Phase Transitions in Planning Problems: Design and Analysis of Parameterized Families of Hard Planning Problems

Itay Hen and Eleanor G. Rieffel and Minh Do and Davide Venturelli
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

There are two common ways to evaluate algorithms: performance on benchmark problems derived from real applications and analysis of performance on parametrized families of problems. The two approaches complement each other, each having its advantages and disadvantages. The planning community has concentrated on the first approach, with few ways of generating parametrized families of hard problems known prior to this work.

Our group's main interest is in comparing approaches to solving planning problems using a novel type of computational device - a quantum annealer - to existing state-of-the-art planning algorithms. Because only small-scale quantum annealers are available, we must compare on small problem sizes. Small problems are primarily useful for comparison only if they are instances of parametrized families of problems for which scaling analysis can be done.

In this technical report, we discuss our approach to the generation of hard planning problems from classes of well-studied NP-complete problems that map naturally to planning problems or to aspects of planning problems that many practical planning problems share. These problem classes exhibit a phase transition between easy-to-solve and easy-to-show-unsolvable planning problems. The parametrized families of hard planning problems lie at the phase transition. The exponential scaling of hardness with problem size is apparent in these families even at very small problem sizes, thus enabling us to characterize even very small problems as hard. The families we developed will prove generally useful to the planning community in analyzing the performance of planning algorithms, providing a complementary approach to existing evaluation methods. We illustrate the hardness of these problems and their scaling with results on four state-of-the-art planners, observing significant differences between these planners on these problem families. Finally, we describe two general, and quite different, mappings of planning problems to QUBOs, the form of input required for a quantum annealing machine such as the D-Wave II.

1 Introduction

The construction of efficient general-purpose planners has been the holy grail for the planning community since the inception of the field. Our team is interested in exploring the potential of quantum annealing to provide effective solutions to difficult planning problems. Because of the limitations of current quantum annealers, with the state-of-the-art devices having only 512 qubits and limited connections between them, only small problems can be run. To compare quantum annealing to state-of-the-art classical planners, we need to evaluate performance on parametrized sets of problems. Because new approaches are most valuable when they can attack the most challenging problems, we are interested in analyzing scaling behavior and comparing performance on small instances of planning problems within parametrized classes of planning problems that are believed to be hard.

Currently, the most common approach to designing benchmark planning problems is to extract solvable problems from real-world applications, then simplify them into a format that fits within different subsets of the standard planning modeling language PDDL. For all existing benchmarks, problems setups are mostly selected randomly with linear size increase. Thus, there is currently no systematic way to ensure an exponential, or higher, increase in problem hardness with linear size increase.

This approach has the benefit of tuning algorithms toward the applications from which the problems are obtained. There are certain drawbacks to this approach. (1) It is hard to explain what makes a domain hard or easy for certain planners. (2) It is hard to predict the performance of a given planner on new domains with novel structure. (3) there are polynomial-time algorithms for certain domains [Hoffmann, 2005]. And (4) it only contains known solvable problems, which is not the case in most real-world applications.

A complementary approach is to design parametrized families of planning problems that contain instances that can be shown to be intrinsically hard in the typical case. This approach has certain advantages. (1) It supports analysis as to which types of planning problems, and which aspects of these problems, are hard or easy for certain planners and why. (2)

Figure 1: Examples of NASA applications utilizing planning and scheduling technology: controlling DS-1 spacecraft, ISS solar panel control, Mars Rover navigation, and scheduling for SOFIA.

It is meaningful to say that even quite small problems, ones which can can be solved quickly by many planners, are hard because they are part of a hard family. (3) Analysis of the behavior of planners on these small problems can highlight the strengths and weaknesses of various planners and planning algorithms, give insight into the reasons for these strengths and weaknesses, and enable prediction of the performance of these algorithms on much larger instances. However, to date, as we discuss in Section 2, parametrized families of hard planning problems have been hard to find, and few were known prior to this work.

Our aim is to establish parameterized sets of planning problems that (1) are intrinsically hard, and therefore challenging for all planners and types of planning algorithms; (2) are controllable by parameters that enable establishing hardness not simply through the sheer number of objects; (3) have explainable reasons for their hardness; (4) contain structure that exist in planning and scheduling domains of practical importance.

Here we propose generating parametrized families of hard planning problems inspired by graph-theoretic problems that are known to be hard. The parametrized families of planning problems we designed fall into two classes: the first class contains navigation-style problems, the second class contains task scheduling problems. Many real-world applications of planning have aspects of both navigation and scheduling.

For many NP-complete problems, phase transitions from almost always solvable to almost always unsolvable have been observed, with the transition becoming sharper as the size of the problems increases. It has also been observed that the hardest problems tend to lie at the phase transition threshold [Cheeseman *et al.*, 1991]. Some NP-complete problems naturally lend themselves to planning problems. Here, we derive planning problems from well-studied graph-theoretic problems for which phase transition results are known, and for which the hardness of the problems at the threshold has been confirmed in empirical studies. The navigation-style families of planning problems are derived from Hamiltonian path problems. The scheduling-style families of planning problems from graph coloring problems.

To summarize, our novel contributions include: (1) analyzing benchmarks derived from previous combinatorial problems exhibiting phase transition instead of arbitrary problems generated from simplified real-world applications.[1] (2) we investigate both solvable and unsolvable problems, which is more realistic. (3) unlike previous investigation of phase-transition in planning, we use contemporary state-of-the-art planners that have been winning most recent International Planning Competitions. Our extensive empirical results with multiple planners, which utilize vastly different planning techniques, confirm the phase-transition in multiple domains.

We first review related work in Section 2, and then provide an overview of classical planning in Section 3. We then describe our parametrized families of planning problems, navigation-based planning problems in Section 4 and scheduling-based planning problems in Section 5. Section 7 contains the result of our evaluation of different planners on these planning problems, confirming their difficulty, examining scaling behavior, and providing a comparison between planners. Section 8 describes two different approaches to mapping planning problems to QUBO form, one of the

[1]Note that the selective combinatorial problems that we investigate are the core aspects of many real-world applications

main steps toward running these problems on a quantum annealing device. In Section 9, we conclude.

2 Related work

Phase transitions in combinatorial optimization problems have long been recognized [Huberman and Hogg, 1987], with the hardest problem instances generally found at these phase transitions [Cheeseman *et al.*, 1991; Selman *et al.*, 1996]. Typically, these problems exhibit an easy-hard-easy behavior with problems on one side of the phase transition being easy to solve, problems on the other side being easy to show unsolvable, and problems on the phase transition either hard to solve or hard-to-show unsolvable. For this reason, finding phase transition behavior is a common strategy for identifying hard problem instances.

Bylander [Bylander, 1996] investigated phase transitions in planning by generating random planning problems from a sampling of random graphs. By changing the ratio of actions to state variables, he succeeded in finding a control parameter such that the probability of a random instance having a plan was almost zero for small values (too few actions) to a probability of almost 1 of having a plan for large values (too many actions). In this pioneering work, Bylander used very simple algorithms, which are far from the sophisticated algorithms used in current state-of-the-art planners, to show the easiness of determining the solvability or unsolvability far from the phase-transition region. He did not investigate the computational difficulty in the phase transition region on realistic planning algorithms. Bylander's work also does not draw from the key differences between random graphs and state-space transition graphs in planning such as directedness and the relation between the binary state variables and the planning-state caused by the fact that planning actions change multiple state-variables. Since his work did not take into account these differences, the planning-state transition graph generated by random actions were non-uniform. Thus, by starting from a random graph instead of a random transition graph, the generated problems are not as random as they meant to be.

Slaney & Thiebaux [Slaney and Thiebaux, 1998] investigated phase transition for both optimization and decision problems using the popular planning benchmark domain: BlocksWorld. Using a depth-bounded depth-first search on a tree, their empirical evaluation exhibited a phase-transition region at certain ratios of the number of towers in the goal configuration to the total number of blocks. While the results were collected on random graphs resembling planning transition graphs, the fact that it uses a particular search strategy customized to this domain means that it does not truly resemble algorithms employed by current state-of-the-art planners, and so the result is not conclusive.

Subsequently, Rintanen [Rintanen, 2004] improved upon the earlier work by Bylander [Bylander, 1996]. First, Rintanen concentrated the analysis on the difficult problems within the phase-transition region (instead of the easy regions). Second, instead of mimicking the way random problems are generated at the phase transition in SAT, Rintanen introduced two alternative models with additional restrictions to eliminate the most trivially unsolvable instances. Unlike the previous work, Rintanen used state-of-the-art planners utilizing different planning algorithms for the empirical evaluation. The results show the easy-hard-easy behavior for the FF planner, but is not conclusive for the other two planners (LPG and SP). While the problems generated are better than those in Bylander's pioneering work, there is no theoretical explanation for the hardness of problems in the phase-transition region and also the problems generated do not represent any typical class of existing planning and scheduling benchmark domains.

Rintanen [Rintanen, 2012a] generated parametrized families of hard planning instances that are different from previous work [Bylander, 1996; Rintanen, 2004] on several fronts. First, given that nearly all existing planning benchmarks, including all used in all International Planning Competitions (IPC), contain only solvable problems, Rintanen generated only proven solvable instances. Second, instead of originating from random graphs, the hardness of the instances in this family is based on controlling the number of directed paths from the initial state to the goal states; thus affecting the probability that forward-state-space planners find a valid path among all available paths. The empirical evaluation showed that such instances are indeed hard for two state-of-the-art forward-state-space planners, LAMA [Richter and Westphal, 2010] and HSP [Bonet *et al.*, 1997], while they are much easier for the non-directional SAT-based planners M and Mp. Unlike this work, our parametrized families target all planners, not just forward-state-space planners. Our families also are based on well-known combinatorial problems with structures that are essential in many planning applications. Furthermore, as shown in our results section, our problems are much harder at small instance sizes.

