Resource-Constrained Planning: A Monte Carlo Random Walk Approach

Hootan Nakhost
University of Alberta
Edmonton, Canada
nakhost@ualberta.ca

Jörg Hoffmann
Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Martin Müller
University of Alberta
Edmonton, Canada
mmueller@ualberta.ca

Abstract

The need to economize limited resources, such as fuel or money, is a ubiquitous feature of planning problems. If the resources cannot be replenished, the planner must make do with the initial supply. It is then of paramount importance how *constrained* the problem is, i.e., whether and to which extent the initial resource supply exceeds the minimum need. While there is a large body of literature on numeric planning and planning with resources, such resource constrainedness has only been scantily investigated. We herein start to address this in more detail. We generalize the previous notion of resource constrainedness, characterized through a numeric problem feature $C \geq 1$, to the case of multiple resources. We implement an extended benchmark suite controlling C. We conduct a large-scale study of the current state of the art as a function of C, highlighting which techniques contribute to success. We introduce two new techniques on top of a recent Monte Carlo Random Walk method, resulting in a planner that, in these benchmarks, outperforms previous planners when resources are scarce (C close to 1). We investigate the parameters influencing the performance of that planner, and we show that one of the two new techniques works well also on the regular IPC benchmarks.

Introduction

Planning is the art of acting intelligently, thus a key aspect of it is the prudent consumption of resources. Indeed, planning with resources, and more generally numeric planning, is one of the most prominent topics in the planning literature (e.g., (Koehler 1998; Haslum and Geffner 2001; Fox and Long 2003; Hoffmann 2003; Gerevini, Saetti, and Serina 2003; Edelkamp 2003; 2004; Coles et al. 2008; Dvorak and Barták 2010)). Many applications of planning involve controlling autonomous agents with limited resources such as energy, fuel, money, and/or time.

Here, we investigate *consumable* resources (Haslum and Geffner 2001). These cannot be replenished, i.e., the planner must make do with the initial supply. That situation occurs quite frequently. Consider, for example, the energy supply in underwater robotics, the fuel supply in space travel, fixed project budgets, and fixed delivery deadlines.

We consider the special case where all resources are consumable. We will refer to this as *resource-constrained planning (RCP)*. This specific case is still relevant, but has been given scant attention in the literature (all existing approaches deal with much more general settings). In particular, only two previous studies (Hoffmann et al. 2007; Gerevini, Saetti, and Serina 2008) consider *resource constrainedness*: the amount by which the initial resource supply exceeds the minimum need. This can be measured in terms of a constant $C \geq 1$, namely the maximum number by which we can divide the resource supply without rendering the task unsolvable. The closer C is to 1, the more *constrained* is the problem; $C = 1$ enforces minimal resource consumption. Obviously, C links to the complexity of (approximate) optimization of resource consumption in the underlying domain. In practice, one would expect planning to become harder as C approaches 1. But what is the state of the art in this situation? Which techniques tend to work well, and which do not? Can we design tailored techniques without losing performance elsewhere?

The literature hardly answers these questions. In the International Planning Competition (IPC) benchmarks, C is not a controlled quantity. The single exception is IPC'11 NoMystery contributed as part of (an earlier stage of) this work. Only two previous works, namely the abovementioned works by Hoffmann et al. (2007) and Gerevini et al. (2008), run experiments controlling C. Each considers only one domain, with a single resource. Each runs a small selection of planners, drawing conclusions about which of these is most effective, but not about how we could design algorithms that work better.

We herein begin to address RCP in more detail. We generalize its investigation to domains with multiple resources. We extend the existing suite of RCP benchmarks with controlled C, introducing one new domain, and generalizing NoMystery to include more than one resource. The benchmarks and generators are publicly available. Controlling C requires domain-specific consumption-optimal solvers, to determine the minimum amount of resources needed. Hence these new benchmarks contribute considerable implementation work.

We conduct a large-scale study of the current state of the art as a function of C. Amongst other observations, we show that, despite all the new developments in satisficing planning, Hoffmann et al.'s and Gerevini et al.'s conclusion about the performance of planners using delete-relaxation heuris-

tics, which declines dramatically as C approaches 1, still holds. Together with the previously observed *resource persistence* in delete-relaxed plans (Coles et al. 2008) – which act as if what was once true will remain true forever – this motivates our attempt to emphasize local search. The working hypothesis is that adding more *exploration* of the search space, as opposed to *exploitation* of the heuristic, will make planners less susceptible to errors in that heuristic.

We design two improvements to the local search of *Arvand* (Nakhost and Müller 2009), a recent Monte Carlo Random Walk approach to planning. These improvements, called *smart restarts (SR)* and *on-path search continuation (OPSC)*, aim to improve Arvand's balance between exploitation and exploration, by trading off previous progress against the risk of repeating previous mistakes. We call the resulting planner Arvand2. In our RCP benchmark suite, Arvand2 almost universally outperforms all other planners when C is close to 1. This goes to show that algorithmic improvements *are* possible in this specific situation.

To verify whether this improvement comes at the price of performance losses in other planning domains, we run tests on the IPC'11 benchmarks. On-path search continuation can be detrimental, while smart restarts improve Arvand's performance there as well. We study the parameters influencing Arvand2's performance, and the effect of interactions between multiple resources.

We next define RCP and summarize the relevant literature. We then discuss resource constrainedness C, and describe our RCP benchmark suite. We explain our enhancements to Arvand, report our experiments, and conclude.

The present paper significantly extends the abstract published in (Nakhost, Hoffmann, and Müller 2010).

Planning with Resources

We define our RCP formalism, and summarize the most relevant prior work in the area. We outline the role of planning with resources in the IPC benchmarks.

