS* trims nodes from the open lists if their f -value is *greater than or equal to* costs of potential solutions that were already found.


 Figure H.2: BS^*_{lb} is not well-behaved

Lemma H.4

BS^* and BS^*_{lb} are ill-behaved.

Proof. Consider problem instance I in Figure H.2. Using the heuristic values written inside the nodes, BS^* starts by expanding $start$ and $goal$ in an unspecified order. Nodes G_1 and S_2 get f -values of 2, and nodes S_1 an f -value of 5. Since BS^* has already found a solution of cost 4, nodes S_1 are trimmed from $Open_F$. At this point $Open_F$ contains only S_2 , while $Open_B$ contains two nodes (G_1). Thus, BS^* is forced to choose S_2 for expansion before terminating.

Next, consider BS^* using h_0 . The beginning of the search is similar: $start$ and $goal$ in an unspecified order. Then all nodes of $Open_F$ get an f -value of 1. Since no node is trimmed, $Open_F$ contains 3 nodes, while the $Open_B$ contains 2 nodes. Thus, BS^* expands the G_1 nodes before terminating, without expanding S_2 .

While applying the propagation changes the f -value of nodes, the behaviour of BS^*_{lb} is identical to BS^* on this example. Thus, both BS^* and BS^*_{lb} are ill-behaved. We note that C3 is violated here because the allowable-set of BS^*_{lb} is forced to contain only one open list. \square

Lemma H.5

BS^* is unreasonable.

Proof. The proof is similar to that of Lemma H.2. Consider the problem instance I in Figure H.1 assuming that $h_F(S_3) = h_B(G_3) = 0$. Similar to the proof of Lemma H.2, BS^* will have to expand $start, goal, \{S_1, S_2, S_3\}$ and $\{G_1, G_2, G_3\}$ before expanding A or C . Since $lb(S_3) = g_F(S_3) + g_B(C) + \epsilon = 5 > C^* = 4$, BS^* is unreasonable. \square

Lemma H.6

BS^*_{lb} is reasonable.

Proof. Since after the propagation the f -value of a node equals its lb , BS^*_{lb} always expands nodes with minimal lb (C1). Finally, BS^*_{lb} terminates as soon as a solution with a cost $c \leq fmin_D = LB$ is found (C2). Therefore, both conditions are satisfied and BS^*_{lb} is reasonable. \square

5.3 fMM

We have already shown that fMM is ill-behaved in Section 2.1. We now show that fMM is also unreasonable.

Lemma H.7

fMM is unreasonable.

Proof. Consider fMM^(1/4) applied to the problem instance in Figure H.3. After expanding *start* and *goal*, a solution of cost 11 is discovered, and $LB = lb(S_2, G_2) = g_F(S_2) + g_B(G_2) + \epsilon = 12$. Since $pr(G_2) = \max\{f_B(G_2), \frac{4}{3}g_B(G_2)\} = 10$, G_2 will be expanded before termination, even though $12 = LB > C^* = 11$. \square

Lemma H.8

fMM_{lb} always expands a node u with $lb(u) = LB$, hence it is reasonable.

Proof. Assume by contradiction that fMM_{lb} chose a node u in direction D for expansion s.t. $lb(u') \neq LB$. Therefore, there exists a pair of nodes (u', v') s.t. u' is in the $Open_D$, v' is in $Open_{\bar{D}}$ and $lb(u') = lb(v') = lb(u', v') = LB < lb(u)$. Using the lb -propagation, we know that $f_D(u') = lb(u') = lb(u', v')$, and $f_{\bar{D}}(v') = lb(v') = lb(u', v')$. Therefore, $f_D(u) = lb(u) > f_D(u')$. Likewise, $f_{\bar{D}}(u) = lb(u) > f_{\bar{D}}(v')$.

$$\begin{aligned} f_D(u) &> f_{\bar{D}}(v') = f_D(u') = lb(u') = lb(u', v') \\ &\geq g_D(u') + g_{\bar{D}}(v') + \epsilon \end{aligned}$$

Since u was chosen for expansion, we know that $pr_D(u) \leq pr_D(u')$ and $pr_D(u) \leq pr_{\bar{D}}(v')$. Thus,

$$\max(f_D(u), \frac{g_D(u)}{p} + \epsilon) \leq \max(f_D(u'), \frac{g_D(u')}{p} + \epsilon)$$

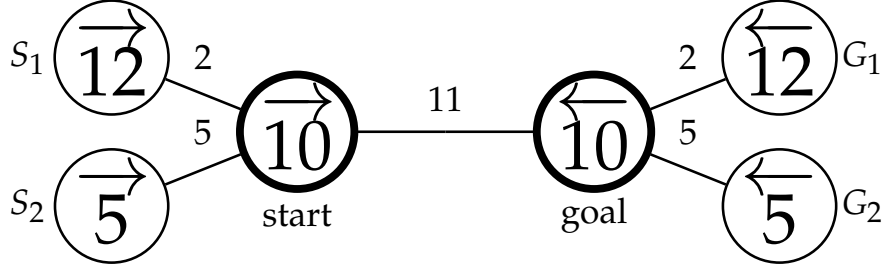


Figure H.3: fMM is not reasonable

and

$$\max(f_D(u), \frac{g_D(u)}{p} + \epsilon) \leq \max(f_{\bar{D}}(v'), \frac{g_{\bar{D}}(v')}{1-p} + \epsilon)$$

Since $f_D(u) = lb(u) > lb(u') = f_D(u') = lb(v') = f_{\bar{D}}(v')$,

$$f_D(u) \leq \frac{g_D(u')}{p} + \epsilon \quad (\text{H.1})$$

$$f_D(u) \leq \frac{g_{\bar{D}}(v')}{1-p} + \epsilon \quad (\text{H.2})$$

By summing inequality (H.1) multiplied by p with inequality (H.2) multiplied by $1 - p$, we get:

$$f_D(u) = pf_D(u) + (1-p)f_D(u) \leq g_D(u') + g_{\bar{D}}(v') + \epsilon$$

In contradiction to: $f_D(u) > g_D(u') + g_{\bar{D}}(v') + \epsilon$ above. \square

Lemma H.9

fMM_{lb} (with lb -propagation) is well-behaved.

