Concurrent Planning and Execution Using Dispatch-Dependent Values

Andrew Coles[1] , **Erez Karpas**[2] , **Eyal Shimony**[3] , **Shahaf Shperberg**[3] , **Wheeler Ruml**[4]
[1]King's College London, UK
[2]Technion, Israel
[3]Ben-Gurion University, Israel
[4]University of New Hampshire, USA
andrew.coles@kcl.ac.uk, karpase@technion.ac.il
shimony@cs.bgu.ac.il, shperbsh@bgu.ac.il, ruml@cs.unh.edu

Abstract

Agents operating in the real world must cope with the fact that time passes while they plan. In some cases, such as under tight deadlines, the only way for such an agent to achieve its goal is to execute an action before a complete plan has been found. This problem is called Concurrent Planning and Execution (CoPE). Previous work on CoPE relied on a value function that assumes search will finish before actions are executed, causing the agent to be overly pessimistic in many situations. In this paper, we define a new value function that takes into account the agent's ability to dispatch actions incrementally. This allows us to devise a much simpler algorithm for concurrent planning and execution. An experimental evaluation on problems with time pressure shows that the new method significantly outperforms the previous state-of-the-art.

1 Introduction

Concurrent planning and execution (CoPE) is crucial in scenarios where meeting strict time constraints is essential, even when the environment operates on a predictable schedule. For example, consider an automated system tasked with navigating a train network to deliver a package by 3 PM. The train schedules are fixed and known in advance. Missing a train means waiting for the next one, which could result in missing the deadline. The challenge is that generating a valid plan—one that guarantees the package will arrive on time—can be computationally demanding. If the system waits to generate the entire plan before starting execution, it risks running out of time to make the necessary connections. By planning and executing concurrently, the system can begin dispatching actions based on a partial plan, continuously refining its strategy to ensure the delivery is completed by the deadline. This approach is particularly effective in time-sensitive scenarios, enabling the system to respond flexibly and efficiently within the given constraints.

Previous work on CoPE [Coles *et al.*, 2024] built upon earlier research in situated temporal planning (sometimes abbreviated as situated planning) [Shperberg *et al.*, 2021; Cashmore *et al.*, 2018], in which time passes during planning, but we can only start executing after we have a complete plan. Specifically, the method used to estimate the value of a search node for situated planning was also used to guide the search in CoPE. However, this estimator is based on the assumption that search needs to find a complete plan before the first action can be executed — a valid assumption in situated planning, but not in CoPE. To compensate for this, the previous CoPE method relied on a rather ad-hoc scheme, where an action is dispatched only if a certain number of nodes have been explored in the subtree under that action.

We present a new approach for CoPE that is based on the insight that there is a big difference between the meta-level search space in situated planning and in CoPE. In situated planning, it is the same as the search space for the offline planning problem: the only decision is which node to expand next. While the decision problem in situated planning is more difficult than in offline planning (as it has to consider not just expected search effort but also deadlines), the number of possible decisions is still the same. In CoPE, for each possible node in the search tree, we have two possible decisions: expand it or execute it. Ideally, we would like to estimate the values that would be obtained by the agent if each of these decisions were taken.

At first blush, this appears hopeless: for a given possible future world state, there might be many different search and execution interleavings that will achieve it. However, in this paper, we show how this principled approach to concurrent planning and execution can be made practical. We first develop the theory of dispatch-and-time-dependent value functions for CoPE, which allows us to estimate the value of all possible paths in the meta-level decision problem that end in the same world state. This allows us to search the same space as the original offline planning problem while still making decisions about which actions to dispatch or not. We then describe how to realize this idea in a planner. An empirical study shows that this new approach achieves state-of-the-art performance in concurrent planning and execution.

2 Background

We begin by reviewing the CoPE problem and previous work that takes into account the passage of time during planning.

2.1 Concurrent Planning and Execution

We adopt the definition of *concurrent planning and execution* (CoPE) as propositional temporal planning with Timed Initial

Literals (TILs), formalized in PDDL 2.2 [Cresswell and Coddington, 2003; Edelkamp and Hoffmann, 2004], as presented by Coles *et al.* [2024]. A CoPE problem Π is represented by the tuple $\Pi = \langle F, A, I, T, G \rangle$, where:

- F is a set of Boolean fluents describing the world's state.
- A is a set of durative actions. Each action $a \in A$ has a duration $dur(a) \in \mathbb{R}^{0+}$ and a start condition $cond_{\vdash}(a)$, invariant condition $cond_{\leftrightarrow}(a)$, and end condition $cond_{\dashv}(a)$ that are subsets of F. Effects are the start effect $\mathit{eff}_{\vdash}(a)$ and the end effect $\mathit{eff}_{\dashv}(a)$, each specifying which propositions in F become true (add effects) or false (delete effects).
- $I \subseteq F$ is the initial state, indicating which propositions are true at time 0.
- T is a set of timed initial literals (TILs). Each TIL $l = \langle time(l), lit(l) \rangle \in T$ consists of a time $time(l)$ and a literal $lit(l)$, defining which propositions in F are already known to become true (or false) at time $time(l)$.
- $G \subseteq F$ defines the goal, propositions that must be among those that are true at the end of plan execution.