Planning Formalism

A STRIPS *planning task* is a tuple (P, I, G, A) with a set of propositions P, *initial state* $I \subseteq P$, *goal* $G \subseteq P$, and a set of *actions* A. Each $a \in A$ is a triple (pre_a, add_a, del_a) of subsets of P. A *state* is identified with the set of propositions $s \subseteq P$ that are true in the state. Action a is *applicable* to s if $pre_a \subseteq s$; the result of executing a is $(s \setminus del_a) \cup add_a$. A *plan* is a sequence of actions in whose iterative execution from I all actions are applicable, ending in a state $s \supseteq G$.

In *resource-constrained planning (RCP)*, STRIPS planning tasks are extended with a set R of resource identifiers as well as functions $i : R \mapsto \mathbb{Q}_{\geq 0}$ and $u : A \times R \mapsto \mathbb{Q}_{\geq 0}$. $i(r)$ denotes the initial amount of resource $r \in R$, and $u(a, r)$ is the amount of r consumed when executing action a. Sufficient resource availability is requested by additional *resource preconditions* taking the form $s(r) \geq u(a, r)$.

Clearly, RCP is related to optimization of resource consumption, as a decision problem. But it is not, in general, the same thing in practice. In many applications, the primary objective is to optimize some other criterion, such as timespan or amount of data collected. The resources then only serve to encode a fixed budget that the plan has to make do with. This is the situation we aim at addressing here.

Previous Work

RCP is a special case of planning with resources, which in general allows resource production in addition to consumption. In turn, planning with resources is a special case of numeric planning. The latter was emphasized in the 2002 IPC (Fox and Long 2003). A prominent line of planners (Hoffmann 2003; Edelkamp 2003; Gerevini, Saetti, and Serina 2003; Edelkamp 2004; Gerevini, Saetti, and Serina 2008) handling this uses direct extensions of delete-relexation heuristics, via "numeric relaxed planning graphs". As observed by Coles et al. (2008), this kind of heuristic suffers from *resource persistence*. Relaxed plans act as if resource values persist forever, and are therefore fundamentally unsuited for reasoning about resource consumption.

Considering not general numeric planning, but planning with resources more specifically, Coles et al. address resource persistence in their LP-RPG planner. They extend relaxed planning with more informed numeric reasoning via linear programming (LP). In a nutshell, the numeric relaxed planning graph is encoded into MILP, and the LP relaxation is used to provide informed upper and lower bounds for each resource. As Coles et al. point out, this is useful to avoid detrimental phenomena arising from the interaction between resource producers and consumers. However, as our experiments will show, for tightly constrained RCP this more informed heuristic is unfortunately still not good enough.

Other work on planning with resources makes use of very different heuristics, based on Graphplan (Koehler 1998) and the h^m family (Haslum and Geffner 2001). The Filuta system (Dvorak and Barták 2010) uses dedicated reasoners to resolve resource conflicts during a plan-space search; there is no heuristic guidance estimating resource consumption.

Regarding local search, obviously the system most closely related to ours is the one we extend here, Arvand (Nakhost and Müller 2009). This relies on the original FF heuristic (Hoffmann and Nebel 2001), but does not compute it on every state. Instead, Arvand runs random walks to define a broader search neigborhood. The heuristic is computed only at the end of each sample, making the sampling very fast and thus allowing a large amount of exploration.

LPG (Gerevini, Saetti, and Serina 2003) performs local search in plan space. A recent version of LPG (Gerevini, Saetti, and Serina 2008) is dedicated to general numeric planning. Apart from our focus on the much more specific RCP problem, the main difference to our work is that every node in LPG's direct search neighborhood is immediately evaluated. Thus LPG puts less emphasis on exploration than Arvand and the Arvand2 planner we propose here.

Identidem (Coles, Fox, and Smith 2007) employs random samples during FF's "enforced hill-climbing" search (Hoffmann and Nebel 2001). The motivation is to escape local minima more effectively, in a STRIPS planning setting not specific to planning with resources. Like for LPG, the main technical difference to Arvand and Arvand2 is that, on the random samples, every state is evaluated with the heuristic.

IPC Benchmarks

Many IPC benchmarks incorporate planning with resources in some form. However, only few of these are RCP domains as considered here, namely: IPC'98 *Mystery* and *Mprime* (fuel); IPC'02 *Satellite* (fuel); IPC'06 *TPP* (money) and *Trucks* (time, i.e., strict delivery deadlines); IPC'11 *NoMystery* (fuel). IPC'02 *Rovers* has energy consumption, but includes also a "recharge" operator.

Several other IPC domains feature resource consumption, yet impose it as an optimization criterion, not as a hard constraint. Since this does not force satisficing planners to make do with a given budget, it is quite different from the situation we are interested in here. That said, anytime planners may incrementally reduce resource consumption, and eventually bring it below the thresholds required. We evaluate this option, using the latest version of LAMA (Richter, Helmert, and Westphal 2008), in our experiments.

Resource Constrainedness

We formalize RCP constrainedness in terms of a parameter C. We describe our benchmark suite controlling C, and summarize previous findings from doing such control.

Characterizing Resource Constrainedness

For the case of a single resource, $R = \{r\}$, considered in previous work (Hoffmann et al. 2007; Gerevini, Saetti, and Serina 2008), defining *resource constrainedness* C is straightforward. C should measure the factor by which the initial resource supply, $i(r)$, exceeds the minimum need. This can be derived from the equation $\frac{i(r)}{C} = M$, where M is the minimal resource consumption of any plan for the task.

In case there are several resources, matters are not that simple since there is no unique "minimum need". A small amount of resource r might be compensated for by a big amount of resource r'. In other words, to define a unique value M, we would need to aggregate resource values (e.g., by their sum or maximum). However, there is no one aggregation method that is adequate across all possible domains. The solution we propose is to define C based on the notion of pareto-minimality. Reformulating $\frac{i(r)}{C} = M$ to $\max\{C \mid \frac{i(r)}{C} \geq M\}$, we observe that the above definition corresponds to downscaling $i(r)$ until it hits the pareto frontier given by the single pareto-minimal solution M. We generalize this simply by downscaling the whole vector i until it hits the more general pareto frontier.