Proof. Here we cannot use Theorem H.1 directly since the allowable-set of fMM_{lb} does not include every node u with $lb(u) = LB$, as some of these nodes might have g -values that raise their priority. Nonetheless, we show that fMM_{lb} using lb -propagation is in fact well-behaved, using a slight modification to Theorem H.1. In Lemma H.8 we showed that fMM_{lb} only expands nodes with minimal lb . In addition, fMM_{lb} terminates when the lowest f -value is $\geq C^*$. Since the f -value of nodes after applying lb -propagation is equal their lb -value, fMM_{lb} stops when $LB \geq C^*$. Thus, C1 and C3 are satisfied. However, C2 is violated by the priority mechanism of fMM, since nodes with the same lb -value might have different priorities due to their g -values and direction. For example, if there is only one node in $Open_F$ with $f_F = 3, g_F = 1$, and only one node in $Open_B$ with $f_B = 3, g_B = 2$, fMM will give lower priority to the node in

$Open_F$, based on his g -value (priority of 3 versus a priority of 5). Nonetheless, we will show that fMM_{lb} is still well-behaved. The problem arises since the first case of the proof of Theorem H.1 reduces to nodes n' with the $lb(n') = lb(n)$ and $pr(n') = pr(n)$. Therefore, nodes n' with $lb(n') = lb(n)$ and $pr(n') \neq pr(n)$ are part of the second case of the proof, which does not cover them. In that case, we considered some v s.t. $lb(n) = lb(v) = lb(n, v)$. Following that, it was clear that $v \notin S_2$. Nonetheless, in our case, v could have been in S_2 if it had a different priority than n when fMM_{lb} was running using h_1 . Since fMM_{lb} expands nodes with minimal priority, $pr(v) > pr(n)$. The priority of v could have been determined by one of the following options:

Case 1: $f_{\bar{D}}(v) = lb(v) = pr(v)$. However, $lb(u) \leq pr(u)$ and $lb(v) = lb(u)$. Therefore, $pr(v) \leq pr(u)$, by contradiction to the assumption that $pr(v) > pr(u)$.

Case 2: $\frac{g_{\bar{D}}(v)}{1-p} = pr(v)$. Since $f_{\bar{D}2}(v) \leq f_{\bar{D}1}(v)$ and $f_{\bar{D}}(v) \leq \frac{g_{\bar{D}}(v)}{1-p} = pr(v)$, we know that the $pr(v)$ using h_2 is less or equal than $pr(v)$ using h_1 . In addition, the priority of node n (and any of its ancestors) when running using h_1 must be strictly less than $pr(v)$, by contradiction to the fact that v was already chosen for expansion.

Therefore, we can conclude that v is still not in S_2 and the proof of Theorem H.1 is generalized to fMM as well. \square

5.4 NBS and DVCBS

NBS (Chen et al. 2017) and DVCBS (Shperberg et al. 2019) are two prominent Bi-HS algorithms that choose nodes for expansion with minimal lb . At any point in the search, NBS chooses a pair of nodes with minimal lb and expands them both. DVCBS expands nodes from a subset of those with minimal lb , determined by maintaining a dynamic version of the G_{MX} (denoted by DG_{MX}) and finding its MVC. Both algorithms terminate as soon as a solution with a cost $c \leq LB$ is found. Since the expansion policy and termination condition of both algorithm already consider LB , their properties remain unaffected by lb -propagation.

Clearly, NBS and DVCBS satisfy conditions C1, and C2 and are therefore reasonable (up to a single additional expansion). However as previously mentioned, the allowable-set of DVCBS includes only nodes that make up the MVC of DG_{MX} , violating C3. In addition, once NBS has chosen a pair (u, v) for expansion, it is committed to expanding both nodes. Therefore, after expanding u , any node u' in the same direc-

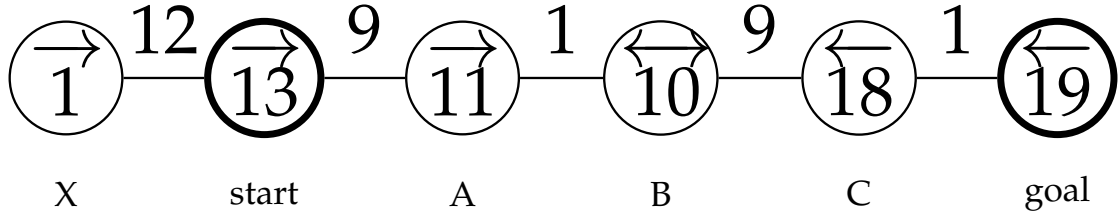


Figure H.4: NBS is not well-behaved

tion of u s.t. $lb(u') = lb(u) = lb(v) = LB$ is not in the allowable set of NBS until after expanding v , in violation of condition C3.

Lemma H.10

NBS is ill-behaved.

Proof. Consider the problem instance of Figure H.4.³ NBS using h_0 will start by expanding *start* and *goal*. Afterwards, *X* and *A* are added to $Open_F$, and *C* is added to $Open_B$. Since $g_F(x) = f_F(x) = 12$, $lb(X, C) = 12$, while $lb(A, C) = 9$. Therefore, the pair (A, C) is chosen for expansion, after which a path of length 20 has been discovered. Since $lb(X, B) \geq 20$ and $lb(B, B) \geq 20$, NBS terminates after expanding *start*, *A*, *C*, and *goal*.

NBS using the heuristic in the nodes starts by expanding *start* and *goal*. Next, *X* and *A* are added to $Open_F$, and *C* is added to $Open_B$. Since $lb(A, C) = 20$ (due to $f_F(A)$) and $lb(X, C) = 19$ (due to $f_B(C)$). Therefore, NBS will expand the pair (X, C) , despite the fact that *X* was not expanded using a weaker heuristic. \square

Lemma H.11

DVCBS is ill-behaved.