Porco et al. [Porco *et al.*, 2011] developed a tool for translating NP-complete problems into planning problems, specifically STRIPS fragments. They applied their tool to the directed Hamiltonian path problem and the graph coloring problem, among others, and evaluated the performance of the M planner on these problems. While our work, like theirs, involves translating NP-complete problems into planning problems, it differs significantly from their work. Their aim was to build a general purpose tool to translate NP-complete problems generally into planning problems. To do so, they first translate NP-complete problems into logical $SO\exists$ sentences, second order existential sentences. Their tool then translates $SO\exists$ sentences into PDDL. Because our translations are direct, rather than going through second order logic, they are more efficient. Porco et al. compared the performance of the M planner on a variety of different problems derived from NP-complete problems. They were interested in the performance in general, not just on hard problems, so they did not look at the performance specifically at the phase transition for these problems. Moreover, their interest was in understanding the performance of a single planner, while we were interested in understanding the relative strengths and weaknesses of multiple planners on the hardest planning problems. In [Rintanen, 2012b], Rintanen builds on the work of Porco et al., and compares the performance of a number of different planners on planning problems translated from NP-complete problems, though not necessarily in the phase tran-

sition region. He compared the performance of different planners on a large number of other instances, including planning-competition benchmark problems.

Taking a different approach from trying to generate hard problems, Hoffman [Hoffmann, 2005] looks at existing benchmarks to identify the characteristics of those domains that make it hard for forward-state-space planners in general and the FF planner [Hoffmann and Nebel, 2001] in particular. In our work, we do not try to analyze existing benchmarks, rather create new benchmarks that are hard for all planning algorithms, not only forward state-space planners.

3 An Overview of the Classical Planning Formalism

Classical planning problems are expressed in terms of binary *state variable* (sometimes called predicates) and *actions* (sometimes called operators). Examples of state variables in the rover domain are "Rover R is in location X" and "Rover R has a soil sample from location X," which may be true or false. Actions consist of two lists,

- a set of *preconditions* and
- a set of *effects* (or postconditions).

In classical planning, it is conventional that the preconditions for an action must be positive, so the set of preconditions is a subset of state variables that must be set to true in order for the action to be possible to carry out. The effects of an action consists of a subset of state variables with the values they take on if the action is carried out. For example, the action "Rover R moves from location X to location Y" has one precondition, "Rover R is in location X = true" and has two effects "Rover R is in location X = false" and "Rover R is in location Y = true."

A specific planning problem specifies an *initial state*, with values specified for all state variables, and a *goal*, specified values for one or more state variables. As for preconditions, goals are conventionally positive, so the specified value for the goal variables is true. Generally, the goal specifies values for only a small subset of the state variables. A plan is a sequence of actions. A valid plan, or a solution to the planning problem, is a sequence of actions $A_1, A_2, ..., A_L$ such that the state at time step t_{i-1} meets the preconditions for action A_i, the effects of action A_i are reflected in the state at time step t_i, and the state at the end has all of the goal variables set to true.

4 Parameterized Families of Navigation-Style Planning Problems

In the planning problems inspired by the directed Hamiltonian path (DHP) problems and the undirected Hamiltonian path (UHP) problems, there is an action associated with each node of the graph, the action of visiting that node. The preconditions and effects of an action corresponding to a node are determined by edges (and non-edges) between that node and the other nodes in the graph. An edge originating from one node and ending in another corresponds to allowing the action corresponding to the second node to follow the action corresponding to the first node.

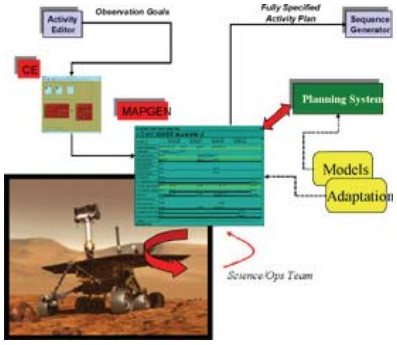


Figure 3: MAPGEN software, a joint effort between Ames and JPL, was used for the Mars Exploration Rover mission

NASA Applications: Navigation is a critical component in many of NASA's planning applications [Chien *et al.*, 2012] and many existing planning benchmarks. Indeed, some of NASA's most visible missions in recent time, the Mars rover missions such as Curiosity (and Spirit and Opportunity before it) require extensive mission planning [Bresina *et al.*, 2005; Aghevli *et al.*, 2007]. Inspired by mission planning activities to support autonomous rovers on Mars, the Moon, and other planets, NASA Ames scientists created the Mars Rover planning benchmark domain [Long and Fox, 2003]. It captures the essential components of NASA's rover navigation problem and it has been part of multiple recent International Planning Competitions.

In the rover navigation domain, the rover is dropped at a given location on a planet such as Mars. Given a list of locations of interest that it needs to visit to, say, take picture or analyze samples, the planner needs to find the rover route that makes optimal use of resources, such as time and power, and satisfies multiple constraints. Given that each location is only likely visited once, and that one often wishes to minimize the traveling distance and time, the high-level navigation problem is similar to the Hamilton Path problem that we are investigating.

4.1 Background on graph traversability problems

The parametrized families of navigation-type planning problems are based on directed Hamiltonian path (DHP) and undirected Hamiltonian path (UHP) problems, both graph-theoretic NP-hard problems.

4.2 Planning problems from directed Hamiltonian path (DHP)

Given a random graph $G(V, E)$ with n nodes and a set of directed edges E, a planning problem instance based on the directed Hamiltonian path (DHP) problem on this graph may be formulated as follows. The DHP domain contains n actions, each with three associated variables: a goal variable, an 'internal' variable, and an 'external' variable. Altogether there will be $3n$ state variables consisting of n goals, n internal variables, and n external variables.

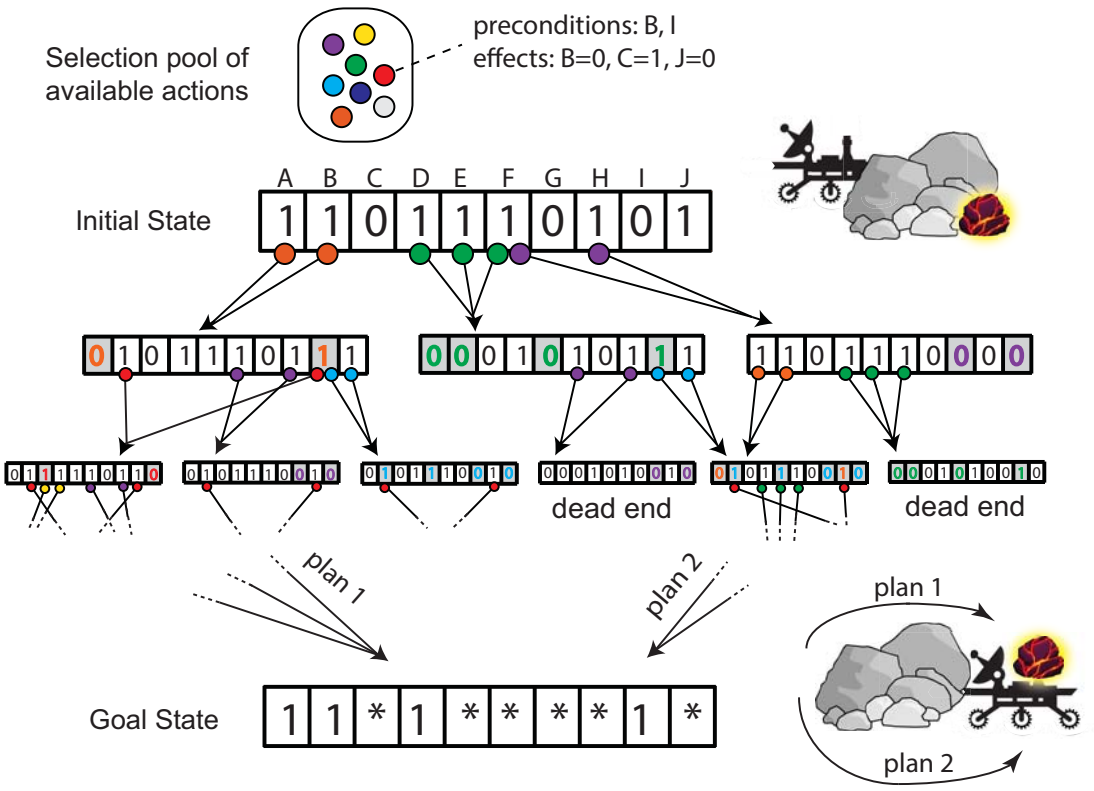


Figure 2: Pictorial view of a planning problem. The initial state (e.g. Rover to the left behind the rocks, without payload) is specified by assigning True (1) or False (0) to state variables (named A-J in this oversimplified example). The planning software navigates a tree, where a path represents a sequence (with possible repetitions) of actions selected from a pool (colors). Each action has preconditions on the state variables (e.g. moves can be done around the rocks and not through) which need to be satisfied in order for the actions to be executed (the circles under the state variables in the search tree needs to match True=1) and has an effect on the state (colored variables in shaded regions of the new state have changed values). A valid search plan (multiple valid plans are possible) will reach the goal state (e.g. Rover in front of the rocks to the right, with a sample collected).

The action for a given site, corresponds to the action of going to that site. The 'internal' variable for the action corresponding to a site can only be set by that action, and it is used to indicate that the site has been visited. The 'external' variable for the action corresponding to a site can be set by actions corresponding to other sites, specifically ones connected by edges to this site. In this way the action corresponding to a given site can only take place after an action corresponding to visiting one of its neighbors, which is how it should be. We now capture this intuitive explanation more formally in terms of preconditions and effects.

Each action has 2 preconditions: its internal variable must be 1, which indicates that this action has not been used in the plan already, and its external variable must be 1, indicating that this action can follow the previous action.

Each action A also has $n + 1$ effects. It sets its own goal variable to 1, to indicate that the corresponding node of the graph has been visited, and sets its own internal variables to 0, thus excluding the action from appearing twice in the plan. It also sets each of the $n - 1$ external variables, one for each of the other actions A', according to whether there is an edge from the vertex corresponding to A to the vertex corresponding to A', indicating whether A' can follow A or not.