For assignments $M, M' : R \mapsto \mathbb{Q}_{\geq 0}$, we write $M \geq M'$ to denote pointwise domination. M is *pareto-minimal* if: (a) the task is solvable when setting $i := M$; and (b) for any M' where $M \geq M'$, and $M(r) > M'(r)$ for at least one $r \in R$, setting $i := M'$ renders the task unsolvable. Denoting by $\mathcal{M}$ the set of all pareto-minimal assignments, we define C as $\max\{C \mid \exists M \in \mathcal{M} : \frac{i(r)}{C} \geq M\}$. In other words, C is the largest factor by which we can downscale the initial resource supply without rendering the task unsolvable.

Computing C for a given planning task is, obviously, hard. Such computation may be useful (e.g., to configure planners), but is not our focus here. Instead, we wish to design benchmarks controlling C, in order to investigate how planning algorithms scale in that parameter. Our methodology for doing so is to implement domain-specific solvers computing $\mathcal{M}$, i.e., all pareto-minimal resource supplies. Benchmark instances with resource constrainedness C are obtained by selecting some $M \in \mathcal{M}$, and setting $i := C \times M$.

RCP Benchmark Domains Controlling C

Given the need to develop domain-specific consumption-optimal solvers, it is not easy to implement benchmarks controlling C. Our RCP benchmark suite currently consists of three domains, namely *NoMystery*, *TPP*, and *Rovers*. The generators as well as all test instances used here are available at http://code.google.com/p/rcp-benchmark-suite/.

TPP is the same domain as used in IPC'06. An agent with a given budget needs to buy a set of products from different markets, selling at different prices. The resource is money. Gerevini et al. (2008) implemented a generator allowing to control C, however that generator is not available anymore. Our test suite contains the original instances.

NoMystery is the same domain as used in IPC'11. A set of packages must be transported between nodes in a graph; actions move trucks along edges, or load/unload packages. Each truck has its own fuel supply, which it consumes when moving. In the original generator we provided for IPC'11, instances were restricted to have only a single truck (and thus only a single resource). Our extended generator removes this restriction. Our test instances contain two trucks, which is already quite challenging to solve for the domain-specific optimal solver as well as for state of the art domain-independent planners.

Rovers is the domain used in IPC'02, except that we removed the "recharge" operator to fit the domain into the RCP framework. The goal is to take a number of rock samples and images, and transfer them to a lander. Each rover has an energy supply, and all actions consume energy. We implemented a generator from scratch, allowing an arbitrary number of rovers. For the same reasons as in NoMystery, our test instances contain two rovers.

In all three domains, the resource amounts in our test instances (initial supply and per-action consumption) are integer, since Arvand2 does not handle numeric variables. The integer values are encoded into STRIPS in the straightforward fashion, using one proposition per possible value. Arvand2 is not even explicitly aware of the resources. As we will see, its performance is very good despite this. Every other planner in our experiments is supplied with the encoding leading to best performance.

Previous Findings when Controlling C

In previous experiments controlling C, Hoffmann et al. (2007) use an older version of the NoMystery domain, whose generator contained bugs, while Gerevini et al. (2008) use the TPP benchmark described above. Both experiments run satisficing heuristic search planners using relaxation heuristics: different versions of FF (Hoffmann 2003), LPG (Gerevini, Saetti, and Serina 2003), SGPlan, and Mips

(Edelkamp 2003). Both observe that the performance of all these planners degrades dramatically as C approaches 1, yet less so for LPG than for the other planners. The latter observation is part of the motivation for our work. The similarity of the heuristics employed – which as observed by Coles et al. (2008) are not informative here – suggests that the advantage of LPG is mainly due to its local search paradigm. We show that one can push the performance margins far higher still, based on highly explorative Monte Carlo search.

Improving Local Search

Arvand (Nakhost and Müller 2009) is a successful stochastic planner which evaluates a heuristic function at the endpoints of random walks. Arvand benefits from a good heuristic, and can also overcome misleading heuristic values to some extent, due to the exploration made by the random walks.

Starting at an initial state, at each *search step* Arvand selects its next state from among all evaluated sample states in a local neighborhood. Each sample is the endpoint of a bounded length random walk, consisting of a sequence of randomly selected applicable actions. After executing a number of random walks, Arvand transitions to an endpoint whose heuristic value under the FF heuristic, h^{FF}, is minimal. Ties are broken randomly. Typically, computing h^{FF} is orders of magnitude more costly than executing a random action. Therefore Arvand is able to sample from a much larger and more diverse neighborhood than planners that focus only on exploiting the heuristic. Arvand searches until either the goal is reached, or the search gets stuck. The latter happens if either the best seen heuristic value does not improve over several search steps, or if all random walk endpoints are *dead ends*: states s with no applicable actions or with $h^{FF}(s) = \infty$. When the search gets stuck, it restarts from the initial state I, beginning a new *search episode*.

The Arvand2 planner introduces two improvements on Arvand which work especially well for RCP: On-Path Search Continuation (OPSC) and Smart Restarts (SR).

On-Path Search Continuation

Arvand uses what we call *End-Point Search Continuation* (EPSC) in this paper: after selecting a sampled endpoint e, Arvand commits to all the actions on the path to e. The drawback for RCP is that, if some of the random actions leading to e consume too many resources and the problem becomes unsolvable, then all search effort from this point until the next restart is wasted. This can cause severe search inefficiencies when resources are tightly constrained.

The new technique of *On-Path Search Continuation* (OPSC) avoids commitment to all actions leading to e, while still benefiting from the guidance of the selected path to e. Let $S = \{s_0, s_1, \ldots, s_k\}$ be the states visited along the path from the initial state $s_0 = I$ to the current endpoint $s_k = e$. Arvand's EPSC chooses s_k as the starting point of all random walks within the current search step. OPSC generalizes this by choosing, for each individual random walk, the starting point according to some fixed probability distribution over S. In all the experiments reported here, OPSC uses a uniform probability distribution over S. The method for updating S is analogous to EPSC. In each search step, after running the random walks, OPSC selects an endpoint e with minimal h^{FF} value. S is then changed to the path from I to e.

