PFPT: a Personal Finance Planning Tool by means of Heuristic Search and Automated Planning

Alberto Pozanco, Kassiani Papasotiriou, Daniel Borrajo[*]
J.P. Morgan AI Research
{alberto.pozancolancho, kassiani.papasotiriou, daniel.borrajo}@jpmorgan.com

Abstract

A crucial component to an individual's financial well-being is staying proactive in terms of the personal finances. Seeking such advice helps individuals or households to plan, save, and spend monetary resources over time, while taking into account various financial risks and future life events. Receiving such advice at the individual level usually happens by consulting a personal finance advisor which can be very expensive. In this paper we present PFPT, a Personal Finance Planning Tool that can use different search approaches to propose actionable plans to end users in order to achieve their financial goals. We evaluate PFPT in different problems using two different approaches: domain-independent automated planning and domain-dependent heuristic search. Results show that while automated planning struggle to generate good plans in this domain, our suggested heuristics are able to scale on generating realistic financial plans.

Introduction

Setting financial goals and planning ahead plays a significant role in ensuring financial health for an individual or a household. Personal finance planning activities include managing monetary resources through expenditure, investments, and savings, while considering various life events, risks and goals. The benefits of financial planning have been studied and quantified using economic well-being indicators in both empirical (Peng et al. 2007; Farinella, Bland, and Franco 2017; Warschauer and Sciglimpaglia 2012) and theoretical settings (Hanna and Lindamood 2010).

The most common way of seeking financial advice is by consulting a personal finance professional who can help clients make decisions about investments, budgeting or other courses of action to achieve their goals. Such services are often very expensive and thus inaccessible to a lot of people. Alternatives to speaking to an advisor include personal finance assessment tools and questionnaires which offer semi-personalized advice to users based on their input. However, these tools fail to recommend actionable points of advice on a more personal and detailed level.

We present in this paper PFPT, a Personal Finance Planning Tool, which offers financial advice at the individual level. It allows users to define both long-term and short-term financial goals and recommends actions to successfully achieve them based on their financial habits. We model this problem from a search perspective by defining states, actions and goals and apply domain-independent automated planning and domain-dependent heuristic search to recommend plans that maximize the likelihood of being executed based on the individual financial habits. To the best of our knowledge this is the first financial tool that applies planning and search techniques for personal finance management.

Previous technical methods of financial planning include expert systems which try to mimic the knowledge and experience of a human experts. The systems collect detailed user information regarding an individual's financial state and consists of a rules base to produce possible solutions to a goal (Kindle et al. 1989; Phillips, Nielson, and Brown 1992). More recent approaches used rule-based approaches based on different metrics and definitions on financial well-being (Althnian 2021). The main weakness of these approaches is that they do not provide flexible and detailed solutions and do not take into account the feasibility of the recommended plans. Other methods use Deep Reinforcement Learning techniques that often address a subset of financial goals, such as portfolio management (Irlam 2018; 2020).

The rest of the paper is organized as follows. In the next section we provide some background on numerical planning. Then, we define the problem solved by PFPT: finding a plan to go from an initial finance state to a goal finance state by maximizing the likelihood of the employed actions. After that, we introduce two different approaches to solve the PFPT problem: domain-independent automated planning and domain-dependent heuristic search, where we define a set of heuristics to guide the search. Later, we evaluate both approaches, focusing on analyzing the behavior of the different heuristics. Finally, we draw our main conclusions and outline future work.

Background

We use the standard classical STRIPS definition of a planning task, augmented with numeric variables (Fox and Long 2003). Formally:

[*]On leave from Universidad Carlos III de Madrid

Definition 1. *A **numeric planning task** is a tuple* $\Pi = \langle F, A, I, G \rangle$*, where* F *is a set of boolean and numeric variables,* A *is a set of actions,* $I \subseteq F$ *is the initial state and* $G \subseteq F$ *is a set of goals.*

We denote with S the set of all states of the planning task Π. A (full) state $s \in S$ is a valuation of all the variables in F; a boolean value for all the boolean variables and a numeric value for the numeric ones.