Concurrent planning and execution allows for action dispatch before a complete plan is available. Specifically, a solution to Π is a sequence σ of pairs $\langle a, t_a \rangle$, where $a \in A$ is an action and $t_a \in \mathbb{R}^{0+}$ is its start time. This sequence must form a valid solution for Π, adhering to all conditions at respective time points and achieving the goal, as in standard temporal planning. A valid schedule is a set of instantaneous *happenings* [Fox and Long, 2003], occurring at action start, action end, and TIL triggers. For a pair $\langle a, t_a \rangle$, action a starts at t_a, requiring $cond_{\vdash}(a)$ to hold some fixed ϵ before t_a, applying $\mathit{eff}_{\vdash}(a)$ at t_a. Action a ends at $t_a + dur(a)$, requiring $cond_{\dashv}(a)$ to hold ϵ before $t_a + dur(a)$, applying $\mathit{eff}_{\dashv}(a)$ at $t_a + dur(a)$. The invariant condition $cond_{\leftrightarrow}(a)$ must hold during $(t_a, t_a + dur(a))$. Furthermore, each TIL l triggers $lit(l)$ at $time(l)$ and G must hold after all happenings.

Situated temporal planning [Cashmore *et al.*, 2018] accounts for the time that passes during planning in that, at any point during search, all actions must be scheduled after the current time t_{now}. Usually, a complete plan is emitted at once, before execution begins. In CoPE, actions are emitted incrementally but may not be scheduled in the past. Each pair in the output sequence is annotated with the time when it is emitted: $(t_1, \langle a_1, t_{a_1} \rangle), \ldots, (t_n, \langle a_n, t_{a_n} \rangle)$, where $t_1 \leq t_2 \leq \cdots \leq t_n$. Each pair $\langle a_i, t_{a_i} \rangle$ must satisfy $t_i \leq t_{a_i}$, meaning actions are dispatched at or later than when they are emitted. Typically, $t_i = t_{a_i}$, unless practical considerations dictate otherwise.

To demonstrate the importance of CoPE, consider again the example in which an automated system is tasked with navigating a train network to deliver a package by 3 PM. A TIL l ensures that the delivery deadline is met, with $time(l) = 3pm$. Without concurrent dispatching, if the current time t_n approaches $time(l)$, all actions must fit within the interval $[t_n, time(l)]$, which may be infeasible for long routes. CoPE allows early dispatching of necessary actions (e.g., starting the journey), allowing the system to meet the deadline even if the complete route cannot be fully planned in advance.

Metareasoning for Situated Temporal Planning

Rational metareasoning [Russell and Wefald, 1991] offers a framework for selecting among various computational actions. In situated temporal planning, these computational actions involve choosing which search node to expand at each iteration, with the goal of optimizing the selection of these actions to maximize the likelihood of finding a valid plan within a given time frame. Shperberg *et al.* [2019] formalized the metareasoning problem by abstracting away the complexities of plan state representation and the search process. They modeled the problem as n processes, denoted as $p_1, \ldots, p_n$, where each process represents a search for a plan (analogous to a search node on the open list). Each process is characterized by a probabilistic performance profile, represented by a random variable indicating the probability of process p_i successfully terminating after a given processing time t. The cumulative distribution function (CDF) of this random variable is denoted M_i. To succeed, a process must terminate before its deadline, which may be uncertain during planning and is therefore treated as a random variable, with its CDF represented as D_i. This deadline is in 'wall clock' time (the total time allocated across all processes so far), whereas M_i accounts only for 'CPU time' (the time allocated so far specifically to process p_i). Given the constraint that information about the actual deadline and the processing time of p_i is revealed only upon its termination, the objective is to find an optimal policy for allocating processing time (ie, search effort) across all processes, maximizing the probability that at least one process will complete and deliver a plan before its respective deadline. A simplified version of this problem, where time is discretized, is called Simplified Allocating Planning Effort when Actions Expire (S(AE)^2).

Due to the computational expense of solving even S(AE)^2, Shperberg *et al.* [2021] introduced a more practical metareasoning scheme known as Delay Damage Aware (DDA). The DDA approach is based on the log-probability of failure (LPF) when allocating t consecutive units of computation time to process i starting at time t_b, denoted $\text{LPF}_i(t, t_b)$. To compute the LPF, one first calculates the probability that process i finds a timely plan when allocated t consecutive time units beginning at time t_b:

$$s_i(t, t_b) = \sum_{t'=0}^{t} m_i(t') \cdot (1 - D_i(t' + t_b)) \tag{1}$$

where $m_i(t) = M_i(t) - M_i(t-1)$, i.e., the probability mass function of M_i. Then, $\text{LPF}_i(t, t_b) = \log(1 - s_i(t, t_b))$. By taking the log of the probability of failure, it can be treated as an additive utility function.

DDA allocates chunks of t_u computational time units, where t_u is a hyperparameter. The utility of a process i is defined by the log-probability of failure of allocating computation time to process i in the next chunk (starting at time t_u with a discount factor of γ) minus the log-probability of failure of allocating time to process i now, thus accounting for the urgency of the process. The amount of computation time used in the utility calculation is chosen by the *most effective computation time* for process i starting at time t_b, defined as $e_i(t_b) = \arg\min_t \frac{\text{LPF}_i(t, t_b)}{t}$, that is, the time allocation

is chosen by its marginal gain. Putting this all together, the DDA scheme allocates the next unit of computation time to the process i with maximal:

$$Q(i) = \frac{\gamma \cdot \text{LPF}_i(e_i(t_u), t_u)}{e_i(t_u)} - \frac{\text{LPF}_i(e_i(0), 0)}{e_i(0)} \quad (2)$$