Proof. Consider the problem instance of Figure H.2. DVCBS using the heuristic in the nodes starts by expanding either *start* or *goal* since both of them are MVCs of DG_{MX} . If *goal* was chosen, *start* must be expanded, since it becomes the only MVC, followed by S_2 for a similar reason, after which DVCBS terminates since $LB = 5 > 4$ (the path that was discovered from *start* to *goal*). Likewise, if *start* was chosen for expansion, DVCBS will expand either S_1 and terminate, or *goal* followed by S_1 . In both cases the G_1 nodes are never expanded. However, DVCBS using h_0 must expand *start* and *goal* in an unspecified order, followed by G_1 . \square

³This example is due to Robert Holte and Sandra Zilles

5.5 GBFHS

GBFHS is an algorithm that iteratively increases the depth of the search ($fLim$). At each depth, a pre-defined *split function* (parameter of the algorithm) is used that determines how deep to search on each side at each iteration by splitting $fLim$ to $gLim_F$ and $gLim_B$ s.t. $fLim = gLim_F + gLim_B + \epsilon - 1$. At every iteration, GBFHS considers nodes for expansion in direction D with $f \leq fLim$ and $g < gLim_D$. GBFHS terminates when as soon as a solution with a cost equals to $fLim$ is found.

GBFHS was proved to be well-behaved.⁴ We show that GBFHS is reasonable by showing that it expands only nodes with minimal lb , even without applying lb -propagation.

Lemma H.12

GBFHS is reasonable.

Proof. Since GBFHS considers nodes for expansion in direction D with $f \leq fLim$ and $g < gLim_D$, a node u that is chosen for expansion will have $lb(u) \leq \max\{gLim_F + gLim_B + \epsilon - 1, fLim\} = flim$ ⁵, and since the $flim$ is increased only after there are no nodes left for expansion, $lb(u) = flim = LB$. Ergo, GBFHS expands nodes with minimal lb . In addition, GBFHS terminates as soon as a solution of cost $flim = LB$ is found. Thus, C1 and C2 are satisfied and GBFHS is reasonable. \square

6 Experimental results

We ran experiments on three domains: **(1) 50 10-Pancake Puzzle** instances with the GAP heuristic (Helmert 2010). To get a range of heuristic strengths, we also used the GAP- n heuristics (for $n = 1 \dots 9$) where the n smallest pancakes are deleted from the heuristic computation; **(2) 50 instances of the 10-disk 4-peg Towers of Hanoi (TOH4)** problem with (8+2) and (6+4) additive PDBs (Felner et al. 2004). **(3) Grid-based pathfinding**: 65 maps from Dragon Age Origins (DAO) (Sturtevant 2012), each with different start and goal points (a total of 1,680 instances).

⁴Even though well-behavedness was not defined in a general manner when GBFHS was created, the proof of Barley et al. 2018 is still applicable to the new definition with slight modifications.

⁵Barley et al. 2018 implicitly assume that ϵ is an integer ≥ 1 .

6. Experimental results

	10-Pancake								TOH-10				Grid	
Algorithm	GAP-0		GAP-1		GAP-2		GAP-3		8+2		6+4		DAO	
	h	h_{lb}	h	h_{lb}	h	h_{lb}	h	h_{lb}	h	h_{lb}	h	h_{lb}	h	h_{lb}
BPHA-Alt	26	26	674	665	9,484	6,916	50,804	14,564	26,435	23,666	96,102	69,130	368	319
BPHA-Min	25	21	465	427	6,375	5,615	34,497	28,127	33,770	13,270	159,079	49,128	413	309
BS*	25	25	374	682	5,528	5,585	30,687	11,957	18,268	18,351	73,434	63,918	311	496
fMM($1/4$)	103	115	5,348	1,985	30,858	11,030	82,396	27,097	22,660	19,899	65,364	57,453	414	407
MM	264	76	2,519	682	5,944	1,684	5,034	2,040	41,407	34,307	89,883	76,852	511	501
fMM($3/4$)	64	81	2,098	1,111	15,424	6,002	48,227	13,263	42,452	36,933	173,968	158,290	442	434

Table H.2: Experimental results of average node expansions across domains

Figure H.5 shows the average number of nodes expanded by MM and by MM_{lb} in the 10-pancake domain across all GAP heuristics. Clearly, adding the lb -propagation significantly reduces the number of nodes expanded. Using h_{lb} seems to reduce the number of node expansions for each of the GAP heuristics up until GAP-7, in which the heuristic effectively becomes h_0 . In addition, this figure clearly demonstrates the anomaly of MM; the average number of nodes expanded by MM using heuristics GAP-2 through GAP-6 is greater than the number of nodes expanded by using heuristics GAP-7 through GAP-9 (notice the “hump-in-the-middle” (Barley et al. 2018)). By contrast, the hump-in-the-middle of MM_{lb} is much smaller, and in fact not visible when considering the average number of expansions. However, there were still some individual problem instances in which MM_{lb} expanded fewer nodes using a weaker heuristic. This is consistent with Theorem H.1, since we are using a predetermined tie-breaking policy and not the best possible tie-breaking policy for every instance. Interestingly, MM_{lb} using $\epsilon = 0$ demonstrates no hump-in-the-middle, even when con-