Each action $a \in A$ is defined in terms of its preconditions (pre(a)) and effects (eff(a)). Effects can set to true the value of a boolean variable (add effects, add(a)), set to false the value of a boolean variable (del effects, del(a)), and change the value of a numeric variable (numeric effects, num(a)). Action execution is defined as a function $\gamma : S, A \rightarrow S'$; that is, it defines the state that results of applying an action in a given state. It is usually defined as $\gamma(s, a) = (s \backslash \text{del}(a)) \cup \text{add}(a)$ if pre(a)$\subseteq$ s when only boolean variables are considered. When using numeric variables, γ should also change the values of the numeric variables (if any) in num(a), according to what the action specifies; increasing or decreasing the value of a numeric variable or assigning a new value to a numeric variable. If the preconditions do not hold in s, the state does not change.

The solution of a planning task is called a plan, and it is a sequence of instantiated actions that allows the system to transit from the initial state I to a state s where goals are true. Therefore, a plan $\pi = \langle a_1, a_2, \dots a_n \rangle$ solves a planning task Π (valid plan) iff $\forall a_i \in \pi, a_i \in A$, and $G \subseteq \gamma(\dots \gamma(\gamma(I, a_1), a_2) \dots), a_n)$. In case the cost is relevant, each action can have an associated cost, $c(a_i), \forall a_i \in A$ and the cost of the plan is defined as the sum of the costs of its actions: $c(\pi) = \sum_i c(a_i), \forall a_i \in \pi$. A plan with minimal cost is called optimal.

PFPT Problem Definition

We aim to find realistic financial plans that allow users to go from their current financial state to their goal financial state. We define a financial state as follows:

Definition 2. *A **financial state** is a tuple* $s = \langle t, \mathit{Inc}, \mathit{DExp}, \mathit{FExp}, B \rangle$*, where:*

- $t \in \mathbb{N}$ *is a time step*
- $\mathit{Inc} \in \mathbb{R}$ *is the income per time step*
- $\mathit{DExp} \in \mathbb{R}$ *are the discretionary expenses per time step*
- $\mathit{FExp} \in \mathbb{R}$ *are the fixed expenses per time step*
- $B = (\hat{B} + \mathit{Inc} - \mathit{DExp} - \mathit{FExp})$ *is the account balance, where* $\hat{B}$ *is the account balance of s's parent at* $t - 1$*.*

The initial financial state is fully specified, while the goal financial state is usually partially specified. For example, the goal state could specify that the balance at a given time step should be higher than a given quantity.

At each time step, some actions can be applied in order to change the financial state into another one. We define two types of **actions**: income increases and discretionary expenses decreases. We assume the fixed expenses cannot be changed, or will be very unlikely changed. Actions might produce changes in the financial state. For example, an income increase of 20% in state s_t will result in a new state at the next time step s_{t+1} with income $\text{Inc}(s_{t+1}) = 1.2 \times \text{Inc}(s_t)$. Besides these effects over the financial state, each action has associated a *likelihood* score, which is a real number between 0 and 1 that reflects how feasible or probable is that a user executes the action. This likelihood score can be given by users or inferred from their financial habits. For example, increasing the income by 0% will have a higher likelihood score than increasing the income by 20%, since the former is an easier or more feasible action than the latter. Table 1 summarizes a potential set of actions along with their effects to the financial state and their likelihood score. We will use this set of actions as an example in the rest of the paper.

Income increase and discretionary expenses decrease actions can be combined to generate **joint actions**. Assuming the actions described in Table 1, we would have 9 possible joint actions that can be applied at each time step, i.e., [Increase Inc 10%, Decrease DExp 0%], [Increase Inc 20%, Decrease DExp 10%], etc. A plan π solves this problem optimally if it achieves the financial goal state by maximizing the likelihood product of its actions. Formally:

$$\max \prod_{a \in \pi} \text{likelihood}(a) \tag{1}$$

We face two obstacles when trying to use search algorithms to (optimally) solve the problem as defined in Expression 1: (1) plan optimality is defined as a product, while search algorithms ordering functions are typically additive; and (2) plan optimality is defined as a maximization task (maximize likelihood), while most search algorithms aims to minimize a given function. To overcome the first problem, we compute the logarithm of each action's likelihood score so we can sum them. To overcome the second problem, we transform the maximization task into a minimization task by subtracting the logarithm of the likelihood score from one. By performing these two transformations, now we have the following additive cost function that search algorithms can minimize:

$$c(a) = 1 - log(\text{likelihood}(a)) \tag{2}$$

Given a plan π and its cost $c(\pi)$, we can compute back its likelihood score by applying the following operation:

$$\text{likelihood}(\pi) = \exp(-(c(\pi) - |\pi|)) \tag{3}$$

In the next sections we describe two different search-based approaches to solve this problem: domain-independent automated planning and domain-dependent heuristic search.