Shperberg *et al.* [2021] implement the DDA strategy in the situated temporal planner of Cashmore *et al.* [2018]. The base planner adapts a conventional temporal plan search to handle TILs and prunes nodes that become infeasible due to time constraints. To apply DDA, the planner estimates the values of M_i and D_i that are needed to compute LPF. An admissible deadline for node i is derived by solving a Simple Temporal Problem (STP) for the action sequence H_i that leads to the search node i, yielding the latest feasible timestamp t_{max} for each step. The minimum value across all steps is the deadline for dispatching i:

$$latest\_start(H_i) = \min_{a_j \in H_i} tmax(a_j)$$

A more informative, though inadmissible, estimate is obtained using the Temporal Relaxed Planning Graph (TRPG) heuristic [Coles *et al.*, 2010] by also considering the deadlines on steps in the relaxed plan π_i from node i. The TRPG heuristic provides a relaxed plan π_i composed of snap-action–timestamp pairs $[\langle a_0, t_0\rangle..\langle a_n, t_n\rangle]$, with all timestamps $\geq t_{now}$. If a step $\langle a_j, t_j\rangle \in \pi_i$ makes use of an action a_j with known deadline $d(a_j)$ (due to the timed initial literals), then this relaxed plan step – were it to be used within a plan extension forwards from i – can be executed no later than $d(a_j)$, but also no earlier than t_j, due to the time taken to satisfy its preconditions (which are admissibly estimated by the TRPG). Hence, the maximum amount of time a_j can be delayed beyond its earliest time t_j is $d(a_j) - t_j$; which, relative to t_{now} gives an absolute time by which the execution of π_i must have started in order to reach the preconditions of a_j and begin its execution on time. The tightest such time can be taken as a deadline for the relaxed plan:

$$latest\_start(\pi_i) = \min_{\langle a_j, t_j\rangle \in \pi_i} t_{now} + (d(a_j) - t_j)$$

Thus, an (inadmissible) latest start time estimate for node i is:

$$latest\_start(H_i, \pi_i) = \min\{ \quad latest\_start(H_i), \\ latest\_start(\pi_i)\}$$

This can be used to define D_i as having probability 1 for all $t \leq latest\_start(H_i, \pi_i)$ and 0 for all later t.

To estimate M_i, the planner uses the *expansion delay* [Dionne *et al.*, 2011] — the average number of expansions between a node's generation and its expansion. A distribution around this estimate is built from search statistics and the length of the relaxed plan π_i [Shperberg *et al.*, 2021].

With estimates of M_i and D_i, the planner computes Q_i for each node and sorts them based on Q. Following the DDA scheme, the planner allocates t_u units of computation time to the chosen node, performing t_u expansions in its subtree. Afterward, any remaining frontier nodes are added back to the open list, and the next state is selected based on updated Q values. Since the M_i estimates evolve during the search, Q values are recomputed after every t_u expansions. This 'allocate-t_u' mechanism is needed to ensure the actual search behaves similarly to how the abstract metareasoning process expects. As we discuss later, this is not needed with the approach we present in this paper.

Metareasoning for CoPE

The S(AE)^2 metareasoning formalization of situated planning was extended to CoPE by Elboher *et al.* [2023]. In addition to the M_i and D_i distributions, each process p_i has an initial action sequence H_i, where every action in H_i is associated with a specific execution time window.

In addition to the meta-level actions of deciding which process to allocate computation time to, the model allows one to actually execute the next domain action in one of the sequences H_i. All domain actions are non-preemptible and mutually exclusive, except that allocating computation time to a process can be run concurrently with at most one executing action. As before, allocating computation time to process i can result in the process terminating (with a probability determined by M_i), at which point the process delivers a solution, and its true deadline is revealed.

Integrating Metareasoning into a Planner

Coles *et al.* [2024] showed how the formal model of Elboher *et al.* [2023] could be adapted for use in a planner to guide search and make dispatch decisions incrementally. In their planner, an action is dispatched at time t_{now} if, among all the actions that could be dispatched, its dispatch is estimated to maximally reduce the estimated probability of failing to reach the goals on time and this reduced probability is at least some threshold amount less than the estimated probability of failure of just carrying on searching and only considering dispatching an action at a later time than t_{now}.

To estimate the probability of failing to reach the goals after dispatching some action a, a projection of the open list is taken, keeping only frontier states in the subtree beneath a. Then, the probability of failing to reach the goals is computed by approximating what search would then do. Key to this is the successive calculation of the probability of failure to reach the goals from the frontier states in this projection: if dispatching a relieves time pressure, the aggregation of the probabilities of failure of the frontier states in the projection for a will improve on that of the full open list (that is, without the assumed dispatch of a).

This dispatch reasoning approach has two limitations, as identified by Coles *et al.* [2024]. First, it is too computationally expensive to perform at every expansion in search, so dispatch reasoning was performed only once per t_u expansions (for their experiments, $t_u = 100$). Second, as expansion decisions are driven exclusively by an aim to reach the goals on time (assuming no further dispatch), the dispatch reasoning may be misled by the estimated probability of failure for dispatch candidates whose subtrees have been scarcely explored (due to appearing to be further from the goal than other options), because the deadlines for states in the subtrees for such dispatch candidates rely heavily on the deadline of their relaxed plans, which in practice are a great deal more optimistic. This can lead to dispatch options that are 'poisoned chalices' – attractive to the dispatch reasoning due to having

the lowest estimated probability of failure, but detrimental to actually reaching the goal, with their dispatch making it difficult or impossible to then reach the goals on time.

In an ad hoc attempt to work around this limitation, Coles *et al.* [2024] only allow the dispatch of an action a if its subtree has been explored to at least some minimum extent; hence the aggregation of the probabilities of failure in the projection for a are based on nodes that are at least somewhat deep in a subtree. Then, if a is identified as an ostensibly promising dispatch candidate but does not meet this exploration requirement, a 'subtree focus' is triggered, in which search is constrained to only expand nodes beneath a until the next time dispatch is considered. In other words, even if the nodes in the subtree beneath a would not otherwise be expanded – because they are not the most promising for reaching the goals – their expansion is forced, as a way of improving the quality of the estimate of dispatching a, as going deeper in its subtree will reduce the extent to which the probability of failure for dispatching a is optimistic.