An observant reader will notice that the internal variable for a site always has the opposite value to the goal variable for the site, and may wonder why there is this duplication. Recall from Section 3 the convention that preconditions must be positive, as must the achievement of goals. To enable both to be positive, we must use two variables.

In the initial state of the planning problem instance, all of the goal variables have value zero while all external and internal variables have value 1, thus any of the n actions can be performed at the start. A valid plan is a sequence of the n actions that corresponds to a path along the edges that visits all nodes exactly once.

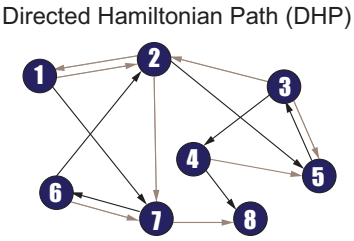
We obtain a parametrized family of DHP-based planning problems, parametrized by n and p, by randomly generating graphs with n vertices and, for any ordered pair of vertices, including the directed edge from the first vertex to the second with probability p, and then deriving planning problems as

described in the preceding paragraphs.

4.3 Planning problems from undirected Hamiltonian path (UHP)

Planning problems are derived from undirected Hamiltonian path problems in a similar way to those derived from directed Hamiltonian path problems. The undirected graph in a Hamiltonian path problem can be viewed as a directed graph by replacing each edge between two nodes with two edges between those two nodes, one in each direction. The corresponding planning problem can then be derived as in Section 4.2. The undirected case has a symmetry that does not exist in the directed case: if action A can follow action A', the action A' can follow action A.

We obtain a parametrized family of DHP-based planning problems, parametrized by n and p, by randomly generating graphs with n vertices and, for any pair of vertices, including the edge between the two vertices with probability p, and then deriving planning problems as described in the preceding paragraph.



	Node	1	2	3	4	5	6	7	8
	External	1	1	1	1	1	1	1	1
	Goal	0	0	0	0	0	0	0	0
1st Action	External	0	1	0	0	0	0	1	0
	Goal	1	0	0	0	0	0	0	0
2nd Action	External	0	0	0	0	0	1	0	1
	Goal	1	0	0	0	0	0	1	0
3rd Action	External	0	1	0	0	0	0	1	0
	Goal	1	0	0	0	0	1	1	0
4th Action	External	1	0	0	0	1	0	1	0
	Goal	1	1	0	0	0	1	1	0
5th Action	External	0	0	1	0	0	0	0	0
	Goal	1	1	0	0	1	1	1	0
6th Action	External	0	1	0	1	1	0	0	0
	Goal	1	1	1	0	1	1	1	0
7th Action	External	0	0	0	0	1	0	0	1
	Goal	1	1	1	1	1	1	1	0
8th Action	External	0	0	0	0	0	0	0	0
	Goal	1	1	1	1	1	1	1	1

End of plan

Figure 4: Example of a DHP problem with 8 sites. Black arrows represent the edges which are chosen for the solution of the problem. The table illustrates the binary representation of the corresponding plan from the choice of the initial action (node 1) to the end of the plan. Note that the internal variables are not shown for brevity, their values being opposite to that of the respective goal variables. (See text.)

5 Parametrized families of scheduling-type planning problems

Most planning applications include scheduling aspects. Scheduling, which deals with assigning resources and time to tasks while taking into account constraints, is in itself an important problem. A number of scheduling problem correspond to graph coloring problems. For this type of scheduling problem, the objective is to schedule activities so that they can all completed in the shortest amount of time. Activities that require the same resource, a specific machine or a human expert, cannot be scheduled at the same time. Time-slots can be thought of as colors, and activities are vertices that need to be assigned to different time-slots. Edges in the graph represent pairs of activities that compete for the same resource and therefore cannot be allocated to the same time-slot.

NASA Applications: Most NASA applications of planning include a scheduling component [Chien *et al.*, 2012]. The types of scheduling decision that graph-coloring resembles permeate different space planning applications such as MEXAR [Oddi and Policella, 2007], Crew Planning [Marquez *et al.*, 2010], and DSN [Johnston *et al.*, 2009]. One example is the on-going Habitat Automation project investigated at NASA Ames [Morris *et al.*, 2013]. In this application, the planning software suite (SPIFe + EUROPA) assists in planning and re-planning crew activities in which multiple crew members are assigned different daily tasks. Each activity can be done by a given crew member (or a subset of crew members).

5.1 Background on graph coloring

Graph coloring is a well-known and well-researched problem in graph theory. It is a special case of graph labeling that requires assigning color labels to elements of the graphs according to certain constraints. In this paper, we concentrate on the *vertex coloring* problem in which the goal is to assign each vertex a color label so that any two vertices connected by an edge do not have the same color.

Most important decision and optimization problems related to graph-coloring are computationally difficult. For example:

- Decision problem: for a given graph G and an integer number k, deciding if G can be properly colored with k colors is $O(n2^n)$ [Bjorklund *et al.*, 2009] and is NP-complete [Dailey, 1980].
- Optimization problem: finding the smallest number of colors needed to color a given graph G (chromatic number) is NP-hard [Bjorklund *et al.*, 2009].
- Counting problem: counting the number of ways a graph G can be colored with k colors is $O(n2^n)$ and is #P-complete.

It is NP-hard to color a 3-colorable graph with 4 colors [Guruswami and Khanna, 2000] and the 3-coloring problem remains NP-complete even on planar graphs of degree 4 [Dailey, 1980].

5.2 Planning problems from Graph Coloring (GC)

In the cleanest form, a scheduling problem S with a set of tasks T, each task requires a certain time-slot, and there are constraints that any pair of tasks $\{t_1, t_2\}$ competing for the same resource cannot be assigned the same time-slot can be mapped to a vertex coloring problem for the graph $G(V, E)$ as follow:

- Each task $t \in T$ is represented as a vertex $v \in V$.
- Each time-slot is represented by a color.

- Each pair of tasks $\{t_1, t_2\}$ competing for a resource is represented by an edge $e = \{v_1, v_2\}$ with v_1, v_2 representing t_1, t_2 accordingly.

The chromatic number of G (smallest number of colors needed to color G) represents the smallest number of timeslots needed for S, thus representing the minimum makespan for S.

Given an undirected graph $G = \{V, E\}$ with a set of vertices V and a set E of edges generated as described by randomly assigning an edge between any two vertices with probability p, the planning problem representing coloring G with k colors is represented as follows.

For each vertex v, there are $k + 1$ associated binary variables: $colored(v)$ representing whether or not v is already colored and k variables $colored(v, c)$ representing if v is colored with color c.

There are $M = |V| \times k$ actions[2] a; each one represents coloring a given vertex v with a color c. Let $C(v)$ be the set of vertices that are connected with v. There are $|C(v)| + 1$ preconditions:

- One precondition representing that v is not already colored (i.e., $colored(v) = false$).
- $|C(v)|$ preconditions representing that any vertex $v_i \in C(v)$ is not already colored in c (i.e., $colored(v_i, c) = false$).

a has two effects:

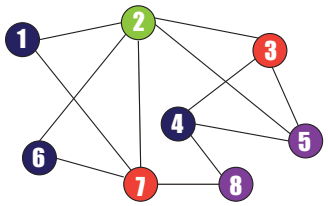
- One negative (delete) effect indicates that v is no longer "not colored": $colored(v) = true$.
- One positive (add) effect indicates that v is now colored in c: $colored(v, c) = true$.

The initial state represents the fact that none of the vertices is colored: $\forall v_i \in V : colored(v_i) = false$.

The goal state requires that all vertices are colored: $\forall v_i \in V : colored(v_i) = true$.

For each random graph, the plan is a sequence of $|V|$ actions, each to color a given vertex $v_i \in V$.

Graph Vertex Coloring (GC)



	Node	1	2	3	4	5	6	7	8
Initial State	Colored	0	0	0	0	0	0	0	0
	Red	0	0	0	0	0	0	0	0
	Blue	0	0	0	0	0	0	0	0
	Green	0	0	0	0	0	0	0	0
	Violet	0	0	0	0	0	0	0	0

	Node	1	2	3	4	5	6	7	8
Goal State	Colored	1	1	1	1	1	1	1	1
	Red	0	0	1	0	0	0	1	0
	Blue	1	0	0	1	0	1	0	0
	Green	0	1	0	0	0	0	0	0
	Violet	0	0	0	0	1	0	0	1

Figure 5: Example of a GC problem with 8 sites. The vertex have been colored according to a possible solution with k=4. The table indicates the initial state and the illustrated goal state. (Note that coloring actions might be executed simultaneously as long as they are not in conflict; see later.)

6 Empirical Evaluation Methods

We evaluated our families of planning problems by:

1. Generating a large number of random instances for each particular set of values for the parameters;
2. Generating the PDDL representation (i.e., pair of domain and problem files) for each instance; and
3. Running a representative set of planners using the PDDL files and collect the running times and other relevant information. Each run has a 7200-second cutoff time. Because we are using a cutoff, we report median, rather than mean, runtimes.

All results were collected using a 64-bit RedHat Linux machine with 8 Intel Core I7 cores running at 2.4 Ghz with 8 GB of RAM.

6.1 Planners used

To get representative results, we sought to use a set of planners that: (1) use different planning algorithms; and (2) are considered state-of-the-art (that have shown strong performance on the existing benchmarks). Specifically, we use the following planners[3]. Table 1 contains a summary of some of the features of these planners.

FF: FF [Hoffmann and Nebel, 2001] is one of the most influential planners that populated the current crop of forward-state-space planners that have won the last several planning competitions. FF search gradually builds a plan starting from the initial state by adding one action at a time. It uses the "relaxed-plan" heuristic based on ignoring the delete list of all actions. Using this heuristic, FF's default search algorithm is *enforced hill-climbing* and it switches over to a complete breadth-first search upon getting stuck. Given that FF is a complete planner (upon switching to BFS) and is guaranteed to find a solution if time and memory limits permit, it is our main planner used in many setups.