Automated Planning Approach

The first approach uses automated planning models and planners to generate solutions. The domain is composed of actions that model the actions described in Table 1. As an example, the action that increases the income, would be modeled as shown in Figure 1.

At each time step, we only allow the execution of one action of each kind: modify income or modify expenses. This

Action	Effect	Likelihood
Increase Income by 0%	-	1.0
Increase Income by 10%	Inc$(s_{t+1}) = 1.1 \times$ Inc(s_t)	0.8
Increase Income by 20%	Inc$(s_{t+1}) = 1.2 \times$ Inc(s_t)	0.6
Decrease Discretionary Expenses by 0%	-	1
Decrease Discretionary Expenses by 10%	DExp$(s_{t+1}) = 0.9 \times$ DExp(s_t)	0.9
Decrease Discretionary Expenses by 20%	DExp$(s_{t+1}) = 0.8 \times$ DExp(s_t)	0.8

Table 1: Actions' summary.

```
(:action increase-income
    :parameters (?p - percentage
                 ?t - time-step)
    :preconditions
      (and (current-time ?t)
           (not (done-income ?t)))
    :effects
      (and (increase (total-cost)
                     (likelihood ?p))
           (increase (income ?t)
                     (* (income ?t)
                        (percentage ?p)))
           (increase (balance ?t)
                     (income ?t))
           (done-income ?t)))
```

Figure 1: Model of the action that increases the income by 20%.

action is the only one defined to modify income and summarizes all possible increase operations over income. The parameters of the action are a time step and a percentage of increase. Since we do not need complex temporal reasoning, we consider discrete temporal problems, so we model time explicitly as a sequence of time steps. Percentages are also represented as discrete amounts and are defined in the problem description. In the case of the percentages defined in Table 1, we would define three objects of type percentage in the problem for the income (0, 10 and 20) and another three for modifying the expenses (also 0, 10 and 20). The reason to separate income percentages from expenses percentages is that we need also to define their corresponding likelihoods (predicate likelihood) which have different values. For instance, the likelihood of increasing income in a 20% is 0.6, while the likelihood of decreasing the expenses in a 20% is 0.8.

The preconditions of the actions are that we are at a given time step and that we did not update yet the income at that time step. The expected effects are that the income will increase based on the percentage, the balance is increased with the new income, and the total cost is updated. We use as cost the one defined in Equation 2. Apart from the increase-income and decrease-expenses, the domain also includes a move-time action that progresses time.

The problem description contains objects related to the set of time steps and the percentages. If the user sets as goal to have a balance x at time step T, the problem will be automatically generated with all the time steps between 0 and T. The initial state defines the initial income, balance, and expenses, as well as the likelihoods of each percentage, the initial total cost of 0, the initial time step of 0 and the needed next predicates to connect in sequence all the time steps. The goal description is compiled from the user goals, as for instance, the balance being greater than a given value at a given time step.

The plans are sequences of actions that achieve the goals from the initial state. They are comprised of a joint action (income, expenses) at each time step, plus a move-time action to progress to the next time step. As an example, a plan would be:

```
(increase-income t0 p-inc-0)
(decrease-expenses t0 p-exp-20)
(move-time t0 t1)
(increase-income t1 p-inc-10)
(decrease-expenses t1 p-exp-0)
```

that would not increase income and decrease expenses in a 20% in the first time step, and increase income in a 10% and not modify expenses in the second time step.

The main drawback of using planning for solving this task is that it is a numerical planning task. First, there are very few planners that can handle this kind of domain complexity. Second, to the best of our knowledge, there is no planner that can perform optimal numerical planning. Thus, we have defined a search-based solution that allows us to compute optimal solutions for this task which is presented in the next section.

Heuristic Search Approach

We use the most popular algorithm for optimal search, A*(Hart, Nilsson, and Raphael 1968), to solve this problem in a domain-dependent fashion. A* uses a function $f(s) = g(s) + h(s)$ to order the nodes in the open list. The solutions returned by A* are guaranteed to be optimal if the heuristic h is admissible, i.e., it does not over-estimate the cost of reaching the goal from any state.