Smart Restarts

The second innovation in Arvand2 is a modified restarting strategy. Arvand restarts from scratch in every new search episode, discarding all previous information. However, previous episodes may contain valuable partial paths, which ultimately ended in failure only because of bad actions later on. *Smart Restarts (SR)* in Arvand2 try to preserve information by maintaining a fixed-capacity *pool* of the *most promising episodes* so far.[1] Smart restarts begin at a state along the trajectory of such an earlier episode, instead of returning to I every time. The method selects a random episode from the pool, then selects a random state along that episode as the next restarting point. If the pool is empty, it selects I.

Let p denote the fixed *pool capacity*, i.e., the maximum number of episodes stored in the pool. The "worst" episode in the pool is replaced whenever the pool is full and a new "better" episode is discovered. The quality of an episode is defined as the smallest h^{FF} value along its trajectory $\langle s_0, \ldots, s_n \rangle$. Let s_{min} be the earliest state among those states with minimum heuristic value. Then the partial trace $\langle s_0, \ldots, s_{min} \rangle$ is the candidate for inclusion into the pool. The motivation for defining s_{min} as the *earliest* such state is that the amount of resources consumed grows monotonically along an episode.

Restarting from a pool of episodes balances exploration and exploitation, and therefore can increase the chance to quickly reach promising regions of the search space. If the pool capacity p is small, then the search is more greedy and concentrates on the hitherto best traces. Large pools lead to more exploration, which intuitively might improve performance on hard instances given sufficient time. In contrast, with limited time, more focus on exploitation should be beneficial.

Arvand2 does not use smart restarts until an initial number N of search episodes has been completed. This avoids a heavy bias towards these early episodes.

Experiments

Based on initial tests, we set $N = 50$ and $p = 50$ in Arvand2. All parameters inherited from Arvand are set to their default values, i.e., we did not fine-tune these. We run tests on the three RCP domains NoMystery, TPP and Rovers, as well as on IPC-2011 domains, on a 2.5 GHz machine with 2 GB memory limit. The runtime cut-off was set to 30 minutes for small instances, and to 40 minutes for large instances (see below). Results for the randomized planners LPG, Arvand, Arvand2 and their variants are averaged over 10 runs per instance.

For the RCP domains, we used several different encodings. The first uses propositions to represent the integer-

[1] Solution-guided search (Heckman and Beck 2011)) is a successful related algorithm in Scheduling. It differs significantly in context and technical details.

valued resource levels; the second uses numeric variables. For tasks with a single resource r, we created cost-augmented variants of both, setting the cost of each action a to its consumption $u(a, r)$. For LAMA, we also supplied an anytime cost-augmented variant, requiring to minimize consumption of r, removing the resource preconditions $s(r) \geq u(a, r)$ but counting a task as solved only if the returned plan satisfied these. To exploit the ability of LPRPGP of handling preferences, we also used an encoding where the resource preconditions are changed to preferences and a unit cost is assigned to each preference violation; therefore, reducing the cost of violations translates to decreasing the number of actions that over-consume the resources. For each (planner, domain) pair, we show data for the most effective encoding for that pair.

A wide range of planners was tested, including Arvand and Arvand2, as well as two variants Arvand2(SR) and Arvand2(OPSC) which use only one of the two new techniques; FF (Hoffmann and Nebel 2001), the top performer at IPC'00; LPG (Gerevini, Saetti, and Serina 2003), the top performer at IPC'02; Fast Downward Autotune 1 (FD-AT1) and Fast Downward Autotune 2 (FD-AT2) (Helmert 2006), recent versions of the top performer at IPC'04, whose parameters were optimized on IPC benchmarks; LAMA2011 (Richter, Helmert, and Westphal 2008), the latest version of the top performer at IPC'08 and IPC'11; the recent SAT-based planners M and Mp (Rintanen 2010) which are very competitive across many domains, and do not suffer from the weakness of relaxation heuristics in planning with resources; and LPRPGP (Coles et al. 2008; Coles and Coles 2011), the most recent version of the LPRPG planner whose improved heuristic addresses that weakness.

We also ran several optimal planners, to check whether these are more effective for small C, as was previously observed by Hoffmann et al. (2007) for their step-optimal SAT-based numeric planner, num2sat, in a NoMystery test suite. Other than this, a comparison to the satisficing planners is not intended and should not be made. We include num2sat as well as: Merge-and-Shrink (M&S) (Helmert, Haslum, and Hoffmann 2007; Nissim, Hoffmann, and Helmert 2011), a state abstraction heuristic; LM-cut (Helmert and Domshlak 2009), the best known admissible relaxation heuristic; Selmax (Domshlak, Karpas, and Markovitch 2010), which uses machine learning to selectively choose a heuristic per-state; and the optimal version of Fast Downward Autotune (FD-AT-OPT) (Fawcett et al. 2011).

Our RCP benchmark set consists of 450 instances. These are obtained from a smaller set of "base" instances, where $C = 1.0$. The benchmarks with larger values of C are generated by increasing the initial resource supply in these base instances. This setup ensures that the results across different C values are exclusively due to the level of resource constrainedness. The instances in TPP are the ones originally designed by Gerevini et al. (2008). The 5 base instances each have 1 agent, 8 markets and 8 products, and each is modified to obtain instances with $C = 1.1, \ldots, 1.5$. In each of NoMystery and Rovers, to demonstrate scaling, we designed a "small" and a "large" group. The small groups have 2 resources and 25 base instances, the large groups have a single resource and 5 base instances. In the large groups, having several resources was not feasible for the optimal solvers implemented inside the generators. Each base instance is modified to obtain instances with $C = 1.1, \ldots, 1.5, 2.0$. Small NoMystery instances contain 2 trucks, 9 locations, and 9 packages; large ones use 1 truck, 12 locations, and 15 packages. Small Rovers instances feature 2 rovers, 11 locations, and 16 objectives; large ones include 1 rover, 15 locations, and 20 objectives.