LPG: LPG [Gerevini *et al.*, 2003] won several tracks at the 3rd and 4th IPCs and is generally considered one of the most flexible and high-performing planner. Unlike most other planners employing the systematic-search framework, LPG uses a unique local-search algorithm that operates on an "action-graph" structure. Specifically, it builds a graph structure with each node represents an action in a partial plan (initially set with a "relaxed plan" found by FF) with multiple ordering constraints to limit the flexibility of the graph. In each search step, LPG will try to either (1) remove an action from the graph; or (2) add an action to the graph (to

[2] In the PDDL representation, the domain file uses $|V|$ action templates with color c as parameters and thus do not represent $|V| \times k$ (ground) actions directly.

[3] We generally had to make small adjustments to each planner to output the information we need, especially when the test problems are not solvable (which is not the case with existing benchmarks).

	FF	LPG	M	Mp
Heuristic search-based	**Y**	**Y**	n	n
SAT	n	n	**Y**	**Y**
Complete	**Y**	**Y**	n	n
Local search	n	**Y**	n	n
Greedy search	**Y**	**Y**	n	n
Off-the-shelf SAT techniques	n	n	**Y**	n

Table 1: Summary of planner features.

support another action or goal); or (3) establish a causal link between two actions already in the graph (i.e., specifying that one action is supporting the other). This process is driven by the heuristic of greedily reducing the number of unsupported goals and actions (with some random non-optimal moves allowed). Like other local-search approaches, this algorithm is not complete and LPG can switch to FF after a pre-defined amount of effort.

M & Mp: M and Mp [Rintanen, 2012b] are the best performing SAT-based planners and are representative of the compilation-based approach. Unlike other planners in this category that utilize off-the-shelf SAT solvers, M and Mp employs several techniques to boost their overall performance such as unique mutual-exclusion rules to lower the encoding horizon (and thus the overall size of the SAT encoding) and customized SAT heuristics to take advantage of the structures in the SAT encoding caused by the planning constraints.

While both M and Mp use the same SAT encoding, they differ in the heuristic used to guide the SAT solver. Specifically, M employs the most popular techniques used in current best-performance state-of-the-art SAT solvers: conflict-driven clause learning algorithm (CDCL) with the VSIDS (Variable State Independent, Decaying Sum) heuristic. Specifically, CDCL (1) selects an unassigned variable; (2) applies unit propagation; (3) builds the implication graph; (4) if there is any conflict then analyzes and non-chronologically backtracks to the appropriate decision level. The VSIDS heuristic: (1) initializes a counter of each variable to be 0; (2) when a clause is added to the clause database, increments the counter associated with each literal in the clause; (3) chases the unassigned literal with highest counter at each decision point (with random tie-breaking); and (3) all counters are divided by a constant, periodically. On the other hand, Mp uses a new way of choosing the decision variables specific to planning: it (1) utilizes goal ordering to order SAT variables representing goals; (2) gives higher priority to variables representing actions supporting goals and orders them based on how constrained those actions are; and (3) makes the solving process less directional by randomly choosing between a fixed set of candidate actions. In short, M uses general-purpose SAT heuristics while Mp uses planning-specific heuristics in choosing the next SAT decision variable.

6.2 Problem generation

The test sets of navigation-style planning problems were generated using a simple C++ program written from scratch. To generate a test set of scheduling-type planning problems, we extended the graph generator program provided by Culberson et al. [Culberson *et al.*, 1995]. This generator can generate different types of graph controlled by various parameters. We extended the generator so that it outputs PDDL files containing the specification of planning problems derived from these graphs (See Section 5) as well as the graphs in the standard DIMACS format. In both cases, we generate random graphs $G_{n,p}$ with n vertices in which for each pair of vertices, the edge between them is included in the graph with probability p for a variety of values of n and p. The resulting graphs have a distribution of number of edges, unlike $G_{n,m}$ graphs which have a fixed number of edges, but the vertices which they connect are chosen randomly.

7 Results and Discussion

In this section, we present the results and analysis of the performance of several different planners on the different planning domains whose generation was described above. We specifically focus on the phase transitions of these domains, which is where the problems are expected to be the hardest to solve. The order parameters for the phase transitions studied here are the same as those of the original graph problems.

To appreciate the hardness of the parametrized families of planning domains described above, we first establish the existence of a solvable/unsolvable phase transition in each of these families. As the connectivity parameter p is varied, we see a transition from almost all unsolvable to almost all solvable instances, with this transition becoming more and more abrupt as the size of the problem increases. We then confirm that the problems in this transition region are the ones that are most challenging to tackle, taking the most time to solve or to show unsolvable, with easy to solve and easy-to-show-unsolvable on either side. We used the FF planner to determine the probability of solvability for all of our problem classes. Similarly, we used the FF planner to confirm the difficulty of the problems in the phase transition region, graphing the median runtime as a function of the hardness parameter. (As we mentioned before, we use the median runtime rather than the mean because we have a time cutoff.)

We then examine the scaling behavior of all four planners as problem size n increases and p is varied as a function of n to stay right on the phase transition. We obtain this order parameter from the literature on the underlying problem. The exponential growth of the median runtime for all planners confirms the intrinsic difficulty of the problem instances of the tested domains at the phase transition. The absolute time and the slope of the exponential enable us to compare the efficacy of the planners on these problems.

The resolution of the runtime was 0.01 seconds. For this reason, we do not show results for problems so small that all problems are solved in less than 0.01 seconds. For this reason, some of the graphs will appear cut-off on the left. Also, since we used a 7200 second (2 hour) cutoff for the planners, once the median runtime was above 7200, we no longer show the results.

Each data point shows the average over testing 50 random instances for the relevant parameters. The error bars show the

35 percentile to 65 percentile confidence interval. The scaling parameter p had a precision of six decimal places.

7.1 Results on DHP-inspired planning problems

Figure 6 confirms the phase transition from unsolvable to solvable as the connectivity p is increased in the planning problems inspired by directed Hamiltonian path problems. The phase transition is sharp already at problem size $n = 40$, where n is the number of actions in the planning problem.

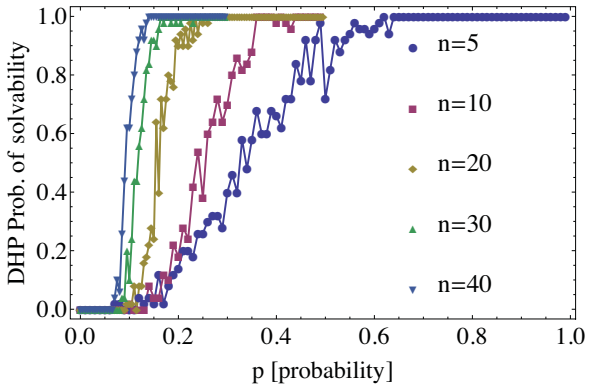


Figure 6: Fraction of solvable instances as a function of the connectivity parameter p for several different problem sizes n directed Hamiltonian path (DHP) planning problems. The phase transition is sharp even for problem sizes as small as $n = 40$.

Figure 7 shows that the runtime increases sharply in the transition region even for small problem sizes. The results shown are for the FF planner.

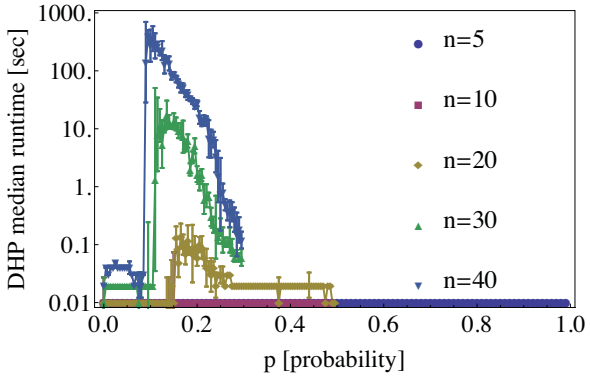


Figure 7: Typical runtime as a function of the connectivity parameter p for several different sizes of directed Hamiltonian path (DHP) planning problems. Around the phase transition, the typical runtime shoots up markedly even at small problem sizes. The results shown are for the FF planner.

Figure 8 shows the runtime of four different planners on problems at the phase transition, where scaling parameter $p = (\log n + \log\log n)/n$. The expected exponential scaling of difficulty, as measured by runtime, with the problem size is seen clearly. The location of the phase transition at the value of the scaling parameter $p = (\log n + \log\log n)/n$ has been established for the closely related undirected Hamiltonian cycle problem in [Komlós and Szemerédi, 1983; Cheeseman *et al.*, 1991]. A simple argument suggests this scaling. Let $\gamma = 2m/n$ be the typical vertex degree, where m is the number of edges. For our construction, the number of edges is roughly $pn^2/2$, so $\gamma = pn$. The typical distribution of vertex degrees will be Poisson distributed, with the probability of a vertex having degree k being

$$p_k = \frac{1}{k!}\gamma^k e^{-\gamma}.$$

If there are any vertices of degree 0, then the graph does not have a Hamiltonian path. The expected number of isolated vertices is $p_0 n = e^{-\gamma} n$. To ensure that this number is less than 1, we need γ to scale as $\log n$.

The FF and LPG perform best, significantly outperforming both M and Mp on these navigation-type planning problems. For the smallest problem sizes, FF performs best, but as the problem sizes increase, LPG soon passes FF. While both M and Mp perform significantly worse than FF and LPG on these small problems, with Mp performing considerably better than M. At larger problem sizes it looks ready to overtake FF, though it appears that its slope is greater than that of LPG, 0.23 rather than 0.15, so LPG appears to be the best planner overall.

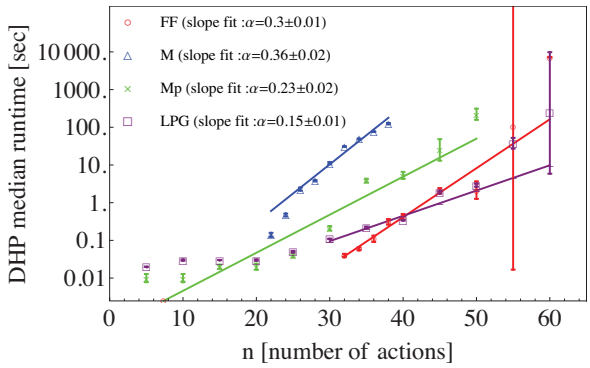


Figure 8: Typical runtime vs problem size at the phase transition, $p = (\log n + \log\log n)/n$, for directed Hamiltonian path (DHP) problems for different planners. The exponential scaling of runtime with problem size is evident for all planners. The exponential coefficient α is given in each case. The FF planner significantly outperforms the M planner on these problems.