The cost of reaching a state s, $g(s)$, is computed using Equation 2. In order to estimate the cost of reaching the goal from s, $h(s)$, we propose the following domain-dependent heuristics.

Minimum Cost Action

The first heuristic, which we called Min, consists on choosing the cost of the cheapest joint action $c(a)_{min}$ and multiply it by the number of remaining time steps: $h(s) = c(a)_{min} \times (t(G) - t(s))$. In our case, the cheapest joint action is to do nothing, i.e., increase the salary by 0% and decrease the discretionary expenses by 0%.

Lemma 1. *Min is admissible.*

Proof. By construction, at each time step, there is no cheaper joint action than $c(a)_{min}$. The result of multiplying the minimum cost by the $(t(G) - t(s))$ will necessarily be less than or equal h^*. Therefore, Min is admissible. □

Greedy

The next heuristic we propose consists on solving a relaxation of the problem where only the same action can be applied at every time step. In other words, the number of potential plans is limited to the number of joint actions considered. The procedure that computes the heuristic is outlined in Algorithm 1. The algorithm receives as input the current (s)

Algorithm 1 Greedy Heuristic

```
Require: $s, G, A$, Admissible
Ensure: GH
 1: GH $\leftarrow \infty$
 2: remainingTimeSteps $\leftarrow t(G) - t(s)$
 3: sortedActions $\leftarrow$ SORTBYCOST$(A)$
 4: for $a \in$ sortedActions do
 5:     $s' \leftarrow$ EXECUTE(remainingTimeSteps, $a, s$)
 6:     if $G \subseteq s'$ then
 7:         if Admissible = True then
 8:             GH $\leftarrow c(a)$
 9:         else
10:             GH $\leftarrow c(a) \times$ remainingTimeSteps
11:         return GH
12: return GH
```

and goal (G) state, the available actions (A), and a parameter that indicates whether we are interested in the heuristic to be admissible or not. The algorithm first computes the number of remaining time steps from s (line 2). If the goal state does not specify any time step, this is set to a high number. Then, the actions in A are sorted according to their cost as per Equation 1. Next, the algorithm iterates over the sorted list of actions, executing the given action a from s for the number of remaining steps, yielding a state s'. If the goal is satisfied in s', the algorithm finishes and returns the heuristic estimate. This heuristic value will depend on the admissibility parameter. If we are interested in an admissible heuristic (GH$_a$), Algorithm 1 will return the cost of executing that action, $c(a)$.

Lemma 2. *GH$_a$ is admissible.*

Proof. Suppose GH$_a$ returns $c(a)$ and $c(a) > h^*$. It means that there is a solution that only uses actions with a cost less than $c(a)$. If it would be using actions whose cost would be greater than $c(a)$, then $c(a)$ would be less than h^*, so the assumption would be false. And, if there would be a solution using only a subset of less costly actions, it would had been found before a, since they are studied from less costly to more costly. Thus, $c(a)$ is less than h^*, and it is admissible. □

If we want a more informative but inadmissible heuristic (GH$_i$), Algorithm 1 returns the cost of executing that action multiplied by the number of remaining steps.

Lemma 3. *GH$_i$ is inadmissible.*

Proof. The heuristic value returned by GH$_i$ considers executing the cheapest possible action a that reaches the goal (ensured by the actions' sorting in line 3 and the loop in line 4) in all the remaining time steps. However, reaching the goal state could only require executing a in a subset of the remaining time steps together with some lower cost actions in the other steps. Thus, GH$_i$ could return greater values than h^* for some state/goal combinations, so it is inadmissible. □

If after iterating over all the possible actions the goal cannot be achieved, Algorithm 1 will return ∞, meaning that the goal is not reachable from a.