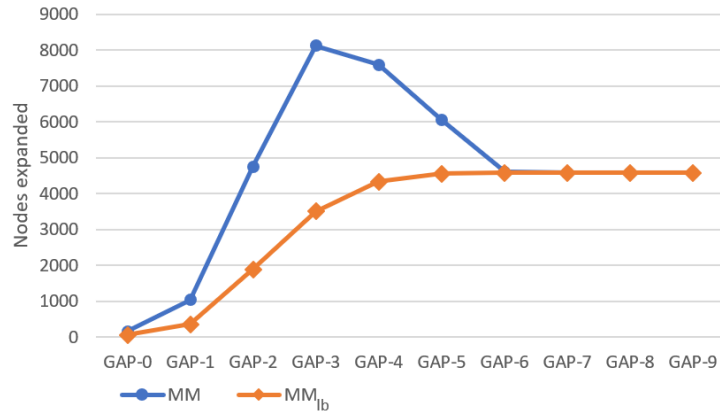


Figure H.5: MM vs. MM_{lb} on 10-pancake

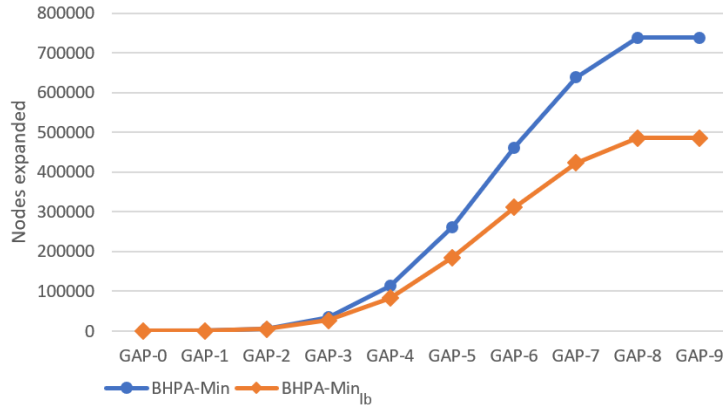


Figure H.6: BHPA-Min vs. BHPA-Min_{lb} on 10-pancake

sidering individual problem instances.

Similarly, Figure H.6 shows the average number of nodes expanded in the 10-pancake domain across all GAP heuristics, with $\epsilon = 1$ by a variant of BHPA denoted by BHPA-Min. BHPA-Min selects the frontier that includes the node with the minimal f -value. Here too, the lb -propagation improves the search by reducing the number of nodes expanded. Even though BHPA-Min is well-behaved, and demonstrates no hump-in-the-middle in the average case, the lb -propagation still improves the algorithm by making it reasonable. This improvement is more evident with GAP-8 and GAP-9; Despite these GAP heuristics behaving like h_0 in the 10-pancake domain, lb -propagation incorporates $gmin_F + gmin_B + \epsilon$ into the f -values of nodes in BHPA-Min, exposing an additional termination condition, and allowing the search process to halt sooner.

The average number of node expanded across domains, using $\epsilon = 1$, appear in Table H.2. There is one row for each algorithm, and one column for each of the domains and their heuristics; h denotes the original heuristic, while h_{lb} denotes the heuristic enhanced by lb -propagation. The algorithms we have tested are BS^* , $fMM(p)$ using $p \in \{1/4, 1/2, 3/4\}$, BHPA-Min and BHPA-Alt, another variant of BHPA that alternates between the frontiers between expansions. The results show that using the lb -propagation reduces the number of node expansions in most cases by up to a factor of 4. The lb -propagation particularly excels when the heuristics are weak. In these cases using h_{lb} always results in fewer node expansions; this is also the case for GAP-4 through GAP-9, which do not appear in the table. Another interesting observation is

that the hump-in-the-middle is less pronounced in all tested algorithms. BS^* seems to be the least affected by the propagation among all algorithms. We posit that the reason for this is that BS^* is highly dependent on the search process; since BS^* selects a node for expansion from the direction with the smallest open list, minor increments in heuristic values might cause nodes to be trimmed away, possibly changing the size balance of the two frontiers. BS^* also assumes that the heuristic is consistent, which other algorithms do not. We have also experimented using $\epsilon = 0$; the results are similar to the reported $\epsilon = 1$ results.

Naturally, maintaining and using lb for each node incurs overheads. In the domains we used, g - h -bucketing requires negligible time and space. However, we did not focus on code optimization, and used naive data-structures. Thus, run times are not reported here. Improving efficiency with a bucketing scheme or adapting the data structures used for efficient lb computations by NBS is reserved for future work.

7 Discussion

We have examined the source of the anomaly exhibited by some Bi-HS algorithms, where using a better heuristic causes the algorithm to expand more nodes. Aiming to improve some algorithms in which the anomaly manifests, the properties of “well-behavedness” and “reasonableness” were defined, and sufficient conditions (C1, C2, C3) for these properties were established. These properties provide insights that lead to the lower-bound propagation scheme (lb -propagation) that can be added to many existing Bi-HS algorithms, in some cases bestowing upon them these desirable properties. Empirical results show that modified algorithms exhibit better behavior, alleviating or even eliminating the undesirable “hump-in-the-middle” effect seen when an algorithm is run with heuristics of varying quality.

The well-behavedness property as defined in this paper ensures that there exists a tie-breaking policy for which the anomaly would not occur. However, the desired tie-breaking policy is not specified. It is a non-trivial issue, left for future research, to define conditions that guarantee a stronger well-behavedness property of an algorithm, such that a dominating heuristic would *never* cause more nodes to be expanded than the weaker heuristic using the **same** tie-breaking policy. Another interesting re-

search direction is to re-examine the three well-behavedness conditions with respect to heuristics that are strictly dominating, i.e., $h_1 > h_2$.

Acknowledgements

This work was supported by Israel Science Foundation (ISF) grant #844/17 to Ariel Felner and Eyal Shimony, by BSF grant #2017692, by NSF grant #1815660 and by the Frankel center for CS at BGU.