Planners Comparison Analysis: Actions in the planning domains for the DHP and UHP problem families are strongly "sequential" in the sense that: (1) each action of visiting a site C enables exactly the set of actions corresponding to visiting other sites that are connected to C by an edge; and (2) there are prevalent mutual-exclusion relations between actions: we cannot visit two cities in parallel. This type of constraint is known to put compilation-based planners such as M and Mp at a disadvantage [Kautz and Selman, 1999]. Because they need to bound the planning horizon to some value h to create a SAT encoding, and then solve incrementally for higher h until the solution is found, this type of domain may require M and Mp to go through multiple unsolvable encodings until

it reaches an h value that is solvable. Moreover, when the problem is not solvable, it is also harder for M and Mp to discover that no matter how high the value of h, there is no solution. FF, on the other hand, can switch to a complete breadth-first-search algorithm that will exhaustively search until the depth level of n to return the correct answer.

However, this type of domain is also does not fit well with FF's heuristic. Given that each solution is of exactly the same length n, when FF explores a given search node X that is reached by a actions from the initial state, all children of X will either have the same distance $n - a$ to the goals or is a "dead end" (which indicates that it can not reach the goals). While the equal heuristic values will not help FF to differentiate between "good" and "bad" nodes to explore next, the dead-end discovery will help it to eliminate many children nodes. This is especially true in the phase-transition region where there are few solutions and thus dead ends should be discovered frequently.

There are a couple of reasons that LPG performs similar to FF: (1) it seeds its initial flawed plan to repair with FF's first relaxed plan; (2) its heuristic function that ranks which flaw to fix next also relies on an FF-style heuristic; (3) it may eventually switch to FF if can't find a solution after a long time. The reason that LPG can outperform FF is that it starts with a relaxed-plan with size equal to the final plan, instead of an empty plan like FF. The relaxed plan may require less number of steps/fixes than building from an empty plan.

7.2 Results on UHP-inspired planning problems

Figure 9 confirms the phase transition from unsolvable to solvable as the connectivity p is increased in the planning problems inspired by undirected Hamiltonian path problems. The phase transition is sharp already at problem size $n = 40$, where n is the number of actions in the planning problem.

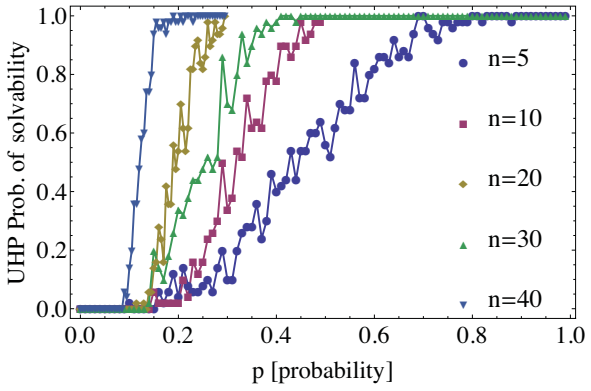


Figure 9: Fraction of solvable instances as a function of the connectivity parameter p for several different problem sizes n undirected Hamiltonian path (UHP) planning problems. The phase transition is sharp even for problem sizes as small as $n = 40$.

Figure 10 shows that the runtime increases sharply in the transition region even for small problem sizes. The results shown are for the FF planner.

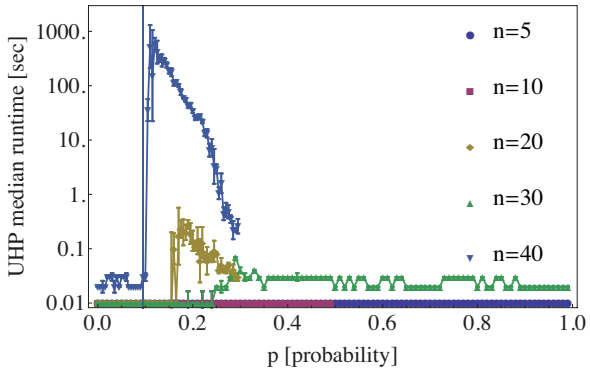


Figure 10: Typical runtime as a function of the connectivity parameter p for several different sizes of undirected Hamiltonian path (UHP) planning problems. Around the phase transition, the typical runtime shoots up markedly even at small problem sizes. The results shown are for the FF planner.

Figure 11 shows the runtime of different planners on problems at the phase transition, where we use the scaling parameter $p = (\log n + \log\log n)/n$ that has been established for the closely related Hamiltonian cycle problem [Komlós and Szemerédi, 1983; Cheeseman *et al.*, 1991]. For the other two planners, the expected exponential scaling of difficulty, as measured by runtime, with the problem size is seen clearly. The FF and LPG planners again outperforms the M and Mp planners. This time, there is a difference in performance between FF and LPG, with FF performing better than LPG. While on the DHP problems, Mp, significantly outperformed M, the reverse is true for the UHP problems. Also, for UHP, the slopes are significantly worse for M and Mp, so for larger problems, FF and LPG are likely to retain their advantage.

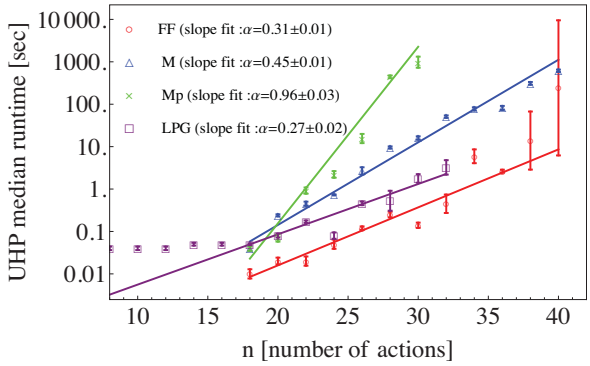


Figure 11: Typical runtime vs problem size at the phase transition, $p = (\log n + \log\log n)/n$, for undirected Hamiltonian path (UHP) problems for different planners. The exponential scaling of runtime with problem size is evident for all planners. The exponential coefficient α is given in each case. The FF planner significantly outperforms the M planner on these problems.

Planners Comparison Analysis: The performance comparison between different planners follows a similar pattern to the DHP problems. One of the main differences is the switch in relative performances between M and Mp: while Mp performs better in DHP, M is performing better in UHP. As de-

scribed in the previous section, the main difference between M and Mp is the SAT variable selection. While M uses a general SAT solver's algorithm, Mp tries to influence the decision by setting the (directed) goal orderings between goals to achieve and prioritize the actions that achieve them (and the sub-goals caused by achieving certain goals). Clearly, this directed goal ordering technique fits better with the DHP problems which have more inherent order to which goals can be achieved through the set of directed edges between nodes. This ordering technique does not work well in UHP where the edges are not directed and thus less order between goals exists in the original problem, especially when we allow any node to be visited first. Note that FF also utilize goal-ordering techniques to partially order goals and related subgoals. That technique seems to help it here to perform better than LPG, which only uses FF's relaxed-plan heuristic.

7.3 Results on GC-inspired planning problems

We now turn to results on scheduling-type planning problems. Figure 12 confirms the phase transition from solvable to unsolvable in the planning problems inspired by 3-color graph coloring problems. The phase transition is sharp already at problem size $n = 18$, where n is the number of actions in the planning problem.

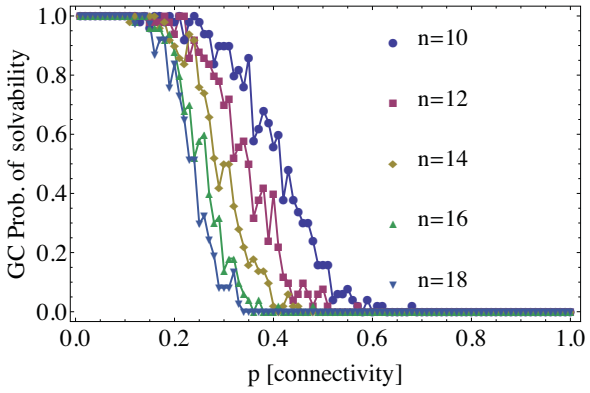


Figure 12: Graph Coloring (GC) Planning problem. Fraction of solvable instances as a function of the connectivity parameter p for several different problem sizes.

Figure 13 shows that the runtime increases sharply in the transition region even for small problem sizes. The results shown are for the FF planner.

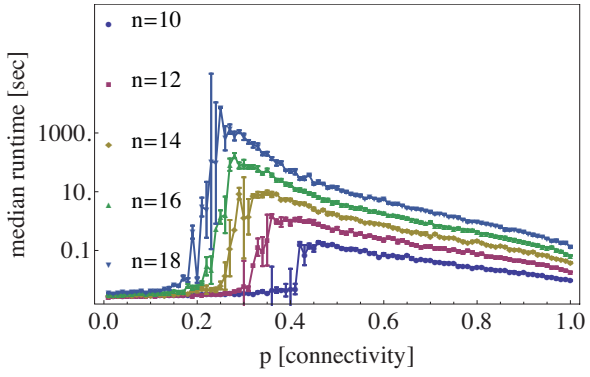


Figure 13: Graph Coloring (GC) Planning problem. Typical runtime as a function of the connectivity parameter p for several different problem sizes.

Achlioptas and Friedgut [Achlioptas and Friedgut, 1999] showed that there is a sharp phase transition threshold in the k-colorability of $G(n, p)$ graphs for all $k \geq 3$ in terms of the parameter $c = m/n = pn$, the ratio of the number of edges to the number of vertices. It is known that the threshold value scales as $k \log k$ as the leading term, but the precise location of this threshold is still an open question, even for $k = 3$. For $k = 3$, the best current lower bound [Achlioptas and Moore, 2003] is $c = 4.03$, and the best current upper bound [Dubois and Mandler, 2002] is $c = 4.94$. See [Coja-Oghlan, 2013] for a recent survey of these results, as well as new results related to upper-bounding the k-colorability threshold in general. Our runs were done with $c = 4.5$, a value intermediate to the best current lower bound and upper bound for the phase transition. The results for the different planners on problems at this phase transition are shown in Figure 14. In contrast to the results for navigation-type problems, the M planner significantly outperforms the other planners on these scheduling type problems.