2.2 Other Related Work

Also relevant, but less directly related to our setting, are works that have investigated taking time into account in the search strategy of a planning agent. For example, Gu *et al.* [2022] reason about when to dispatch an action by modeling the tighter search focus that re-rooting the search at a deeper node would cause and estimating the resulting larger reduction of uncertainty in that single subtree. While very principled, their method does not take other temporal constraints, such as deadlines, into account.

Cserna *et al.* [2017] show how to take into account the fact that expansions will occur concurrently during the future execution of an action, allowing a planner to decide to execute long-duration actions in order to perform more search before making a crucial dispatch decision. However, their method does not handle temporal constraints, and it depends on ad hoc estimates of uncertainty.

Thomas *et al.* [2024] address real-time planning in a dynamic environment, where the agent must issue its next action within a strict time constraint. They take time into account by representing the value of a node as a so-called functional, i.e. a function of time. However, the real-time requirement means that they do not need to reason about when to dispatch actions; rather, the problem setting dictates when that must happen.

3 Dispatch-Dependent Value Functionals

To set the stage for our new approach, we first describe *dispatch-dependent value functionals*[1] for CoPE. Unlike the time-dependent value functionals of Thomas *et al.* [2024], in which the search space and the temporal constraints are fully known, here we must account for the partial knowledge our search has about the deadlines.

To define these functionals, let H_i be some action sequence reaching some node i in our search space. We can partition H_i into two consecutive segments $H_i = H_i^d \cdot H_i^c$ where:

- H_i^d is the sequence of actions that have already been dispatched. H_i^d is fixed for all nodes in the search space.
- H_i^c are the remaining actions to reach node i, that could be dispatched, but have not yet been.

Given such a partition, we define the value functional to be the probability that search will find a timely plan, assuming we start searching beneath i at time t, having previously committed to the dispatch of H_i^d:

$$V(H_i^d, H_i^c)(t) = \Pr_{m \sim M_i}(t + m \leq d(i)) \tag{3}$$

where M_i is the distribution on remaining search time from node i, π_i is a relaxed plan from node i, and $d(i) = latest\_start(H_i^c, \pi_i)$ is the deadline for node i based on H_i^c and the relaxed plan π_i.

Note that the functional for H_i yields a function that depends on t, representing how the value reachable through expanding nodes in the subtree of i will change over time. If the problem has a global deadline, V is monotonically decreasing in t: the more time passes, the more nodes in the subtree expire, until it is empty and hence flat-lines at the value 0.

Returning to the idea of dispatch, were the next action in H_i^c to be dispatched, it would be moved from the head of H_i^c to the tail of H_i^d; hence $latest\_start$ would be taken over a smaller H_i^c. More generally, if the first n actions in H_i^c were dispatched, they would be moved to the tail of H_i^d, and $latest\_start$ would be computed over an even smaller H_i^c; until, at the limit, with all steps in H_i^c dispatched, the latest start time of an empty H_i^c would be infinite, and only $latest\_start(\pi_i)$ would matter for computing the latest start time of node i.

We can now define our *dispatch-dependent value functionals*: for each prefix H_i^p of H_i^c, we can compute the functional assuming we have already committed to dispatching H_i^p. Thus, we get $|H_i^c| + 1$ different functionals – one for each possible prefix.

So far, we have considered a single node i. We now take a broader perspective, looking at the search process over the entire search space. As our decision making is embedded in a forward search, we argue that it is not necessary to consider all $|H_i^c|+1$ possible prefixes – it is sufficient to consider only the prefixes corresponding to none, or all, of H_i^c. Suppose we have a node j reached by plan H_j and with relaxed plan $\pi_j = [a_0..a_k]$. For the functional in which all $|H_j^c|$ actions are assumed dispatched ($H_j^p = H_j^c$), then any deadlines are due only to π_j. If π_j is a reasonable approximation of the plan from j to the goals, then in forward search, j will have a successor i where $H_i = (H_j \cdot a_0)$ and $\pi_i = [a_1..a_k]$, i.e. the remainder of π_j's relaxed plan after a_0. Then suppose we wanted to evaluate an assumed dispatch functional for i where only $|H_i^c| - 1$ of the actions in H_i^c are assumed to be dispatched. As the plan to i is $H_i = (H_j \cdot a_0)$, this would mean assuming H_j was dispatched but a_0 and $\pi_i = [a_1..a_k]$ were not. This is exactly the assumed dispatch considered for its parent j in the 'assume all dispatched' case: assume H_j dispatched and assume $[a_0..a_k]$ not dispatched. Hence, if relaxed plans are reasonable, it is a reasonable position when calculating dispatch-dependent value functionals to only consider dispatching none of the actions to a node, or all of them;

[1] A functional is a function which returns a function

as at least for nodes that are consistent with the relaxed plan, other intermediate dispatch prefixes have already been considered at their ancestor nodes.

4 A Simpler Approach to CoPE

When considering concurrent planning and execution, at any point during search there is a decision to be made: to choose a node for expansion or to dispatch one or more actions to be executed. Dispatch decisions are made according to what is known about the search space, taking this to be a proxy for the underlying state space. From this perspective, expansion and dispatch are fundamentally intertwined decision-making problems. Expanding a node develops the search space, which as a proxy for the state space, informs execution decisions; and vice versa, dispatching an action restricts search to then only having access to the subtree beneath what has been dispatched, affecting what can then be chosen to be expanded. Critically, as dispatching actions may be irreversible, a bad dispatch decision may then make the resulting search problem unsolvable, i.e. there is no way to reach the goals, and/or to do so in a timely manner.