References of Paper H

- [Bar+18] Michael W. Barley et al. “GBFHS: A Generalized Breadth-First Heuristic Search Algorithm”. In: *SoCS*. 2018, pp. 28–36.
- [Bur+12] Ethan Andrew Burns et al. “Implementing Fast Heuristic Search Code”. In: *SoCS*. 2012.
- [Cha83] Dennis de Champeaux. “Bidirectional Heuristic Search Again”. In: *J. ACM* 30.1 (1983), pp. 22–32.
- [Che+17] Jingwei Chen et al. “Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions”. In: *IJCAI*. ijcai.org, 2017, pp. 489–495.
- [CS77] Dennis de Champeaux and Lenie Sint. “An Improved Bidirectional Heuristic Search Algorithm”. In: *J. ACM* 24.2 (1977), pp. 177–191.
- [DP85] Rina Dechter and Judea Pearl. “Generalized Best-First Search Strategies and the Optimality of A*”. In: *J. ACM* 32.3 (1985), pp. 505–536.
- [Eck+17] Jurgen Eckerle et al. “Sufficient Conditions for Node Expansion in Bidirectional Heuristic Search”. In: *ICAPS*. 2017, pp. 79–87.
- [FKH04] Ariel Felner, Richard E. Korf, and Sarit Hanan. “Additive Pattern Database Heuristics”. In: *J. Artif. Intell. Res.* 22 (2004), pp. 279–318. doi: 10.1613/jair.1480. URL: <https://doi.org/10.1613/jair.1480>.
- [GFS16] Daniel Gilon, Ariel Felner, and Roni Stern. “Dynamic Potential Search - A New Bounded Suboptimal Search”. In: *SoCS*. 2016, pp. 36–44.
- [Hel10] Malte Helmert. “Landmark Heuristics for the Pancake Problem”. In: *SoCS*. 2010.
- [Hol+17] Robert C. Holte et al. “MM: A bidirectional search algorithm that is guaranteed to meet in the middle”. In: *Artif. Intell.* 252 (2017), pp. 232–266.
- [Hol10] Robert C. Holte. “Common Misconceptions Concerning Heuristic Search”. In: *SoCS*. Ed. by Ariel Felner and Nathan R. Sturtevant. AAAI Press, 2010. URL: <http://aaai.org/ocs/index.php/SOCS/SOCS10/paper/view/2073>.

- [KK97] Hermann Kaindl and Gerhard Kainz. “Bidirectional Heuristic Search Reconsidered”. In: *J. Artificial Intelligence Resesearch (JAIR)* 7 (1997), pp. 283–317.
- [Kwa89] James B. H. Kwa. “BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm”. In: *Artif. Intell.* 38.1 (1989), pp. 95–109.
- [Poh71] Ira Pohl. “Bi-directional search”. In: *Machine intelligence* 6 (1971), pp. 127–140.
- [RN16] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Malaysia; Pearson Education Limited, 2016.
- [Sha+16] Guni Sharon et al. “Extended Abstract: An Improved Priority Function for Bidirectional Heuristic Search”. In: *SoCS*. 2016, pp. 139–140.
- [Sha+17] Eshed Shaham, Ariel Felner, Jingwei Chen, et al. “The Minimal Set of States that Must Be Expanded in a Front-to-End Bidirectional Search”. In: *SoCS*. 2017, pp. 82–90.
- [Sha+18] Eshed Shaham, Ariel Felner, Nathan R. Sturtevant, et al. “Minimizing Node Expansions in Bidirectional Search with Consistent Heuristics”. In: *SOCS*. 2018, pp. 81–98.
- [Shp+19] Shahaf Shperberg et al. “Enriching Non-parametric Bidirectional Search Algorithms”. In: *AAAI*. 2019.
- [Stu12] Nathan R. Sturtevant. “Benchmarks for Grid-Based Pathfinding”. In: *IEEE Trans. Comput. Intellig. and AI in Games* 4.2 (2012), pp. 144–148. DOI: 10.1109/TCIAIG.2012.2197681. URL: <https://doi.org/10.1109/TCIAIG.2012.2197681>.

Discussion

This work involves the definition, development, and solution of meta-level problems for several prominent settings. Since there is no single global metareasoning scheme that can be directly applied to all problem settings, an effort should be invested in designing a meta-level problem tailored to each setting. However, as this work demonstrates, the effort is often rewarded with improvement of the overall performance.

In papers A and B, the focus was the selection problem, in which an item from a set of unknown utilities needs to be selected. In this problem, measurements of item values prior to selection are allowed, each measurement comes at a cost. The aim is to find a subset of items to measure in order to maximize the expected utility, i.e., the utility of the selected item minus the cost of the measurements. In Paper B, we examined theoretical properties of VOI for the selection problem, and identified cases of submodularity and supermodularity in which the VOI can be efficiently approximated. In addition, we have proposed a greedy algorithm which efficiently chooses a batch of items to measure. In Paper A the selection problem was used as a meta-level problem for node-selection in MCTS. We have defined Batch Value of Perfect Information (BVPI) as a generalization of the Value of Perfect Information (VPI) of Russell and Wefald 1991 for game trees, when considering information gathered for a set of nodes (items) instead of individual nodes. Using BVPI, we have developed a selection-scheme in order to select batches of nodes which are likely to provide meaningful information when measured (i.e., expanded and sampled). This selection scheme was implemented in several domains and empirically outperformed existing selection schemes.

A promising future research direction is to adapt the VOI node-selection schemes to cases where a neural network (NN), rather than simulations, provides the value estimations (e.g., AlphaZero Silver et al. 2017). These schemes traditionally use node-

selection methods which are based on variants of UCT, which still try to (incorrectly) minimize cumulative regret.

Papers C, D, and E involve situated temporal planning. In situated temporal planning, the planner has to account for time that passes during planning, as some of the actions expire during the search for a plan. We considered a case where the aim is to find a plan on time, regardless of the plan's cost. We also considered a case in which the aim is to minimize the expected cost (potentially at the risk of not finding a plan at all). For each case we have defined a metareasoning problem that aims to find a CPU time allocation schedule to processes, each of which has a distribution over the time they need to complete their execution, and over deadlines by which they need to terminate. We have proposed optimal pseudo-polynomial algorithms for the case of known deadlines when aiming to maximize the probability that at least one process would complete its execution on time, and to the case where there is only one running process (and n completed plans) and the goal is to minimize solution cost. For the general case of each problem setting, we proposed an effective greedy scheme which balances between the utility gain per time-unit of a process and its urgency (which is modeled by the damage in utility rate by delaying its execution). The greedy schemes were evaluated on synthetic data using known families of distributions, as well as on a distribution constructed by running the OPTIC planner on the RCLL domain. In addition, our DDA greedy scheme was integrated into the OPTIC planner. The version of OPTIC that was infused by DDA was able to solve more problems on average.