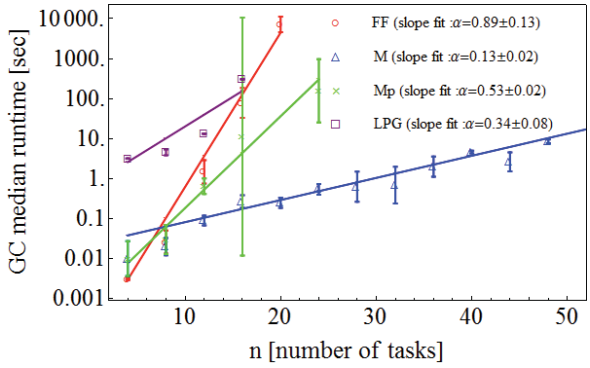


Figure 14: Graph Coloring (GC) Planning problem. Typical runtime vs problem size at the phase transition, for different planners.

Planners Comparison Analysis: Graph coloring problems have very different structure than the navigation-based problems such as DHP and UHP. The main difference is that there are multiple actions that can be executed in parallel. Specifically, any two actions that do not use the same color or use the same color on two nodes that are not connected together can be executed in parallel. This structure favors compilation-based planners such as M and Mp because they only have to setup the encoding horizon to be equal either to the number of colors or the number of nodes to guarantee that solving that single encoding is enough. However, sequential planners such as FF can not take advantage of that. By adding one action at a given search step (and thus always returning a sequential plan even if that plan is highly parallelizable), the FF planner always find solutions at depth n while M and Mp can find solution at a much lower planning horizon. This leads to better performance by M and Mp compared to FF. Given that there is no inherent ordering between goals (which nodes should be colored first), the Mp planner, which builds and utilizes goal

orderings, does not perform as well as the M planner, which uses general SAT solvers. While it's rather easy to see that LPG, which utilizes FF's relaxed-plan heuristic that is not informed in this domain, does not do well in this domain, it's not clear why it seems to perform worse than FF.

7.4 Results on Rintanen's family of planning problems

We tested the performance of the FF and M planners on the family of hard but solvable planning problems designed by Rintanen [Rintanen, 2012a] that we discussed briefly in Sec. 2. We confirmed Rintanen's results that these problems are indeed hard for the two planners we tried, and that the M planner significantly outperforms the FF planner on these problems. Fig. 15 shows the exponential scaling of the solution runtime with the size of the problem for both planners. The difficulty of these problems, however, increases an order of magnitude more slowly than for the previous sets of problems we described. Even on the largest problems we tried the M planner typically solves these problems in just a few seconds, and at small sizes the exponential behavior is not apparent.

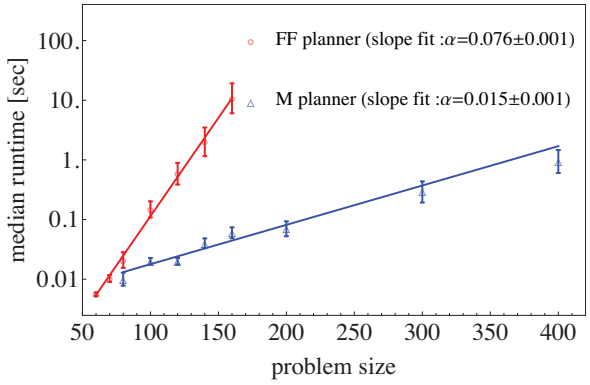


Figure 15: Scaling of runtimes on the designed by Rintanen. Typical (median) runtime vs problem size for Rintanen's family of hard but solvable problems on two different planners. Both show exponential scaling behavior, but the M planner significantly outperforms the FF planner. On problems of size 180 or greater, the FF planner takes much longer that our two-hour cutoff time.

8 Mapping planning problems to QUBO form

Quantum annealing [Das and Chakrabarti, 2008; Johnson *et al.*, 2011; Smelyanskiy *et al.*, 2012] works by starting the system in a state corresponding to a known, easy-to-implement Hamiltonian H_I and gradually varying the Hamiltonian until it becomes a Hamiltonian H_P that encodes the problem at hand. The Hamiltonians correspond to cost functions and quantum annealing explores the cost-function landscape, but has means of exploration not open to classical methods such as quantum tunneling (Figure 16). The D-Wave quantum annealing machine can accept problems phrased in terms of a Hamiltonian in Ising form:

$$E_{\text{Ising}}(s_1, \ldots, s_N) = -\sum_{i=1}^{N} h'_i s_i + \sum_{\langle i,j \rangle \in \mathrm{E}} J'_{i,j} s_i s_j, \quad (1)$$

where $s_i = \pm 1$. In traditional computer science, it is unusual to have variables s_i whose values can be taken only from $\{-1, 1\}$, but it is common to have binary variables z_i that take values from $\{0, 1\}$. It is easy to convert between the two forms by taking $s_i = 1 - 2z_i$. Any quadratic function of variables z_i can be converted to Ising form, up to a constant which does not affect the energy minimization and so can be ignored:

$$q(z_1, \ldots, z_N) = -\sum_{i=1}^{N} h_i z_i + \sum_{\langle i,j \rangle \in \mathrm{E}} J_{i,j} z_i z_j, \quad (2)$$

Thus, it suffices to express the problem we want solved as a Quadratic Unconstrained Binary Optimization (QUBO) problem [Choi, 2008; Smelyanskiy *et al.*, 2012; Lucas, 2013], which will then be converted to Ising form to run on the D-Wave machine.

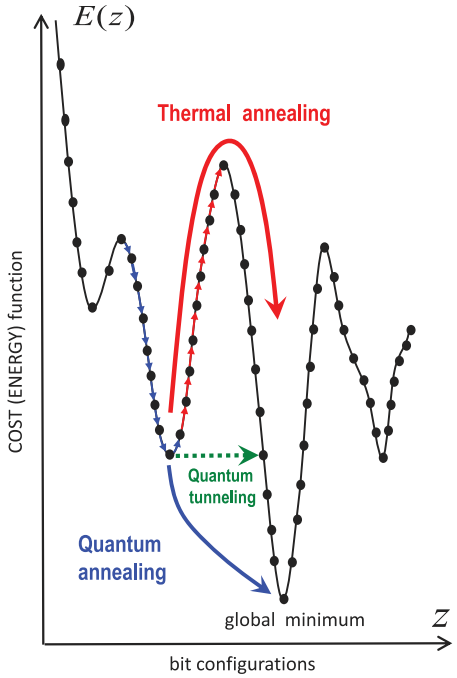


Figure 16: A schematic illustrating quantum annealing, including its capability to use quantum tunneling which is not available to classical approaches

In this section, we describe two different mappings from general classical planning problems, as described in Section 3, to QUBO form. The first is a direct time-slice approach. The second first maps the planning problem to a constraint satisfaction problem, and then reduces higher order terms to quadratic terms through a series of moves. Note that it would be wise to first simplify the planning problems before carrying out these mappings since quantum approaches would not be bound by, for example, the constraints that the precondition and goal variables be positive.

8.1 Direct time-slice method

This mapping from general classical planning problems to QUBO form is a variant of the one developed and described in [Smelyanskiy *et al.*, 2012].

If the original planning problem has N state variables and we are looking for a plan of length L, then the QUBO problem will have $N(L+1)$ binary variables $x_i^{(t)}$, where $t \in \{0, \ldots, L\}$ is the time index, and i is the index of the state variable in the original planning problem. In addition, if the original planning problem has M possible actions, we will have LM additional binary variables $y_j^{(t)}$ which indicate whether the jth action is carried out at time step t or not. We can think of the entire set of binary variables as an alternating string of N variables corresponding to the state at a given time and M variables corresponding to the actions taken at a given time. The structure of the QUBO is illustrated in Figure 17.

The total cost function is written as a sum

$$\begin{aligned} H &= H_{\text{initial}} + H_{\text{goal}} + H_{\text{no-op}} \\ &\quad + H_{\text{precond}} + H_{\text{effects}} + H_{\text{conflicts}}. \end{aligned}$$

We first give a mapping that is more general than we need, and then explain how it can be simplified in our situation. The first two terms are straightforward. They capture the initial condition and the goal condition. We describe the mapping for general classical planning problems that do not necessarily follow the convention that preconditions and goals must be positive. Let $\mathcal{I}^{(+)}$ be the set of state variables that are 1 in the initial condition and $\mathcal{I}^{(-)}$ be the set of state variables that are initially set to 0. Similarly, let $\mathcal{G}^{(+)}$ be the set of goal variables with value 1 and $\mathcal{G}^{(-)}$ be the set of goal variables with value 0. To capture the boundary conditions, the requirement that a plan start in the appropriate initial state and meets the goals, we include the following terms in the cost function:

$$H_{\text{initial}} = \sum_{i \in \mathcal{I}^{(+)}} \left(1 - x_i^{(0)}\right) + \sum_{i \in \mathcal{I}^{(-)}} x_i^{(0)}$$

and

$$H_{\text{goal}} = \sum_{i \in \mathcal{G}^{(+)}} \left(1 - x_i^{(L)}\right) + \sum_{i \in \mathcal{G}^{(-)}} x_i^{(L)}.$$

We next need to add terms to the cost function that penalize a plan if an action is placed at time t but the prior state does not have the appropriate preconditions or if the subsequent state does not reflect the effects of that action. Furthermore, we must penalize variable changes that are not the result of an action. We start with this term, the $H_{\text{no-op}}$ term, that penalizes variable changes:

$$H_{\text{no-op}} = \sum_{t=1}^{L} \sum_{i=1}^{N} \left[x_i^{(t-1)} + x_i^{(t)} - 2 x_i^{(t-1)} x_i^{(t)} \right]$$