Hence, for concurrent planning and execution to work well, node expansion decisions are motivated by two concerns:

1. As in conventional planning, nodes should be chosen for expansion in order to make progress towards finding a solution plan.
2. Additionally, nodes should be chosen for expansion in order to reveal the information needed to make good dispatch decisions.

For the former, we can rely on heuristic search that evaluates nodes with a value function that rewards progress towards finding a solution plan. For the latter, some sort of search modification is needed to ensure the necessary information is gathered. The subtree focus approach of Coles *et al.* [2024] (described in Background) can be seen as a means of enforcing expansions that will reveal the information needed to make good dispatch decisions. The main limitations of that approach are, first, that a restriction to a subtree will only be considered periodically, once every t_u expansions, when performing dispatch reasoning, so if there are n poisoned chalices that will be recognized as in fact being poor candidates for dispatch once their subtrees have been explored, it will take $t_u \times n$ expansions to eliminate them all. Second, whenever a subtree is chosen for expansion, the expansions are still driven by the value function of search, which primarily chooses for expansion nodes that are closer to the goal – which are not necessarily the expansions that will, within that subtree, most efficiently reveal the information needed to assess its suitability for dispatch.

4.1 Dispatch-Dependent Value Functions

In contrast with the subtree focus approach, we propose to determine which nodes to expand by using two value functions: one that aims to quickly guide search towards the goals and another that gathers information needed to inform dispatch decisions, i.e. that lead to the expansion of nodes whose plan prefixes appear to be good candidates for dispatch, considering the time pressure that is apparent at time t_{now}. In fact, these functions can be viewed as values of the functional defined in Section 3, at different points.

First, as a value function to promote the expansion of states progressing towards the goals, the obvious candidate is the classic weighted f value used by weighted A*: $f(i) = g(i)+w\times h(i)$, where $h(i) = |\pi_i|$ is the relaxed plan distance-to-go (relaxed plan length) from node i to a goal state. $f(i)$ pays no regard to any time pressure – it does not depend at all on the (assumed) latest start time of i. Instead, it is widely used to reduce the number of expansions required to reach a goal node when planning under time pressure [Ebendt and Drechsler, 2009], and is used by many temporal planners, including OPTIC [Benton *et al.*, 2012] upon which we build.

Second, we note that the dispatch decisions of Coles *et al.* [2024] are based on an aggregation of the probability of failure to reach the goals on time from the nodes in a subtree beneath a candidate action to dispatch. This motivates using a value function that expands nodes that one can expect to be part of a subtree for a favorable dispatch option. We cannot use this dispatch reasoning itself as a value function, in part because it is answering the wrong question (candidate actions to dispatch are at the root of the tree; nodes to expand are at the frontier), but also because it is too computationally expensive to do for every state. Instead, we turn to our dispatch-dependent value functionals for CoPE (equation 3), which give the probability of failure to reach the goal from a node as a candidate value function; i.e. for node i, what proportion of the distribution M_i falls after the node's deadline. We compute the value functional assuming all remaining non-dispatched actions along the path to i (H_i^c) have been dispatched, following our earlier argument in support of this. For brevity, we denote this assumed-dispatch value function that gives the probability of failure as $pf(i)$. Between two nodes with the same $pf(i)$ value, the one that appears to be closer to the goal is better, thus we break ties between equal-$pf(i)$ value states according to $f(i)$.

Both of these two value functions are useful, hence we propose to use them with dual open-list search [Helmert, 2006], where the choice of which state to expand alternates between $f(i)$ and $pf(i)$, with removal of a state from one open list removing it from both and the successor states resulting from an expansion being added to both. As with the $Q(i)$ values of Shperberg *et al.* [2019], given that time is passing and the distribution M_i is continually updated during search, it is necessary to periodically recompute the values of $pf(i)$. So $pf(i)$ is first computed when i is inserted into the open lists, with $t = t_{now}$ at that time using the then-current M_i, and then every $t_u = 100$ expansions, the $pf(i)$ values are re-computed for all nodes on the open lists with updated $t = t_{now}$ and M_i.

Using a dual open list with these value functions will result in search that is somewhat self-balancing with respect to time pressure. If all deadlines are far in the future, the $pf(i)$ value for nodes with good $f(i)$ values will be 0, as there is estimated to be sufficient time to complete the search from i to the goals. Hence, as $pf(i)$ is tie-broken by $f(i)$, the best state on the $f(i)$ and $pf(i)$ open lists is the same, so search will attempt to reach the goals quickly, which is sensible in the absence of apparent time pressure. Then, as time passes, the $pf(i)$ values for nodes with more time pressure will increase