In the abstract model, we assume the performance profiles of the different processes to be independent of one another. However, if this assumption is relaxed and there could be some dependency between the performance profiles of different processes, then the information obtained by executing one process would affect the other processes. Nevertheless, having dependencies significantly complicates the model as such dependencies introduce an additional value of information. For example, an optimal solution in this case could be to allocate time units to a process which is very unlikely to lead to a valid solution, if the resulted time allocation would provide more accurate information for the other processes. One option to tackle these dependencies is to have a belief over performance profiles which can be updated by time allocations, but this POMDP is much more challenging to solve than the original

MDP, and the current greedy schemes are not likely to be good approximations for this case.

There are still many challenges in making situated temporal planning practical. First, there is still room to improve the different approximation methods used in this work, e.g., the quality of the performance profiles based on the search information, improving the runtime of the metareasoning, and even finding better greedy time allocation schemes. Another key challenge is to relax the Independence assumption between performance profiles. In the context of the motivating problem of situated temporal planning, processes are partial plans. Therefore, it is very likely that the performance profiles of processes obtained from similar partial plans would be dependent. Nevertheless, having dependencies significantly complicates the model, as such dependencies introduce value of information. For example, an optimal solution in the case of dependent performance profiles could be to allocate time units to a process which is very unlikely to lead to a valid solution, if the resulting time allocation would provide more accurate information for the other processes. One option to tackle these dependencies is to have a belief over performance profiles which can be updated by allocating time to dependant processes, but this POMDP is much more challenging to solve than the original MDP, and the current greedy schemes are not likely to be good approximations for this case. Finally, a promising line of research is to extend the metareasoning techniques to interleave planning and execution, a setting in which one is allowed to start executing actions before having a complete plan. In the standard planning paradigms, a complete plan is required in order to begin execution. However, realistic schemes allow for interleaving planning and execution, wherein actions can be executed even during the search for a plan is complete in order to increase the chances of finding and executing a plan on time, or even to find plans with lower costs.

Paper F was motivated by the AI-birds competition for autonomously playing the game of angry birds. In this paper we have considered the problem of selecting pairs of optimization problem instances and algorithms in order to maximize the expected score under a time limit. We have analyzed the selection problem under the assumption that the performance profiles (distributions) of the algorithms on the problem instances ("levels") is known. Even under this simplifying assumption, the

selection problem is NP-hard. Nonetheless, we have proposed a pseudo-polynomial method to solve this problem, as well as an efficient greedy scheme which is based on choosing a pair of problem instance and algorithm that maximize the expected improvement in score per time unit. On top of being very fast to compute, the greedy scheme was efficient in an empirical evaluation that used data collected by running the different agents on many angry-birds level. Thus, this improved-rate greedy scheme was chosen for our meta-agent. Since the assumptions of known performance profiles is clearly wrong when playing unseen levels, we have used a learning scheme which attempts to learn a distribution over performance profiles. This distribution over distribution is updated (using a Bayesian model) whenever a new observation (a result obtained by applying an agent to one of the levels) is made. The improved-rate greedy algorithm combined with the learning scheme was implemented as a meta-agent that uses agents which have previously participated in AI-birds competitions. This meta-agent outperformed all individual agents when evaluated on unseen game levels from past competitions.

Paper G and H have made some contributions to the field of bidirectional heuristic search (Bi-HS). Paper G introduced the DVCBS algorithm, which maintains an abstract graph called a DG_{MX} and finds a minimal vertex cover of this graph in order to guide the search. The process of constricting a DG_{MX} and finding its minimal vertex cover takes time. However, it is possible to use the information obtained from a DG_{MX} to expand multiple nodes. As in all meta-level problems, there is a tradeoff between computation time and getting better information (by computing the DG_{MX} often, or even after every expansion). Our experiments showed that expanding a cluster of nodes for every DG_{MX} computation provides a good balance which enabled DVCBS to empirically outperform existing algorithms in terms of time and node expansions. Paper H has shown that the lower-bound on solution cost, which can be computed using a DG_{MX} can be used for making heuristic functions more accurate. This method of heuristic enhancement was applied to existing algorithms in order to improve their performance, and even granting some of them the new desirable properties of being well-behaved and reasonable.

An important take-home message that is evident from this work is that existing algorithms are able to adapt to new settings by considering a suitable meta-level prob-

lem that controls their computational actions. While most meta-level problems are (at least) NP-hard, greedy algorithms tend to provide a good balance between the required computation time and the solution quality. In several applications the greedy approach of maximizing utility gain per time unit was empirically successful. In addition, many meta-level problems are based on performance profiles, or distributions. The standard methods for obtaining these distributions are to assume that the distribution belong to a known family of distributions, to collected offline statistics and use them as a prior distribution, or to estimate the distributions using online measures (e.g., one-step error). An interesting future research direction would be to incorporate reinforcement learning (RL) methods, and in particular distributional reinforcement learning (DRL, e.g., Dabney et al. 2018) methods, to learn these distributions. For example, for AlphaZero-like applications, instead of using a standard RL method that returns a single value-estimation for each node, a DRL method could be used in order to return a value-distribution estimation. By having value-distribution estimations instead of single-value estimations, the VOI-based schemes can be applied. I believe that a successful combination of learning methods, proven to be rewarding in many cases, with search and planning techniques, is the key to improve the decision-making abilities of AI systems. Metareasoning could very well be a methodological way to bridge the gap between these two approaches.