The base instances were generated randomly, except that in the small groups – where there are several resources – we explored a second interesting problem feature, namely the *distribution* of the resource budget. We first generated 5 instances randomly. Then, we selected 5 pareto-optimal resource allocations for each of these, yielding 25 base instances. The selection was made so that we included: an allocation in which the total amount of resources is assigned to one of the trucks/rovers (this is analogous to a problem with one resource); an allocation that is closest to where the resources are evenly distributed between trucks/rovers; and 3 others in between these two extremes.

Figure 1 summarizes coverage results in the RCP benchmarks. Missing data indicates that a planner did not solve any instance for that domain; the single exception is Arvand2(OPSC), not shown in Figure 1 (a,b,c) because there it almost coincides with Arvand2. The most effective encoding for each planner is: numerical encoding - FF, M, Mp, LPG, LPRPGP, and num2sat; propositional encoding - LM-cut, M&S, Selmax, and FD-AT-OPT. propositional for small domains, propositional + costs for large domains and TPP - LAMA, FD-AT1, FD-AT2, Arvand, and all Arvand2 variations. Exceptions: For LAMA in large-rovers and TPP, propositional + costs with no hard constraints was best. The main observations are:

(i) Optimal planners can be effective for scarce resources, but only in small instances.

(ii) The improved heuristic of LPRPGP is not sufficient to obtain competitive performance here.

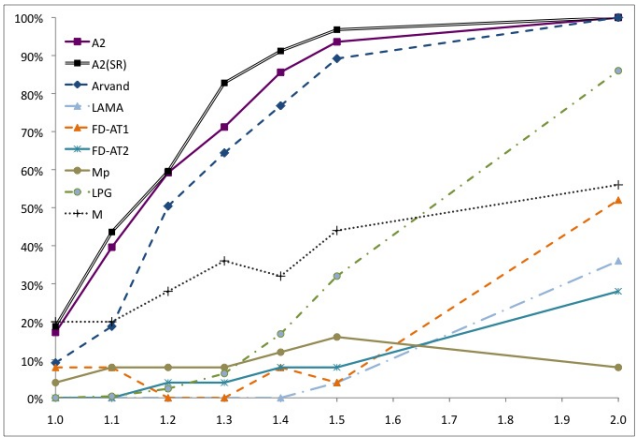
(iii) Current satisficing planners excel when resources are plentiful, but are very limited when these get scarce. The same holds for M and Mp.

(iv) Stochastic local search can be a powerful tool to attack RCP. In particular, Arvand2 almost universally outperforms other planners when C is close to 1.

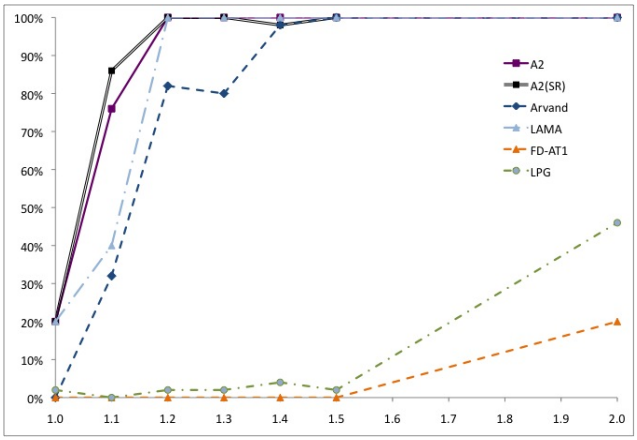
(v) Arvand2 has a consistent and significant advantage over Arvand, showing the effectiveness of smart restarts and on-path search continuation.

To see (i), note that none of the optimal planners solved any instance, in any domain other than small NoMystery. In the latter domain – compare Figures 1 (e) and (c) – optimal planners are more effective than satisficing planners, when C is close to 1. For M&S, the value of C has little effect. All optimal planners fail as instance size increases (Figure 1 (d)).

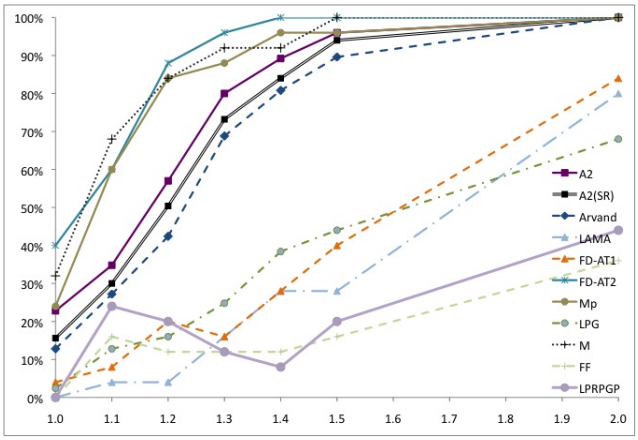
Regarding point (ii), LPRPG does not solve any instance in Rovers, and is quite weak also in NoMystery and TPP. Point (iii) is obvious in all the plots showing data for satisficing planners (Figure 1 (a,b,c,d,f)). For the heuristic planners,