This term gives cost penalty of 1 for every time a variable is flipped. Of course, when the effect of an action does result in a variable flipping, we do not want this penalty, so we will make up for this penalty when we add the term that corresponds to

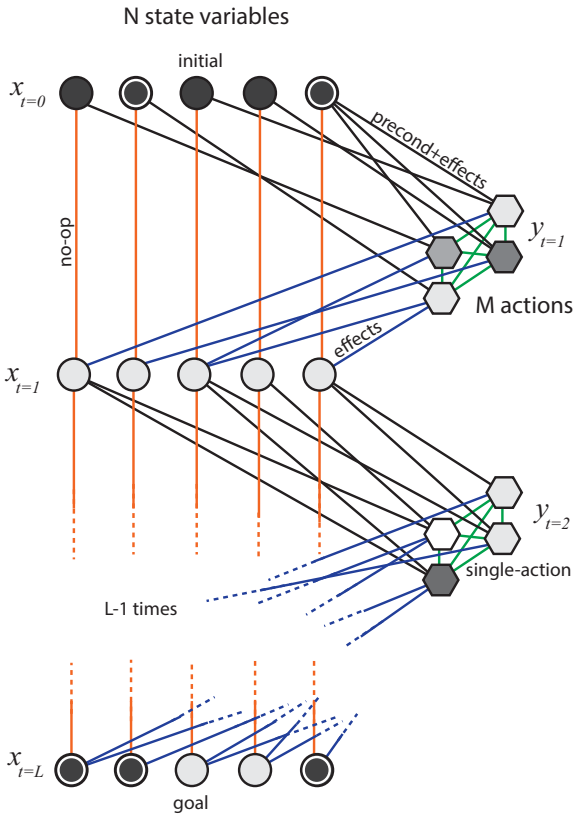


Figure 17: Direct time-slice QUBO structure for a planning problem with only positive preconditions and goals. Each node represents a state variable (left) or an action (right) at any given time t. Time flows from top to bottom, and variables $y_i^{(t)}$ for the actions at time t are shown between the state variables $x_i^{(t-1)}$ for one time step and the state variables $x_i^{(t)}$ for the next time step. The node grayscale intensity represents the magnitude of local field (bias) h_i applied to a given qubit i, and the double contour in a node indicates a negative bias. (One interesting property of this mapping is that the nodes representing state variables for $t \neq 0, L$ are equally biased, have the same h value, since they all come from H_{no-op}. For this reason, they are all shown in same color in the diagram). Edges represents the quadratic couplings J_{ij}. Their weight that is not illustrated in the figure. In this example we consider $H_{\text{single-action}}$ instead of $H_{\text{conflicts}}$, so all of the actions at a given time step are coupled to each other.

the effects of an action. First, the term that penalizes violation of the preconditions looks like

$$H_{\text{precond}} = \sum_{t=1}^{L}\sum_{j=1}^{M}\left(\sum_{i\in\mathcal{C}_j^{(+)}}\left(1-x_i^{(t-1)}\right)y_j^{(t)} + \sum_{i\in C_j^{(-)}} x_i^{(t-1)}y_j^{(t)}\right)$$

where $\mathcal{C}_j^{(+)}$ is the set of positive preconditions for action j and $\mathcal{C}_j^{(-)}$ is the set of negative preconditions. Let $\mathcal{E}_j^{(+)}$ be the set of positive effects for action j and $\mathcal{E}_j^{(-)}$ the set of negative effects. The penalty if the appropriate effects to not follow the actions is captured by the following term:

$$H_{\text{effects}} = \sum_{t=1}^{L}\sum_{j=1}^{M}\left(\sum_{i\in\mathcal{E}_j^{(+)}} y_j^{(t)}\left(1+x_i^{(t-1)}-2x_i^{(t)}\right) + \sum_{i\in\mathcal{E}_j^{(-)}} y_j^{(t)}\left(2x_i^{(t)}-x_i^{(t-1)}\right)\right).$$

In order to understand this term, we must consider it together with the no-op term. When $y_j^{(t)}=1$, the corresponding term for $i\in\mathcal{E}_j^{(+)}$ (resp. $i\in\mathcal{E}_j^{(-)}$), *taken together with the no-op term*, can be written

$$\left(1+2x_i^{(t-1)}\right)\left(1-x_i^{(t)}\right)$$

(resp.

$$\left(3-2x_i^{(t-1)}\right)x_i^{(t)}$$

for negative effects), results in positive penalty unless $x_i^{(t)}=1$ (resp. $x_i^{(t)}=0$). By using this form we have corrected for the corresponding no-op term.

Classical planners often allow for more than one action to take place at one time if they could have been done in any order, meaning that the effects of any one action do not conflict with preconditions of the other action. What we have done so far works fine when the preconditions mean that only one action can take place per time period as is the case in the navigation problems. In the scheduling problems, multiple actions can take place at the same time without conflicting. For general planning problems, we can either rule out multiple actions by imposing an additional term

$$H_{\text{single-action}} = \sum_{t=1}^{L}\left(\sum_{j=1}^{M} y_j^{(t)}-1\right)^2,$$

or we need to add terms to penalize potential conflicts. To complicate matters, when more than one action can take place at a given time, we are in danger of over-correcting for the no-op term. If multiple actions at the same time have the same effect, the H_{effects} term will add a term for each of those actions, thus overcompensating for the no-op penalty. To avoid overcompensating, we penalize multiple actions at the same time having the same effect, discouraging all such actions. A less stringent way to avoid overcompensating would be to add this penalty only when the effect changes the variable, as we have done in the no-op term. The problem is that natively that is not a quadratic term. Of course one could then reduce that term, but here we choose to use the more stringent solution. To ensure that two actions that conflict in the sense that positive preconditions of one overlap with negative effects of the other or vice versa, and to avoid overcompensating, we include the penalty

$$H_{\text{conflict}} = \sum_{t=1}^{L}\sum_{i=1}^{N}\left(\sum_{j|i\in\mathcal{C}_j^{(+)}\cup\mathcal{E}_j^{(-)}}\sum_{j'\neq j|i\in\mathcal{E}_{j'}^{(-)}} y_j^{(t)}y_{j'}^{(t)} + \sum_{j\left|i\in\mathcal{C}_j^{(-)}\cup\mathcal{E}_j^{(+)}\right.}\sum_{j'\neq j\left|i\in\mathcal{E}_{j'}^{(+)}\right.} y_j^{(t)}y_{j'}^{(t)}\right).$$

While for explanatory purposes it was useful to include variables for the state at time $t=0$, those can be set ahead of time, so that we don't need to include the H_{initial} term. Furthermore, since in our setting we have followed the convention that preconditions and goals must be positive, we can use simpler versions for the corresponding terms:

$$H'_{\text{goal}} = \sum_{i\in\mathcal{G}^{(+)}}\left(1-x_i^{(L)}\right)$$

and

$$H'_{\text{precond}} = \sum_{t=1}^{L}\sum_{j=1}^{M}\sum_{i\in\mathcal{C}_j^{(+)}}\left(1-x_i^{(t-1)}\right)y_j^{(t)}.$$

For the navigation problems, the QUBO simplifies to

$$H = H'_{\text{goal}} + H_{\text{no-op}} + H'_{\text{precond}} + H_{\text{effects}},$$

and for the scheduling problems the QUBO simplifies to

$$H = H'_{\text{goal}} + H_{\text{no-op}} + H'_{\text{precond}} + H_{\text{effects}} + H_{\text{single-action}},$$

or, if we would like to allow multiple actions at the same time we can replace $H_{\text{single-action}}$ with H_{conflict}.

8.2 CNF approach

We used the M planner to output planning problems in conjunctive normal form (CNF). A CNF expression over n Boolean variables x_i consists of a bunch of clauses C_a consisting of k variables, possibly negated, connected by logical ORs:

$$b_1 \vee b_2 \vee \cdots \vee b_k$$

where

$$b_i \in \{x_1, x_2, ..., x_n, \neg x_1, \neg x_2, ..., \neg x_n\},$$

and the number of variables k in the clause can vary from clause to clause. A CNF for a k-SAT expression consists of clauses that all have the same number of variables k. In a CNF coming from 2-SAT, for instance, all clauses have the form $b_1 \vee b_2$. In a CNF, all of the clauses must be satisfied, which means they are connected by an AND operator (the reason for the "conjunctive" in "conjunctive normal form). An example of full CNF expression consisting of L clauses connected by logical ANDs is

$$\begin{aligned} C_1 \wedge C_2 \wedge ... \wedge C_L \quad = \quad & (b_1^{(1)} \vee b_2^{(1)} \vee \cdots \vee b_{k_1}^{(1)}) \wedge \\ & (b_1^{(2)} \vee b_2^{(2)} \vee \cdots \vee b_{k_2}^{(2)}) \wedge \\ & \cdots \wedge (b_1^{(L)} \vee b_2^{(L)} \vee \cdots \vee b_{k_L}^{(L)}). \end{aligned}$$

We now discuss how to turn a CNF expression into a QUBO. We can translate each clause into a polynomial expression in binary variables $z_1, \ldots, z_n$. with 1 corresponding to TRUE and 0 corresponding to FALSE. For each Boolean variable x_i included in a clause C_a, we include a $(1 - z_i)$ factor and for each negated variable $\neg x_j$ we include a z_j factor. The product of these factors is a polynomial expression P_a that is 0 for values of z_i corresponding to the original clause being TRUE and 1 for values of the z_i that corresponding to the original clause being FALSE. For example, the clause $C_1 = x_1 \vee \neg x_2 \vee \neg x_3$ translates to $P_1 = (1 - z_1) z_2 z_3$, which is zero if $z_1 = 1$ or $z_2 = 0$ or $z_3 = 0$ and 1 otherwise (i.e. if $z_1 = 0$ and $z_2 = 1$ and $z_3 = 1$). We can sum together the polynomial expressions for all of these clauses to obtain

$$P = \sum_{a=1}^{L} P_a$$

which is zero exactly for values that correspond to making the original CNF TRUE. If we had started with a 2-SAT problem, we would now be done because the polynomial expression P would be quadratic. For general CNF expressions that include clauses with more that two variables, however, we have a little more work to do to obtain QUBO form.