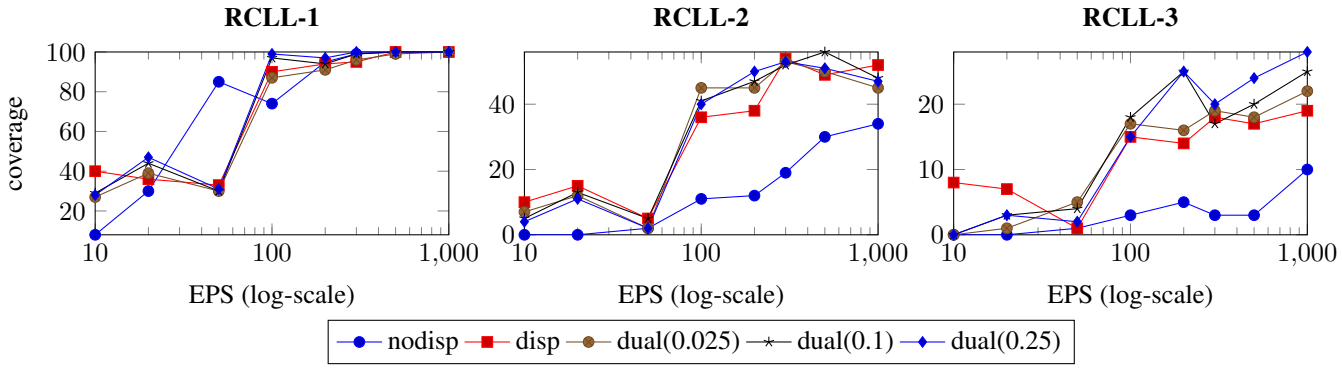

Figure 1: Number of Solved Instances (Coverage, y-axis) at Different Expansion Rates (EPS, x-axis)

(their deadlines are closer, so the proportion of M_i after the deadline is greater), so the best state on each open list can differ, leading to a balance of expansions according to $f(i)$ that aim to reach the goals quickly and those according to $pf(i)$, which gather the information needed to make sensible dispatch decisions.

Finally, while we propose alternating between expansions according to $f(i)$ and $pf(i)$ as being a principled basis for search, there are additional possibilities for either or both of these value functions. For the first open list, which nominally assumes no dispatch and uses $f(i)$, one could instead use $Q(i)$ [Shperberg *et al.*, 2019] (with no assumed dispatch), or the probability of failure to reach the goals from i (with no assumed dispatch). For the second open list, which assumes dispatch and nominally computes $pf(i)$, one could instead compute $f(i)$, or $Q(i)$ (with assumed dispatch). We evaluate these possibilities below.

5 Empirical Evaluation

We have presented a technique for concurrent planning and execution that relies on a simple dual-open-list mechanism. To evaluate this, we have implemented it within the planner OPTIC [Coles *et al.*, 2012].[2] For our evaluation, for the $f(i)$ open list (WA*), we set the 'W' value to 5 – the default used in OPTIC. Action dispatch reasoning is performed according to Coles *et al.* [2024], but without the dispatch reasoning constraining search to expanding a promising subtree, as the divergence between the best states according to $f(i)$ and $pf(i)$ is an alternative to this. The only remaining parameter for the dispatch reasoning is the dispatch threshold, which we will explore within this evaluation. For brevity, we refer to our new dual-open list search planner as **dual**.

Similarly to previous work on concurrent planning and execution [Coles *et al.*, 2024], we evaluate our techniques on the PDDL encoding of the Robocup Logistics League (RCLL) [Niemueller *et al.*, 2015]. RCLL simulates robots moving between different production machines in a factory, using the machines to process raw materials into a final product. Each order has a deadline, making this domain challenging for concurrent planning and execution. We use the same 100 RCLL scenarios as the previous work and create 3 versions for each one, with 1, 2, and 3 robots. Increasing the number of robots increases the branching factor, as the robots can act concurrently, making the search problem harder. On the other hand, increasing the number of robots also decreases the time pressure, as more actions can be executed in parallel.

Also following previous work on concurrent planning and execution, we evaluate each problem instance using different simulated CPU speeds, implemented here as a fixed simulated expansion rate (measured in expansions per second – EPS). We used values for EPS ranging from 10 expansions per second (a very slow CPU) to 1000 expansions per second (a very fast CPU). Increasing the expansion rate allows concurrent planning and execution to perform more search before an action must be dispatched, thus allowing us to control the time pressure.

We compare the performance of **dual** to two baselines:

nodisp a situated temporal planner [Shperberg *et al.*, 2021] that does not dispatch actions until a complete plan is found.

disp the previous CoPE planner [Coles *et al.*, 2024]. This planner needs a dispatch threshold. Additionally, it relies on the subtree focus mechanism, with a subtree focus threshold set to half the dispatch threshold.

For dispatch threshold, we considered three possible values (0.025, 0.1, 0.25) for both **disp** and **dual**. Additionally, for both baselines we considered whether or not to enable the allocate-t_u mechanism (from [Shperberg *et al.*, 2019] – when node i is expanded, $t_u = 100$ expansions are allocated to its subtree). Thus, for **nodisp** we have two versions, and for **disp** we have 6 versions. For each of the baselines we also compare to the Virtual Best Solver (VBS) – a clairvoyant planner that chooses the best configuration for the specific domain (RCLL-1, RCLL-2, or RCLL-3) and for the specific EPS. Thus, the VBS is a hard baseline to beat.

In addition to these baselines, we also compare to a modification of **dual** that uses different priority functions to sort the two open lists. As discussed earlier, the possible value functions are f, probability of failure (pf), and Q. We use the notation p_1/p_2 to indicate that the first open list is sorted by p_1 and the second by p_2. As in **dual**, p_1 is computed assuming no dispatch of any of the steps along the path to a node; whereas p_2 is computed assuming dispatch of all the

[2] Source code freely available from the authors upon request.