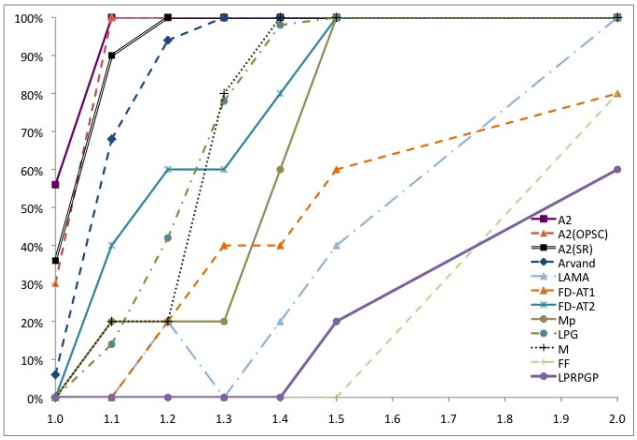
(a) Rovers, small



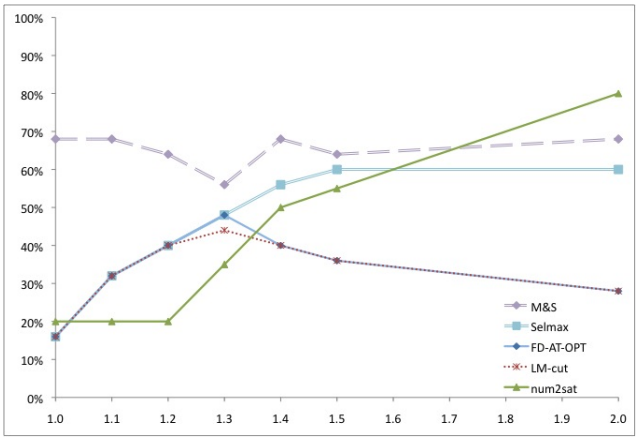
(b) Rovers, large



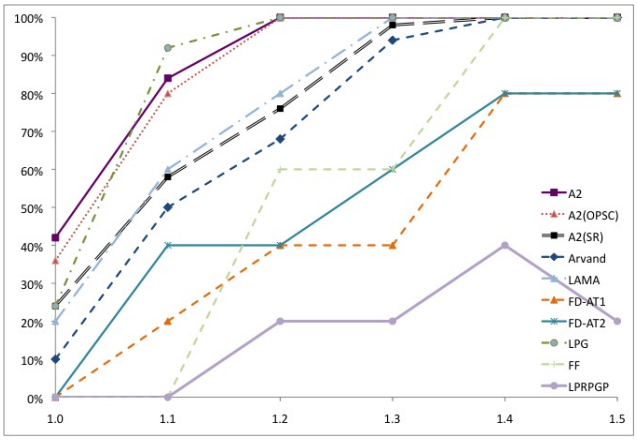
(c) NoMystery, small



(d) NoMystery, large



(e) NoMystery, small (optimal planners)

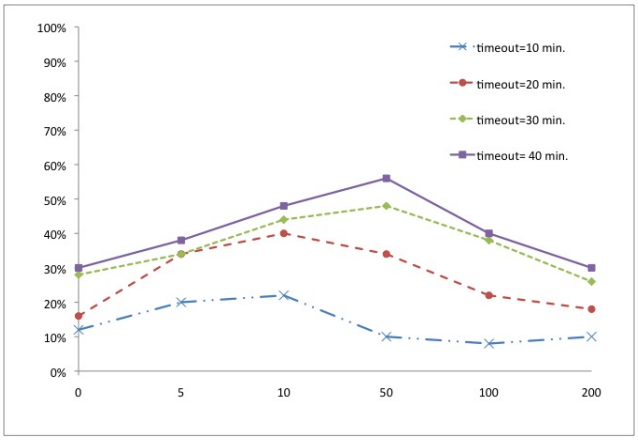


(f) TPP

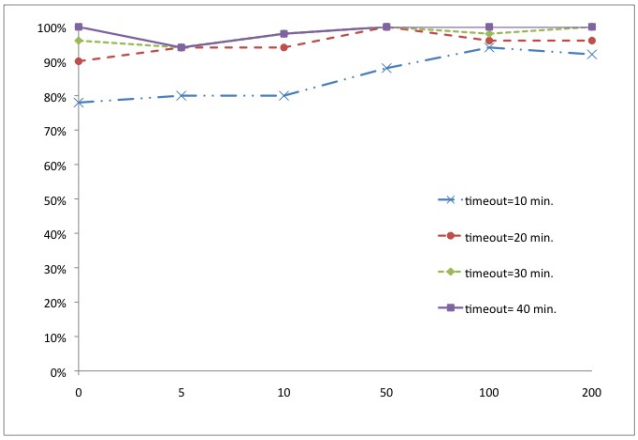
Figure 1: Coverage of planners over resource constrainedness C, in (a,b) Rovers, (c,d,e) NoMystery, and (f) TPP. Randomized planners are run 10 times per instance, where each run counts separately towards coverage.

this can be expected given the pitfalls of relaxed planning and was previously observed in much smaller experiments by Hoffmann et al. (2007) and Gerevini et al. (2008). For M and Mp, it is quite interesting that their behavior over C is similar to that of the heuristic planners. It is not clear to us what causes this behavior; a plausible explanation could be that these planners, too, act in a rather greedy way.

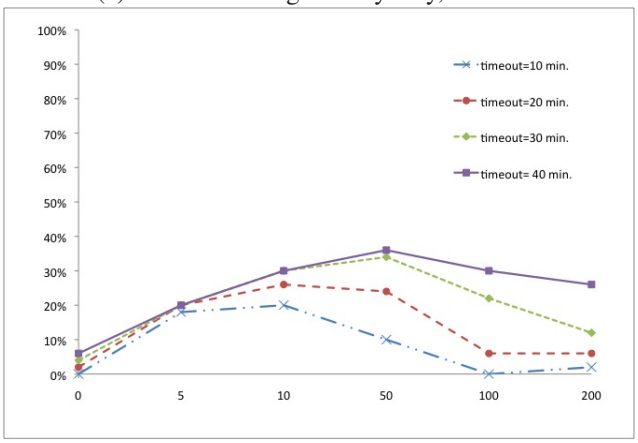
Point (iv) is evident from the results of Arvand2 and Arvand, in all the domains. With small C, Arvand2 vastly outperforms all other planners, by factors of 6 and more in coverage. The only exceptions are LAMA in large Rovers (Figure 1 (b)); FD-AT2, M, and Mp in small NoMystery