Heuristics Behavior Example

Let us exemplify how the heuristics work and their accuracy by computing them at the initial state $(h(I))$ of the following PFPT problem:

$$I = \langle t = 0, \text{Inc} = 5, \text{DExp} = 2, \text{FExp} = 2, B = 10\rangle$$
$$G = \langle t = 4, B = 17\rangle$$

The optimal solution to this problem has a cost of 8.43 $(h^*(I))$. The minimum cost action heuristic Min returns the cost of the cheapest action multiplied by the number of remaining time steps. The cheapest joint action is to Increase Inc 0% and Decrease DExp 20%, and has an associated cost of 2. After multiplying it by the 4 remaining time steps, Min will return a cost of 8, which is a lower bound on $h^*(I)$. The greedy algorithm returns that [Increase Inc 0%, Decrease DExp 20%] is the cheapest joint action that can be subsequently executed from I in the remaining time steps to achieve the goal. This joint action has an associated cost of 2.22. Therefore, GH$_a$ will return that cost, which is a lower bound on $h^*(I)$, while GH$_i$ will return $2.22 \times 4 = 8.88$, which is an upper bound on $h^*(I)$.

Evaluation

We randomly generate PFPT problems of increasing difficulty by increasing the time horizon at which the goal balance has to be achieved. To generate hard problems, we (1) set the goal balance to be twice the initial balance; and (2) make the expenses per time step (sum of DExp and FExp) to be a random ratio between 0.9 and 1 of the income per time step, thus rendering problems where little savings are generated if the initial financial state remains unchanged. We solve these problems with the previously described heuristics: Min, GH$_a$ and GH$_i$, plus Blind, which we will use as a baseline to compare heuristics' search performance. All the heuristics have been implemented in Python 3.6, as well as the search algorithm, which is a vanilla implementation of A*. Heuristic search experiments were run in Intel(R) Xeon(R) CPU E3-1585L v5 @ 3.00GHz machines with 64GB of RAM. We also tried to solve the PDDL version of these problems using LPG (Gerevini, Saetti, and Serina 2004), a stochastic planner for numerical planning. Since the planner is stochastic, we ran the planner five times on each problem. Automated planning experiments were run in an Apple M1 Pro machine with 16GB of RAM.[1]

Automated Planning vs Heuristic Search

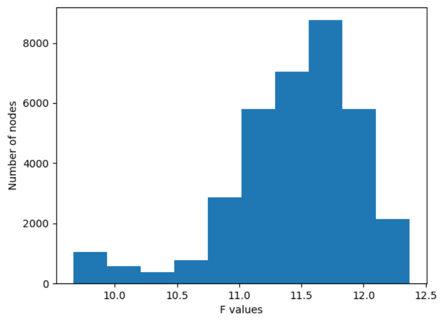
For the first set of experiments, we generated 10 random problems with the time horizon t set to 4. LPG is only able to solve 4 out of the 10 problems, reporting that no solution could be found for the other 6. The reason LPG failed to solve the problems was not due to time nor memory constraints. The execution time in the solved problems is always below 2 seconds, but the solutions returned are suboptimal. Given that the machine where LPG was run is different than the machine where the rest of code was run, the time cannot be compared directly. Instance #9 is the problem where LPG gets the closest to the optimal, returning a plan with cost 8.80, while the optimal cost is 8.32. On the other hand, instance #8 is where LPG obtain the worst results, returning a plan with cost 10.19 in one of the executions, while the optimal cost is again 8.32. Cost differences might look small, but they translate into large likelihood score differences. In instance #8, the optimal plan has a likelihood score of 0.73, while the plan returned by LPG has a likelihood score of 0.11, meaning it would be a very unrealistic plan to propose to an end user.

Heuristic	Plan Cost	Expanded	Generated	Search Time (s)
Blind	**8.80**	1698.8	14350.0	56.5
Min	**8.80**	5320.3	10310.1	60.1
GH$_a$	**8.80**	344.7	2931.2	1.6
GH$_i$	8.81	**11.3**	**91.2**	**0.0**

Table 2: Heuristics comparison in random problems with $t = 4$. Numbers represent average across 10 problems. Best values are shown in bold.

Table 2 summarizes the results of the different heuristics in the same set of problems. In this case, all the heuristics are able to solve all the problems. As expected, the three admissible heuristics return the optimal plan in all the problems (average plan cost of 8.80), while GH$_i$ returns a slightly suboptimal solution in one of the problems, thus increasing the average plan cost to 8.81. In terms of search efficiency, Min is not able to outperform Blind in this problem setting. The distribution of f values of both searches is shown in Figure 2. As we can see, Min generates search spaces where many states have the same f value of 8, generating a large plateau at the beginning of the open list. This occurs because many states have the same f even if they are still far from reaching the goal due to the heuristic value being a constant function of the remaining time steps, which translates into more expanded nodes. On the other hand, Blind generates more diverse f values, with most of the nodes having f values between 11 and 12, therefore being able to better discriminate between the states that are closer to the goal time horizon. This behavior could not be substantially improved even when considering different tie-breaking rules.