References

- [Bro+12] Cameron Browne et al. “A Survey of Monte Carlo Tree Search Methods”. In: *IEEE AI Games* 4.1 (2012), pp. 1–43.
- [Cas+18] Michael Cashmore et al. “Temporal Planning while the Clock Ticks”. In: *ICAPS*. AAAI Press, 2018, pp. 39–46.
- [Che+17] Jingwei Chen et al. “Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions”. In: *IJCAI*. ijcai.org, 2017, pp. 489–495.
- [Che86] Christopher Cherniak. *Minimal Rationality*. Computational models of cognition and perception. MIT Press, 1986.
- [Col15] Andrew M Colman. *A dictionary of psychology*. Oxford quick reference, 2015.
- [Dab+18] Will Dabney et al. “Distributional Reinforcement Learning With Quantile Regression”. In: *AAAI*. AAAI Press, 2018, pp. 2892–2901.
- [DP85] Rina Dechter and Judea Pearl. “Generalized Best-First Search Strategies and the Optimality of A*”. In: *J. ACM* 32.3 (1985), pp. 505–536.
- [Eck+17] Jurgen Eckerle et al. “Sufficient Conditions for Node Expansion in Bidirectional Heuristic Search”. In: *ICAPS*. 2017, pp. 79–87.
- [FD14] Zohar Feldman and Carmel Domshlak. “Simple Regret Optimization in Online Planning for Markov Decision Processes”. In: *J. Artif. Intell. Res.* 51 (2014), pp. 165–205.
- [Goo71] Irving John Good. “Twenty-seven principles of rationality”. In: *Foundations of statistical inference* (1971), pp. 108–141.
- [How66] Ronald A Howard. “Information value theory”. In: *IEEE Transactions on systems science and cybernetics* 2.1 (1966), pp. 22–26.
- [KS06] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *ECML*. Vol. 4212. Lecture Notes in Computer Science. Springer, 2006, pp. 282–293.

References

- [Ont+13] Santiago Ontañón et al. “A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft”. In: *IEEE Trans. Comput. Intell. AI Games* 5.4 (2013), pp. 293–311.
- [RW91] Stuart Jonathan Russell and Eric Wefald. *Do the right thing: studies in limited rationality*. MIT press, 1991.
- [Shp+19a] Shahaf S. Shperberg, Andrew Coles, Bence Cserna, et al. “Allocating Planning Effort When Actions Expire”. In: *AAAI*. AAAI Press, 2019, pp. 2371–2378.
- [Shp+19b] Shahaf S. Shperberg, Ariel Felner, Solomon Eyal Shimony, et al. “Improving Bidirectional Heuristic Search by Bounds Propagation”. In: *SOCS*. AAAI Press, 2019, pp. 106–114.
- [Shp+19c] Shahaf S. Shperberg, Ariel Felner, Nathan R. Sturtevant, et al. “Enriching Non-Parametric Bidirectional Search Algorithms”. In: *AAAI*. AAAI Press, 2019, pp. 2379–2386.
- [Shp+20] Shahaf S. Shperberg, Andrew Coles, Erez Karpas, Solomon Eyal Shimony, et al. “Trading Plan Cost for Timeliness in Situated Temporal Planning”. In: *IJCAI*. ijcai.org, 2020, pp. 4176–4182.
- [Shp+21] Shahaf S. Shperberg, Andrew Coles, Erez Karpas, Wheeler Ruml, et al. “Situated Temporal Planning Using Deadline-aware Metareasoning”. In: *ICAPS*. 2021.
- [Sil+17] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nat.* 550.7676 (2017), pp. 354–359.
- [SS17] Shahaf S. Shperberg and Solomon Eyal Shimony. “Some Properties of Batch Value of Information in the Selection Problem”. In: *J. Artif. Intell. Res.* 58 (2017), pp. 777–796. DOI: 10.1613/jair.5288. URL: <https://doi.org/10.1613/jair.5288>.
- [SSF17] Shahaf S. Shperberg, Solomon Eyal Shimony, and Ariel Felner. “Monte-Carlo Tree Search using Batch Value of Perfect Information”. In: *UAI*. AUAI Press, 2017.

- [SSY19] Shahaf S. Shperberg, Solomon Eyal Shimony, and Avinoam Yehezkel. “Algorithm Selection in Optimization and Application to Angry Birds”. In: *ICAPS*. AAAI Press, 2019, pp. 437–445.
- [TS12] David Tolpin and Solomon Eyal Shimony. “MCTS Based on Simple Regret”. In: *AAAI*. AAAI Press, 2012.

פיתרון לבעיה (למשל קוביה הונגרית מסודרת במלואה). מכיוון שעץ החיפוש ממדל הבעיה הנפתרת באופן מהימן, מסלולים בעץ המובילים מהמצב ההתחלתי למצב היעד מהווים רצף החלטות שניתן לקבל בעולם האמיתי בכדי לפתור את הבעיה בפועל. ישנן הגדרות שונות תחתיהן אלגוריתמי חיפוש פועלים, אך כל האלגוריתמים חולקים את האתגר החשוב ביותר, כיצד להשקיע את המאמץ החישובי (למשל איזה חלק מעץ החיפוש לסרוק) על מנת למצוא פתרונות (רצף של החלטות) טובים במהירות.

מטא-היסק הוא רעיון מרכזי בהיקשר של קבלת החלטות אשר נמצא בתפר בין אנושות לאינטליגנציה. הרעיון הזה הוא שאפשר להרוויח הרבה על ידי חשיבה (היסק) על תהליך החשיבה עצמו. בהקשר של בעיות חיפוש ותכנון, מטא-היסק מתייחס לקבלת החלטות על צעדי החישוב של האלגוריתמים (ולא ישירות על הבעיה עצמה). על מנת לקבל החלטת אלו, מתבצעת השוואה של "עלותם" של צעדי חישוב (למשל במונחים של משאבי חישוב כגון זמן או זיכרון) אל מול תרומתם הצפויה לקידום החיפוש לקראת פתרון. כדי ליישם את הרעיון של מטא-היסק יש להגדיר ולפתור בעיות ברמת מטא (מטא-בעיות) עבור תשתיות אלגוריתמיות ספציפיות. ככלל, קשה לפתור מטא-בעיות, ופתרון אף עלול להיות מסובך יותר מהבעיה המקורית אותה מנסים לפתור. עם זאת, אפילו קירוב מהיר לפתרון של מטא-בעיות יכול להניב תוצאות טובות ולשפר את האלגוריתמים עליהם הן מיושמות.