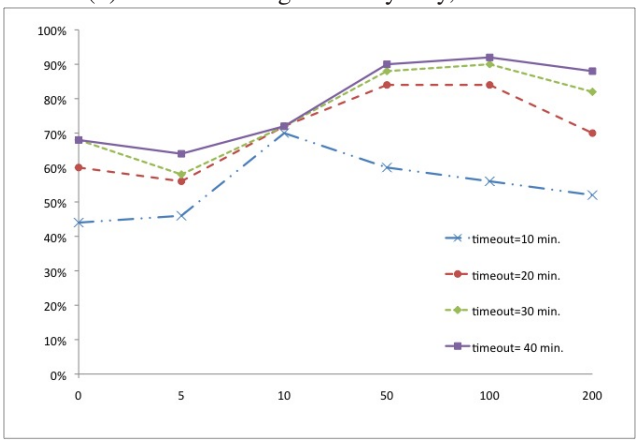
(a) Arvand2 in large NoMystery, $C = 1.0$



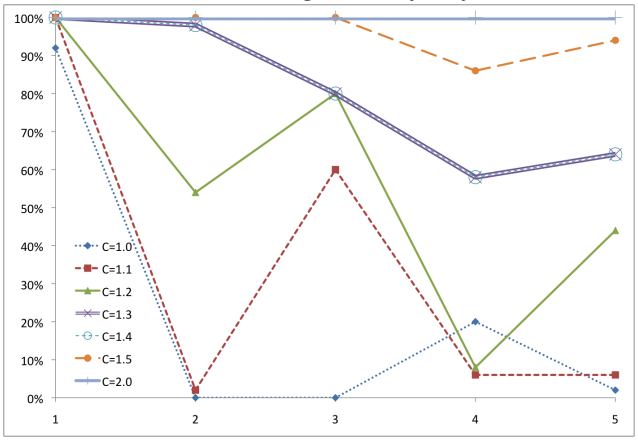
(b) Arvand2 in large NNoMystery, $C = 1.1$



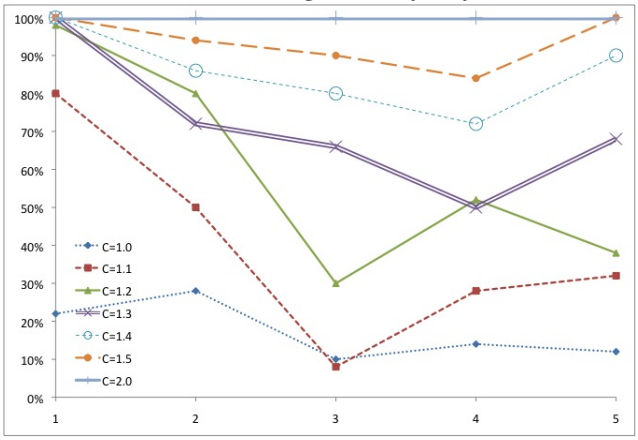
(c) Arvand2(SR) in large NNoMystery, $C = 1.0$



(d) Arvand2(SR) in large NNoMystery, $C = 1.1$



(e) Arvand2 in small NoMystery



(f) Arvand2 in small Rovers

Figure 2: Coverage of Arvand2 as a function of different parameters. (a,b,c,d) vary the runtime cut-off, and the pool size p for smart restarts (x-axis; for $p = 0$, smart restarts are turned off). (e) and (f) vary the distribution of budget across resources (see text).

(Figure 1 (c)); and LPG in TPP (Figure 1 (f)). Of these, LAMA's prowess in large Rovers is due to the anytime cost-augmented encoding, and does not extend to small Rovers (Figure 1 (a)) with several resources. FD-AT2, M, and Mp fall behind significantly in large NoMystery (Figure 1 (d)). LPG is competitive with Arvand2 only in TPP.

Point (v) is also evident. In fact, in these benchmarks, neither smart restarts nor on-path search continuation ever hurt performance when added to Arvand. Their effectiveness does depend on the domain, though. In Rovers, it is

Domain	Arvand		Arvand2(SR)		Arvand2(OPSC)		Arvand2		LAMA		FD-AT1		FD-AT2		M		Mp		LPRPGP	
	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T
barman	0%	1800	0%	1800	0%	1800	0%	1800	100%	5	100%	256	0%	1800	0%	1800	0%	1800	10%	1669
elevators	100%	12	100%	12	25%	1414	20%	1478	100%	47	95%	173	90%	340	5%	1735	65%	633.94	80%	637
floortile	5%	1783	10%	1687	10%	1626	5%	1749	25%	1372	20%	1452	45%	996	0%	1800	0%	1800	10%	1666
nomystery	95%	164	95%	142	95%	194	100%	54	50%	900	45%	1023	80%	381	80%	561	75%	456.71	35%	1192
openstacks	100%	26	100%	24	100%	92	100%	60	100%	44	95%	289	50%	1128	0%	1800	0%	1800	85%	546
parcprinter	100%	8	100%	8	100%	21	100%	20	100%	0	100%	0	85%	273	100%	0	100%	0.09	35%	1171
parking	15%	1778	15%	1648	0%	1800	0%	1800	95%	408	65%	1097	75%	926	0%	1800	0%	1800	35%	1561
pegsol	100%	21	100%	63	100%	5	100%	7	100%	2	100%	0	100%	14	85%	330	100%	3.97	100%	82
scanalyzer	85%	271	90%	207	85%	276	85%	286	100%	21	100%	102	90%	202	50%	937	80%	362.39	90%	192
sokoban	10%	1635	10%	1665	5%	1711	10%	1705	90%	322	95%	129	85%	464	0%	1800	10%	1621.62	45%	1180
tidybot	85%	385	85%	335	80%	681	85%	606	80%	392	70%	584	75%	753	0%	1800	30%	1268.48	95%	221
transport	65%	651	70%	569	35%	1219	35%	1243	85%	523	75%	634	70%	705	0%	1800	5%	1711.42	0%	1800
visitall	65%	28	70%	40	65%	662	65%	633	100%	42	10%	1621	40%	1128	0%	1800	0%	1800	20%	1448
woodworking	100%	102	100%	78	5%	1710	15%	1602	100%	12	100%	16	65%	647	100%	1	100%	1.27	0%	1800
total	66%	619	68%	591	50%	944	51%	932	88%	292	76%	527	68%	697	30%	1283	40%	1076	46%	1083