Method alg	a	d	EPS 10	20	50	100	200	300	500	1000	SUM
RCLL-1											
nodisp			6	14	43	57	80	90	99	99	488
nodisp	✓		8	30	**85**	74	95	99	**100**	**100**	591
nodisp		VBS	8	30	85	74	95	99	100	100	591
disp		0.025	26	27	27	83	85	91	99	**100**	538
disp	✓	0.025	39	36	33	90	90	94	99	**100**	581
disp		0.1	26	30	27	88	89	92	99	**100**	551
disp	✓	0.1	**40**	36	33	86	93	95	**100**	**100**	583
disp		0.25	27	33	27	88	89	93	99	**100**	556
disp	✓	0.25	**40**	36	33	88	94	95	**100**	**100**	586
disp		VBS	40	36	33	90	94	95	100	100	588
pf/f		VBS	25	35	30	92	91	97	100	100	570
pf/pf		VBS	28	37	26	77	86	80	95	95	524
Q/pf		VBS	32	41	32	89	87	93	100	99	573
Q/Q		VBS	28	30	30	79	88	95	99	100	549
dual		0.025	27	39	30	87	91	96	99	**100**	569
dual		0.1	29	44	30	97	94	99	**100**	**100**	593
dual		0.25	28	**47**	31	**99**	**97**	**100**	**100**	**100**	**602**
RCLL-2											
nodisp			0	0	0	0	5	13	23	34	75
nodisp	✓		0	0	2	11	12	19	30	34	108
nodisp		VBS	0	0	2	11	12	19	30	34	108
disp		0.025	9	14	0	26	30	34	34	33	180
disp	✓	0.025	6	13	4	34	36	53	46	51	243
disp		0.1	**10**	**15**	**5**	27	27	31	34	36	185
disp	✓	0.1	7	13	4	36	38	53	48	**52**	251
disp		0.25	7	11	2	30	28	30	34	34	176
disp	✓	0.25	7	13	5	35	34	**54**	49	52	249
disp		VBS	10	15	5	36	38	54	49	52	259
pf/f		VBS	10	16	4	46	48	48	44	41	257
pf/pf		VBS	9	12	1	10	12	16	12	18	90
Q/pf		VBS	5	17	3	19	18	25	28	35	150
Q/Q		VBS	5	11	2	31	27	29	32	30	167
dual		0.025	7	12	2	**45**	45	53	50	45	259
dual		0.1	5	13	**5**	41	47	52	**56**	48	**267**
dual		0.25	4	11	2	40	**50**	53	51	47	258
RCLL-3											
nodisp			0	0	0	0	2	1	3	10	16
nodisp	✓		0	0	1	3	5	3	3	9	24
nodisp		VBS	0	0	1	3	5	3	3	10	25
disp		0.025	**8**	6	0	15	12	13	15	17	86
disp	✓	0.025	2	2	0	14	10	10	4	14	56
disp		0.1	**8**	5	0	11	11	16	17	18	86
disp	✓	0.1	2	3	0	14	10	8	7	15	59
disp		0.25	5	**7**	1	13	14	18	17	19	94
disp	✓	0.25	2	2	1	12	12	11	6	14	60
disp		VBS	8	7	1	15	14	18	17	19	99
pf/f		VBS	1	2	1	16	21	21	26	21	109
pf/pf		VBS	3	3	3	7	12	4	4	15	51
Q/pf		VBS	1	6	2	13	15	19	15	19	90
Q/Q		VBS	7	6	2	9	13	11	12	14	74
dual		0.025	0	1	**5**	17	16	19	18	22	98
dual		0.1	0	3	4	**18**	**25**	17	20	25	112
dual		0.25	0	3	2	15	**25**	**20**	**24**	**28**	**117**

Table 1: Coverage — the number of solved problems by each technique for different expansion rates (EPS). a indicates whether allocate-t_u is used, d is the dispatch threshold.

steps along the path to a node. We compare to pf/f, pf/pf, Q/pf, and Q/Q. For each of these, we try the three possible dispatch thresholds and report the VBS.

Table 1 shows the coverage (number of problems solved) for each technique (including the VBS) for each domain and expansion rate. For each technique we also summarize the number of problems solved across all expansion rates, to get a single number to compare. The best non-VBS methods for each domain/EPS combination are in bold. The results show that the dual technique beats the VBS of each baseline and of the other dual open list configurations overall, as well as in most settings. Specifically, **dual** with a dispatch threshold of 0.1 always beats the VBS of both baselines and other dual configurations on the total number of problems solved in all 3 domains. Furthermore, **dual** with a dispatch threshold of 0.25 does even better for both RCLL-1 and RCLL-3. Looking more closely at the best method for each domain/EPS combination, **dual** is the best in 16 cases, while one of the baseline methods is the best in 9 cases (ties are counted twice).

EPS	10	20	50	100	200	300	500	1000
RCLL-1								
dual(0.025)	77.8	79.6	72.6	39.8	30.8	32.4	22.6	16.7
dual(0.1)	77.8	79.3	72.1	37.5	30.1	31.7	22.8	17.9
dual(0.25)	77.2	78.9	72.4	34.7	29.5	34.5	22.7	20.0
RCLL-2								
dual(0.025)	68.2	61.6	54.3	38.0	32.9	30.7	32.0	30.6
dual(0.1)	65.7	61.8	52.3	39.2	34.8	30.7	31.7	28.8
dual(0.25)	64.6	61.7	53.3	35.6	35.2	30.2	31.9	28.9
RCLL-3								
dual(0.025)	62.9	59.6	51.6	38.5	33.1	28.2	27.2	22.9
dual(0.1)	61.0	57.8	50.8	35.0	31.7	28.9	25.4	21.7
dual(0.25)	61.3	56.1	48.8	37.8	27.6	32.1	22.6	22.9

Table 2: Average Agreement Rate Between Open Lists (Percent)

To illustrate this graphically, Figure 1 plots the coverage for each EPS of the two baseline VBS and for **dual** with the different dispatch thresholds. In these plots, it is easy to see that the lines for **dual** are typically above that of **disp**, which is typically above **nodisp** – especially when the expansion rate is 100 EPS or more.