For clauses with $k > 2$ variables, we introduce $k - 2$ auxiliary variables, $y_1, ..., y_{L-2}$. We then rewrite the clause using these auxiliary variables as in the following example: the four-variable clause

$$C_a = (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$$

is transformed into a six-variable logical expression,

$$C_a' = (x_1 \vee y_1) \wedge (y_1 \bar{\oplus} (\neg x_2 \vee y_2)) \wedge (y_2 \bar{\oplus} (\neg x_3 \vee x_4)) \ ,$$

where $\bar{\oplus}$ is the symbol for the logical operator NOT XOR, The resulting expression contains only expressions of the form $b_i \vee b_j$ or $b_1 \bar{\oplus} (b_1 \vee b_2)$. As before, we convert expressions of the first form into a quadratic expression over binary variables. We need to make sure we can do so for the second type of term. If all of the variables are not negated, then we can convert

$$y_1 \bar{\oplus} (x_1 \vee x_2)$$

into the quadratic expression

$$(x_1 + x_2)(1 - y_1) + c_2 [x_1 x_2 + y_1 (1 - x_1 - x_2)] \ .$$

If the expression contains negated variables, replace each negated variable x_i with $1 - x_i$ in the above expression. In this way we convert any CNF to QUBO form.

9 Conclusions and Future Work

Our parametrized families of planning problems are based on well-known NP-complete problems, for which the phase transition is known, and the easy-hard-easy pattern is seen. Our families of planning problems fall into two main classes, navigation-type and scheduling-type. We have confirmed the parameters that yield hard problems based on existing results for the NP-complete problems on which these planning problems are based. Even at small problem sizes the exponential increase in the difficulty of the problem with the size of the problem is evident in the time it takes state-of-the-art planners to solve these problems. Different planners perform comparatively well or badly, depending on the problem family. We have analyzed results from four state-of-the-art planners, and discussed their implications.

Advantages of this approach, complementing current benchmark sets obtained by extracting problems from real world applications include insight into what types of domains are easy or hard for different planners, the ability to define what it means for a small problem to be hard, and the capability to do examine scaling behavior with increasing problem size. These families of problems can be used as benchmark problems for new planning algorithms as well as existing planners. Their small size complements the large problems generally used for benchmarking planning problems in planning competitions today.

The next step will be to run instances of these problems on the quantum annealing device, exploring tradeoffs in different ways of mapping these problems to quantum annealing, as well as comparing performance with classical approaches. As mentioned above, it is a good idea to first simplifying the planning problem statement to remove redundant variables that were only needed to satisfy constraints on certain classical planners. One step not discussed in this work is the need to embed the problems in the specific hardware graph of the quantum annealing device at hand. The interaction between different mapping approaches, different embedding strategies, and the performance of the quantum device on the same problem in these different guises will be interesting to understand.

On the purely classical side, we hope that this work will spur the development of more parametrized families of planning problems that capture other aspects common to many planning problems.

Acknowledgements

The authors would like to thank Jüssi Rintanen for sharing with us his code for the generation of the hard family of problems he designed, for making his code for the M planner available, and for generously answering our questions.

The authors would also like to thank Dimitris Achlioptas for discussions related to the state-of-the-art knowledge about phase transitions in graph coloring problems.

They are grateful to Jeremy Frank, Vadim Smelyanskiy, and Sergey Knysh for useful discussions throughout the course of the work, and to Jeremy for detailed comments on an earlier draft of this report, and to Bryan O'Gorman for

reading through and giving us comments on a nearly final version of this report.

References

[Achlioptas and Friedgut, 1999] Dimitris Achlioptas and Ehud Friedgut. A sharp threshold for k-colorability. *Random Structures and Algorithms*, 14(1):63–70, 1999.

[Achlioptas and Moore, 2003] Dimitris Achlioptas and Cristopher Moore. Almost all graphs with average degree 4 are 3-colorable. *Journal of Computer and System Sciences*, 67(2):441–471, 2003.

[Aghevli *et al.*, 2007] A. Aghevli, A. Bachmann, J.L. Bresina, J. Greene, R. Kanefsky, J. Kurien, M. McCurdy, P.H. Morris, G. Pyrzak, C. Ratterman, A. Vera, and S. Wragg. Planning applications for three mars missions. In *International Workshop on Planning and Scheduling for Space*, 2007.

[Bjorklund *et al.*, 2009] A. Bjorklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546563, 2009.

[Bonet *et al.*, 1997] Blai Bonet, Loerincs G., and Héctor Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, 1997.

[Bresina *et al.*, 2005] John Bresina, Ari Jonsson, Paul Morris, and Kanna Rajan. Activity planning for the Mars exploration rovers. In *Proceedings of International Conference on Automated Planning and Scheduling*, pages 40–49, 2005.

[Bylander, 1996] Tom Bylander. A probabilistic analysis of propositional STRIPS planning. *Artificial Intelligence Journal*, 81:241–271, 1996.

[Cheeseman *et al.*, 1991] Peter Cheeseman, Bob Kanefsky, and William M Taylor. Where the really hard problems are. In *IJCAI*, volume 91, pages 331–337, 1991.

[Chien *et al.*, 2012] Steve Chien, Mark Johnston, Jeremy Frank, Mark Giuliano, Alicia Kavelaars, Christoph Lenzen, Nicola Policella, and Gerald Verfailie. A generalized timeline representation, services, and interface for automating space mission operations. In *12th International Conference on Space Operations*, 2012.

[Choi, 2008] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing*, 7(5):193–209, 2008.

[Coja-Oghlan, 2013] Amin Coja-Oghlan. Upper-bounding the k-colorability threshold by counting covers. arXiv:1305.0177, 2013.

[Culberson *et al.*, 1995] Joseph Culberson, Adam Beacham, and Denis Papp. Hiding our colors. In *Proceedings of the CP95 Workshop on Studying and Solving Really Hard Problems*, pages 31–42, 1995.

[Dailey, 1980] D.P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discrete Mathematics*, 30(3):289293, 1980.

[Das and Chakrabarti, 2008] Arnab Das and Bikas K. Chakrabarti. Colloquium: Quantum annealing and analog quantum computation. *Rev. Mod. Phys.*, 80:1061–1081, 2008.

[Dubois and Mandler, 2002] Olivier Dubois and Jacques Mandler. On the non-3-colourability of random graphs. arXiv:math/0209087, 2002.

[Gerevini *et al.*, 2003] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

[Guruswami and Khanna, 2000] V. Guruswami and S. Khanna. On the hardness of 4-coloring a 3-colorable graph. In *15th Annual IEEE Conference on Computational Complexity*, page 188197, 2000.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Hoffmann, 2005] Jörg Hoffmann. Where ignoring delete lists works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.

[Huberman and Hogg, 1987] Bernardo A Huberman and Tad Hogg. Phase transitions in artificial intelligence systems. *Artificial Intelligence*, 33(2):155–171, 1987.

[Johnson *et al.*, 2011] M. W. Johnson, M. H. S. Amin, S. Gildert, and et al. Quantum annealing with manufactured spins. *Nature*, 473:194–198, 2011.

[Johnston *et al.*, 2009] M.D. Johnston, D. Tran, B. Arroyo, and C. Page. Request-driven scheduling for NASA's deep space network. In *International Workshop on Planning and Scheduling for Space (IWPSS)*, 2009.

[Kautz and Selman, 1999] Henry A. Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *Proceedings of IJCAI'1999*, 1999.

[Komlós and Szemerédi, 1983] János Komlós and Endre Szemerédi. Limit distribution for the existence of Hamiltonian cycles in a random graph. *Discrete Mathematics*, 43(1):55–63, 1983.

[Long and Fox, 2003] Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res.(JAIR)*, 20:1–59, 2003.

[Lucas, 2013] Andrew Lucas. Ising formulations of many NP problems. arXiv:1302.5843, 2013.

[Marquez *et al.*, 2010] J. Marquez, M. Ludowise, M. McCurdy, and J. Li. Evolving from planning and scheduling to real-time operations support: Design challenges. In *40th International Conference on Environmental Systems*, 2010.

[Morris *et al.*, 2013] Paul Morris, Mark Schwabacher, Michael Dalal, and Charles Fry. Embedding temporal constraints for coordinated execution in habitat automation. In *Proceedings of the 8th International Workshop on Planning and Scheduling for Space (IWPSS'2013)*, 2013.

[Oddi and Policella, 2007] Angelo Oddi and Nicola Policella. Improving robustness of spacecraft downlink schedules. In *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, pages 887–896, 2007.

[Porco *et al.*, 2011] Aldo Porco, Alejandro Machado, and Blai Bonet. Automatic polytime reductions of NP problems into a fragment of STRIPS. In *ICAPS*, pages 178 – 185, 2011.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: guiding cost-based anytime planning with landmarks. *Journal of Artificial Research*, 39:127–177, 2010.

[Rintanen, 2004] Jussi Rintanen. Phase transitions in classical planning: an experimental study. In *Proceedings of the 14th International Conference on Automated Planning & Scheduling (ICAPS-2004)*, pages 101–110, 2004.

[Rintanen, 2012a] Jussi Rintanen. Generation of hard solvable planning problems. In *Technical Report (TR-CS-12-03), Australian National University.*, 2012.

[Rintanen, 2012b] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[Selman *et al.*, 1996] Bart Selman, David G Mitchell, and Hector J Levesque. Generating hard satisfiability problems. *Artificial intelligence*, 81(1):17–29, 1996.

[Slaney and Thiebaux, 1998] John Slaney and Sylvie Thiebaux. On the hardness of decision and optimization problems. In *Proc. of the 13th European Conference on Artificial Intelligence*, 1998.

[Smelyanskiy *et al.*, 2012] Vadim N Smelyanskiy, Eleanor G Rieffel, Sergey I Knysh, Colin P Williams, Mark W Johnson, Murray C Thom, William G Macready, and Kristen L Pudenz. A near-term quantum computing approach for hard computational problems in space exploration. arXiv:1204.2821, 2012.