[1]As the following results show, the fact that we are using different machines is not relevant in this comparison.



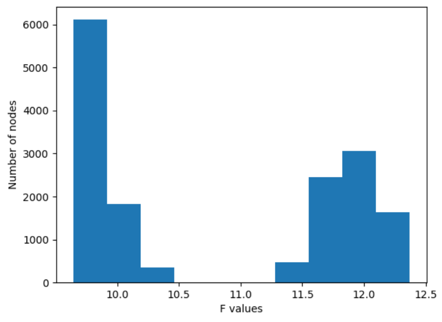


Figure 2: Open list f values distribution at the end of the search when using Blind (upper histogram) and Min (lower histogram) heuristics.

GH$_a$ reduces the search effort by an order of magnitude, returning optimal solutions in less than two seconds on average. The inadmissible greedy heuristic GH$_i$ achieve the best results in terms of search efficiency, expanding only around 11 states to reach the goal. This represents a 0.6% of the states that a Blind search needs to expand, meaning this heuristic is really informative. GH$_i$ also yields faster searches that reach the goal in less than a second.

Heuristics Scalability and Optimality

As we have seen, we cannot rely on LPG to solve this kind of problems. The behavior of the Blind, Min, and GH$_a$ heuristics also quickly deteriorates as we increase t. For example, Blind and Min are not able to find a solution for any of the 10 problems with $t \geq 5$ in less than $1800s$, while GH$_a$ cannot find solutions within that time limit in most problems with $t \geq 6$. Hence, we evaluated the scalability and performance of GH$_i$ in harder problems with longer time horizons. The scalability of GH$_i$ is shown in Figure 3, where we use violinplots to show the search time distribution when solving problems with $t = 4$, 6, 8, and 10. As we can see, solving time grows exponentially with the time horizon. However, GH$_i$ is able to generate fast searches that can find the solution in less than a second in many of the problems. We are not solving problems with larger time horizons because our vanilla implementation of A^* is not able to scale in some of

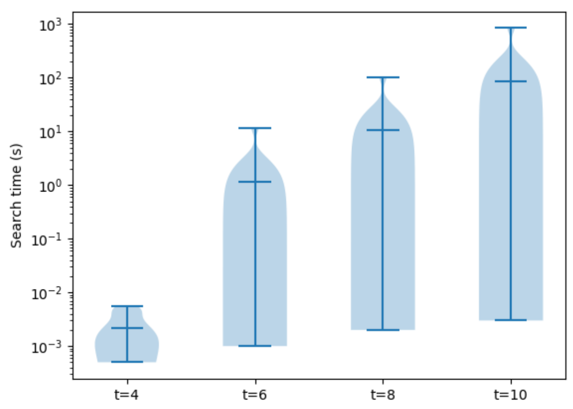


Figure 3: Search time (log scale) needed to find a solution by GH$_i$ in problems of increasing difficulty, i.e., longer time horizons.

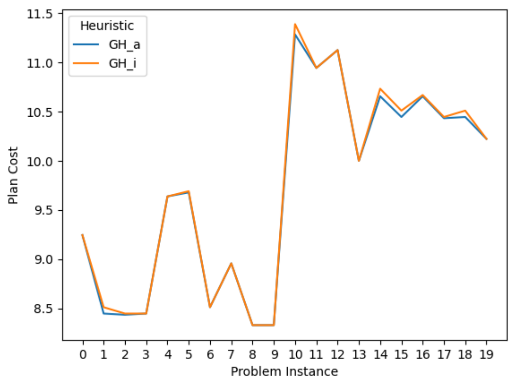


Figure 4: Plan cost as returned by GH$_a$ and GH$_i$ in problems of increasing difficulty, i.e., longer time horizon.

the bigger instances. However, our suggested greedy heuristics would be likely able to scale provided a better implementation of A^*, which we leave as future work.