Looking more closely into the results in different domains, in RCLL-1 the search space is small enough that even **nodisp** is competitive with concurrent planning and execution. However, for RCLL-2 and RCLL-3, where the search space is larger, using concurrent planning and execution pays off significantly – solving at least twice as many problems as **nodisp**. The difference between **dual** and **disp** is larger in RCLL-3 than in RCLL-2, indicating that **disp** gets lost in the large search space of RCLL-3 more than **dual**.

To better understand **dual**, Table 2 shows the fraction of times both open lists agreed on the next node to expand (in percent). This ratio is averaged across all instances for the given domain/EPS combination. As these results show, as the expansion rate increases, the agreement rate decreases, with a sharp transition of around 100 expansions per second. This is likely because, for a slow expansion rate, the second open list detects the time pressure and prefers actions with an urgent deadline. As the expansion rate increases, time pressure decreases. Furthermore, in this slow CPU regime, we can also see that, as the number of robots increases, the agreement rate decreases. Again, this is because increasing the number of robots increases the size of the search space and thus increases time pressure. However, when the expansion rate passes the critical region of 100 expansions per second, this is less pronounced.

6 Conclusion

We have presented a new approach for CoPE. This approach relies on building blocks developed in previous work on situated temporal planning and CoPE, such as methods for estimating the distributions of remaining search time and deadlines. However, the new notions of time-and-dispatch-dependent value functions for concurrent planning and execution enable a dual-queue approach that is both simpler than the previous state-of-the-art and outperforms it. This work enables agents to gracefully manage planning and acting in challenging problems involving time pressure.

Acknowledgments

This research was supported by Grant No. 2019730 from the United States-Israel Binational Science Foundation (BSF) and by Grant No. 2008594 from the United States National Science Foundation (NSF). The project was also funded by the EPSRC-funded project COHERENT (EP/V062506/1), the Israeli CHE Data Science Grant, by the Israel Science Foundation (ISF) grant #909/23, and by Israel's Ministry of Innovation, Science and Technology (MOST) grant #1001706842, in collaboration with Israel National Road Safety Authority and Netivei Israel.

References

[Benton *et al.*, 2012] J. Benton, Amanda Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of ICAPS*, 2012.

[Cashmore *et al.*, 2018] Michael Cashmore, Andrew Coles, Bence Cserna, Erez Karpas, Daniele Magazzeni, and Wheeler Ruml. Temporal planning while the clock ticks. In *Proceedings of ICAPS*, pages 39–46, 2018.

[Coles *et al.*, 2010] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *Proceedings of ICAPS*, pages 42–49, 2010.

[Coles *et al.*, 2012] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)*, 44:1–96, 2012.

[Coles *et al.*, 2024] Andrew Coles, Erez Karpas, Andrey Lavrinenko, Wheeler Ruml, Solomon Eyal Shimony, and Shahaf S. Shperberg. Planning and acting while the clock ticks. In *Proceedings of ICAPS*, pages 95–103, 2024.

[Cresswell and Coddington, 2003] Stephen Cresswell and Alexandra Coddington. Planning with timed literals and deadlines. In *Proceedings of 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, pages 23–35, 2003.

[Cserna *et al.*, 2017] Bence Cserna, Wheeler Ruml, and Jeremy Frank. Planning time to think: Metareasoning for on-line planning with durative actions. In *Proceedings of ICAPS*, 2017.

[Dionne *et al.*, 2011] Austin J. Dionne, Jordan Tyler Thayer, and Wheeler Ruml. Deadline-aware search using on-line measures of behavior. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2011.

[Ebendt and Drechsler, 2009] Rüdiger Ebendt and Rolf Drechsler. Weighted A* search – unifying view and application. *Artificial Intelligence*, 173(14):1310–1342, 2009.

[Edelkamp and Hoffmann, 2004] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg, 2004.

[Elboher *et al.*, 2023] Amihay Elboher, Ava Bensoussan, Erez Karpas, Wheeler Ruml, Shahaf S. Shperberg, and Solomon Eyal Shimony. A formal metareasoning model of concurrent planning and execution. In *Proceedings of AAAI*, 2023.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124, 2003.

[Gu *et al.*, 2022] Tianyi Gu, Wheeler Ruml, Shahaf S. Shperberg, Solomon Eyal Shimony, and Erez Karpas. When to commit to an action in online planning and search. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, pages 83–90, 2022.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)*, 26:191–246, 2006.

[Niemueller *et al.*, 2015] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *ICAPS Workshop on Planning and Robotics (PlanRob)*, 2015.

[Russell and Wefald, 1991] Stuart J. Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1-3):361–395, 1991.

[Shperberg *et al.*, 2019] Shahaf S. Shperberg, Andrew Coles, Bence Cserna, Erez Karpas, Wheeler Ruml, and Solomon Eyal Shimony. Allocating planning effort when actions expire. In *Proceedings of AAAI*, pages 2371–2378, 2019.

[Shperberg *et al.*, 2021] Shahaf S. Shperberg, Andrew Coles, Erez Karpas, Wheeler Ruml, and Solomon Eyal Shimony. Situated temporal planning using deadline-aware metareasoning. In *Proceedings of ICAPS*, pages 340–348, 2021.

[Thomas *et al.*, 2024] Devin Wild Thomas, Wheeler Ruml, and Solomon Eyal Shimony. Real-time safe interval path planning. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pages 161–169, 2024.