עבודת גמר זו מתמקדת בפיתוח והערכה של שיטות מטא-היסק שפותחו עבור בעיות מסוגים שונים על מנת לשפר מגוון רחב של אלגוריתמי חיפוש, תכנון ותזמון.

תקציר

קבלת החלטות היא היבט מהותי בבינה מלאכותית. היכולת לקבל החלטות הגיוניות ולנמקן הינה הכרחית להשגת כל רמה של אוטונומיה. ישנם יישומי בינה מלאכותית רבים הנמצאים בשימוש יום-יומי המבוססים על קבלת החלטות, בין יישומים אלו נמנים יישומי ניתוב ומציאת דרכים, ניהול מחסנים אוטומטיים (למשל אלו של אמזון), טיפולים רפואיים, מכוניות עצמוניות (אוטונומיות), ייצור (כיפוף מתכות, פסי הרכבת של צוללות, מטוסים ומכוניות וכו'), מערכות קידוח נפט חכמות, עוזרים וירטואלים (סירי, אלקסה וכד'), אבטחת סייבר, הרכב שירותי רשת, חקר חלל (למשל רוברים שנשלחו למאדים), יצירת סרטי אנימציה, משחקים, ועוד יישומים רבים נוספים. עבור הבעיות המעניינות ביותר, אי אפשר לשקול את כל ההחלטות האפשריות ואת ההשלכות שלהן תוך פרק זמן מעשי, במיוחד במקרים שבהם החלטה אחת משפיעה על החלטות עוקבות (תהליכי החלטה). עם זאת, יש חשיבות רבה להצליח לפתור בעיות מעניינות אלו. לכן, נוהג נפוץ הוא הגבלת הזמן הזמין לקבלת החלטה. כתוצאה מכך, לאלגוריתמים שמטרתם לפתור בעיות אלו יש זמן מוגבל מאוד לחישוב שלאחריו הם צריכים לקבל את ההחלטה שלדעתם תביא לתוצאה הטובה ביותר. קבלת החלטות טובות יותר תשפר פתרונות עבור מגוון רחב של יישומים. לפיכך, אחת המטרות המרכזיות בבינה מלאכותית היא שיפור האופן בו אלגוריתמי בינה מלאכותית מנצלים את הזמן לחישוב לצורך קבלת החלטות טובות יותר.

אלגוריתמי חיפוש ותכנון פותרים בעיות קבלת החלטות על ידי התבוננות קדימה באמצעות מידול דרכי פעולה אפשריות עתידיות. מידול זה בנוי כעץ הידוע בשם עץ חיפוש. כל קדקוד בעץ חיפוש מייצג את מצב העולם בהקשר של הבעיה אותה מנסים לפתור, וכן שורשו של עץ חיפוש מייצג את המצב ההתחלתי של הבעיה. הבנים של כל קדקוד בעץ הם כל המצבים אליהם ניתן להגיע ממנו ע"י ביצוע פעולה אחת (השקולה לקבלת החלטה אחת). לדוגמה, בקוביה הונגרית מצב הינו הצבע של כל אחד מ-36 הריבועים השונים (9 ריבועים בכל פאה) ופעולה היא סיבוב של הקוביה באחד הכיוונים. אלגוריתמי חיפוש סורקים את עץ החיפוש (מחפשים) במטרה למצוא מצב יעד המהווה

הצהרת תלמיד המחקר עם הגשת עבודת הדוקטור לשיפוט

אני החתום מטה מצהיר/ה בזאת :

הצהרת תלמיד המחקר עם הגשת עבודת הדוקטור לשיפוט

אני החתום מטה מצהיר/ה בזאת : (אנא סמן) :

X חיברתי את חיבורי בעצמי, להוציא עזרת ההדרכה שקיבלתי מאת מנחה/ים.

X החומר המדעי הנכלל בעבודה זו הינו פרי מחקרי מתקופת היותי תלמיד/ת מחקר.

X בעבודה נכלל חומר מחקרי שהוא פרי שיתוף עם אחרים, למעט עזרה טכנית שאינה כוללת ניתוח תוצאות, הנהוגה בעבודה ניסיונית. לפי כך מצורפת בזאת הצהרה על תרומתי ותרומת שותפי למחקר, שאושרה על ידם ומוגשת בהסכמתם.

____ תזה במתכונת אסופת מאמרים, כוללת מאמרים בהם אני מחבר ראשון משותף (equal contribution). במקרה זה, נא לצרף הצהרה של המחבר השותף על חלקו בפרסום, ואישורו לכך, שמאמר זה לא יוכל להיכלל בתזת מאמרים נוספת, שהוא יגיש.

תאריך: 05/8/2021 שם התלמיד/ה: שחף שפרברג חתימה Shelef S. Shperberg

העבודה נעשתה בהדרכת

פרופ' אייל שמעוני

במחלקה למדעי המחשב


מטא-שיטות לפתרון בעיות תכנון, חיפוש ותזמון

מחקר לשם מילוי חלקי של הדרישות לקבלת תואר "דוקטור לפילוסופיה"

מאת

שחף שפרברג

הוגש לסנאט אוניברסיטת בן גוריון בנגב



אישור המנחה

אישור דיקן בית הספר ללימודי מחקר מתקדמים ע"ש קרייטמן

05/08/2021

כ"ז באב

באר שבע

מטא-שיטות לפתרון בעיות תכנון, חיפוש ותזמון

מחקר לשם מילוי חלקי של הדרישות לקבלת תואר "דוקטור לפילוסופיה"

מאת

שחף שפרברג

הוגש לסנאט אוניברסיטת בן גוריון בנגב

05/08/2021

כ"ז באב

באר שבע