Finally, we also wanted to better understand GH$_i$'s optimality loss in relation to the optimal solution. Figures 4 and 5 show the plan cost and likelihood respectively as returned by GH$_a$ and GH$_i$ in problems with $t = 4$ and $t = 5$, where GH$_a$ can compute the optimal plan within the time bound. As we can see in Figure 4, the optimality gap is really small (less than 1% on average), meaning that both heuristics achieve plans with very similar costs. GH$_i$ is able to compute the optimal plan in 11 out of the 20 problems. We see the biggest optimality gap in a problem with $t = 5$ (problem instance $\#10$), where there is a 0.93% optimality gap. When we translate the costs of this problem back into likelihood scores (see Figure 5), we get that the optimal plan likelihood is 0.27, while the likelihood of the plan returned by GH$_i$ is 0.24. On average, GH$_a$ returned plans with an

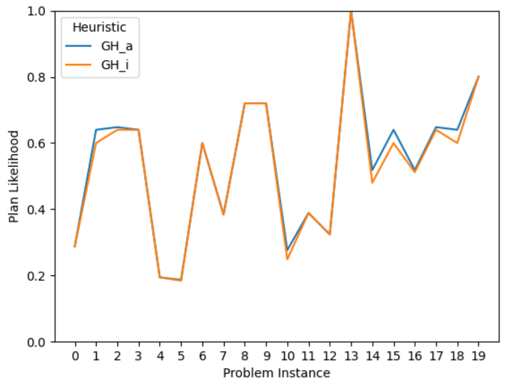


Figure 5: Plan likelihood as returned by GH$_a$ and GH$_i$ in problems of increasing difficulty, i.e., longer time horizon.

optimal likelihood score of 0.54, while GH$_i$ returned plans with a likelihood score of 0.52.

Conclusions and Future Work

We have proposed PFPT, a personal finance planning tool that offers financial advice at the individual level. The suggested financial plans achieve users' financial goals by maximizing the likelihood of being executed based on their financial habits. We model this problem from a search perspective and propose two different approaches to solve it: domain-independent automated planning and domain-dependent heuristic search. We evaluated both approaches in a set of financial problems with increasing complexity. Results showed that, as expected, while automated planning struggles to generate good plans in this domain, our suggested heuristics are able to scale on generating realistic financial plans.

Currently, our set of actions is limited to income increases and discretionary expenses decreases. We are exploring how to enrich the action space to include actions such as invest in different financial products that might yield different interest rates. We would also like to have the ability to impose arbitrary constraints to the generated plans. For example, users might want to see plans that do not suggest any income increase. Finally, we are currently assuming constant likelihood scores. In future work we would like to consider conditional likelihood scores, where the likelihood of executing one action depends on the previous actions.

Acknowledgements

This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or

sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

Althnian, A. 2021. Design of a rule-based personal finance management system based on financial well-being. *International Journal of Advanced Computer Science and Applications* 12(1).

Farinella, J.; Bland, J.; and Franco, J. 2017. The impact of financial education on financial literacy and spending habits. *International Journal of Business, Accounting, & Finance* 11(1).

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research* 20:61–124.

Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning with numerical expressions in lpg. In *Proceedings of the 16th European Conference on Artificial Intelligence*, 667–671.

Hanna, S. D., and Lindamood, S. 2010. Quantifying the economic benefits of personal financial planning. *Financial Services Review* 19(2).

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Irlam, G. 2018. Financial planning via deep reinforcement learning ai. *Available at SSRN 3201703.*

Irlam, G. 2020. Multi scenario financial planning via deep reinforcement learning ai. *Available at SSRN 3516480.*

Kindle, K. W.; Cann, R. S.; Craig, M. R.; and Martin, T. J. 1989. Pfps-personal financial planning system. In *IAAI.*

Peng, T.-C. M.; Bartholomae, S.; Fox, J. J.; and Cravener, G. 2007. The impact of personal finance education delivered in high school and college courses. *Journal of family and economic issues* 28(2):265–284.

Phillips, M. E.; Nielson, N. L.; and Brown, C. E. 1992. An evaluation of expert systems. *Journal of Financial Counseling and Planning* 3(1).

Warschauer, T., and Sciglimpaglia, D. 2012. The economic benefits of personal financial planning: An emperical analysis. *Financial Services Review* 21(3).