Table 1: Coverage (C) and average runtime (T; in seconds) in the IPC'11 benchmark domains. For unsolved instances, the time limit of 30 minutes is inserted into the computation of T.

slightly better to use smart restarts only (as pointed out, Arvand2(OPSC) almost coincides with Arvand2 there). In NoMystery, both techniques contribute to the improvement over Arvand. In TPP, smart restarts have a beneficial effect but on-path search continuation is more important by far.

For smart restarts, an influential parameter is the pool capacity p. Two observations are clear from the data in Figure 2 (a,b,c,d):

(vi) Pool size induces a sweet-spot behavior in both Arvand2 and Arvand2(SR), especially for small values of C.

(vii) As the time-out increases, the sweet-spot behavior tends to become more pronounced, and the best value for p tends to become larger.

An intuitive explanation for both is that, as previously discussed, p controls the exploitation-exploration trade-off in smart restarts. Larger pools yield a more explorative search, which may pay off by solving more instances – but only if there is enough time. With limited runtime, a more greedy search may succeed more often.

Figure 2 (e,f) examines the performance of Arvand2 as a function of the distribution of the resource budget. On the x-axis, we distinguish the 5 qualitatively different pareto-optimal resource allocations described above: $x = 1$ stands for an allocation assigning the whole budget to just one of the two resources, whereas $x = 5$ stands for an allocation assigning the budget as evenly as possible, i.e., minimizing the difference between the initial resource supplies. In between, we interpolate such that this difference decreases monotonically with growing x. Each value of x corresponds to 5 base instances, and the data shown are averages. The data is noisy, but still allows to observe:

(viii) Distributing the resource budget more evenly results in worse performance, especially with small C.

An intuitive explanation is that a more even distribution of the budget implies that the planner needs to reason more about which resource to use.

Finally, we ran our new planners on the standard IPC'11 benchmarks, to cross-check whether their superior performance in RCP is bought at the price of deteriorated performance in other settings. All the planners were run under IPC'11 conditions: 2GB memory and 30 minutes runtime; Table 1 confirms that smart restarts work well on IPC benchmarks – Arvand2(SR) has better coverage than Arvand. On-path search continuation, by contrast, can be detrimental. The reduction in coverage stems mainly from 4 of the 14 domains.

Conclusion

We all must consume our resources prudently, and so must planners in a multitude of applications. While this general issue has long been researched, not much has been done to specifically address the RCP situation where resources are scarce and cannot be replenished. Starting to investigate this more carefully, we have shown that state of the art planners typically behave very badly. We have demonstrated the potential of local search to improve this, and contributed an extended test base for future research.

To further improve local search, one might try to aggressively exploit an explicit encoding of resources. The danger here probably would be to not over-fit the algorithms and lose too much performance elsewhere. Apart from this, important topics for future research include: understanding the behavior of various algorithms, such as M and Mp, better; investigating whether approaches other than local search can be tailored to the RCP setting; investigating to what extent we can devise automatic configuration methods for deciding whether or not to switch these tailored techniques on. We hope these will inspire other researchers as well.

Acknowledgments. Work performed while Jörg Hoffmann was employed by INRIA, Nancy, France. Hootan Nakhost and Martin Müller were supported by NSERC and iCORE.

References

Coles, A., and Coles, A. 2011. LPRPG-P: Relaxed plan heuristics for planning with preferences. In *Proc. ICAPS'11*.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A hybrid relaxed planning graph - LP heuristic for numeric planning domains. In *Proc. ICAPS'08*, 52–59.

Coles, A.; Fox, M.; and Smith, A. 2007. A new local-search algorithm for forward-chaining planning. In *Proc. ICAPS'07*, 89–96.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To max or not to max: Online learning for speeding up optimal planning. In *Proc. AAAI'10*, 1071–1076.

Dvorak, F., and Barták, R. 2010. Integrating time and resources into planning. In *Proc. ICTAI'10*, 71–78.

Edelkamp, S. 2003. Taming numbers and durations in the model checking integrated planning system. *JAIR* 20:195–238.

Edelkamp, S. 2004. Generalizing the relaxed planning heuristic to non-linear tasks. In *Proc. KI'04*, 198–212.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Automated configuration of Fast Downward. IPC 2011 planner abstracts.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *JAIR* 20:239–290.

Gerevini, A.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *AI* 172(8-9):899–944.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. ECP'01*, 121–132.

Heckman, I., and Beck, J. C. 2011. Understanding the behavior of solution-guided search for job-shop scheduling. *Journal of Scheduling* 14(2):121–140.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS'09*, 162–169.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. ICAPS'07*, 176–183.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J.; Kautz, H.; Gomes, C.; and Selman, B. 2007. SAT encodings of state-space reachability problems in numeric domains. In *Proc. IJCAI'07*, 1918–1923.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291–341.

Koehler, J. 1998. Planning under resource constraints. In *Proc. ECAI'98*, 489–493.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In *Proc. IJCAI'09*, 1766–1771.

Nakhost, H.; Hoffmann, J.; and Müller, M. 2010. Improving local search for resource-constrained planning. In *SOCS-10*, 81–82.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *Proc. IJCAI'11*, 1983–1990.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI'08*, 975–982.

Rintanen, J. 2010. Heuristics for planning with SAT. In *Proc. CP'10*, 414–428.