Combining Domain-Independent Planning and HTN Planning: The Duet Planner

Alfonso Gerevini[†] and **Ugur Kuter**[‡] and **Dana Nau**[‡] and **Alessandro Saetti**[†] and **Nathaniel Waisbrot**[‡]

Abstract. Despite the recent advances in planning for classical domains, the question of how to use domain knowledge in planning is yet to be completely and clearly answered. Some of the existing planners use domain-independent search heuristics, and some others depend on intensively-engineered domain-specific knowledge to guide the planning process. In this paper, we describe an approach to combine ideas from both of the above schools of thought. We present Duet, our planning system that incorporates the ability of using hierarchical domain knowledge in the form of *Hierarchical Task Networks (HTNs)* as in SHOP2 [14] and using domain-independent local search techniques as in LPG [8]. In our experiments, Duet was able to solve much larger problems than LPG could solve, with only minimal domain knowledge encoded in HTNs (much less domain knowledge than SHOP2 needed to solve those problems by itself).

1 Introduction

Most classical planners fall into one of two categories: planners that use domain-independent knowledge, i.e., that work in any classical planning domain, and planners that can exploit domain-specific knowledge. It has been shown, both theoretically and experimentally, that each approach has its own advantages and disadvantages:

- A planner that can exploit domain-specific knowledge in order to guide its planning can solve much larger planning problems and can generally solve them much faster than the planners that don't use such knowledge. The biggest downside of such planning systems, however, is that they require an expert human to give them extensive knowledge about how to solve planning problems in the planning domain at hand. Usually this knowledge is expressed using either temporal logic (e.g., TLPlan [1] and TALplanner [13]) or task decomposition (e.g., SHOP2 [14], SIPE-2 [17], and O-PLAN [6]), and might not be easy for the general user to specify.
- A planner that uses domain-independent heuristic information (e.g., FF [11], AltAlt [15], SGPlan [5], HSP [3], FastDownward [10], and LPG [8]) usually does not need expert-provided domain knowledge, since the planner itself computes a heuristic for each domain. This makes the domain formalization simpler and the planner easier to use; but the planner may often perform much worse than a planner that exploits specific domain knowledge.

In this paper, we describe Duet, a new planning system that combines the advantages of using domain-independent heuristics and domain-specific knowledge, while avoiding their drawbacks.[1] To accomplish this, Duet incorporates adaptations of two well-known planners: LPG, which uses domain-independent heuristics in a stochastic local search engine [8], and SHOP2, which uses domain-specific Hierarchical Task Networks (HTNs) to organize its search space [14]. We extended the SHOP2 and LPG formalisms to allow the planners to communicate in Duet by generating subgoals of a planning problem for each other. Duet organizes the planning process by passing these subgoals to the individual planners until no subgoals are left to achieve.

We present our experiments with Duet on a new planning domain, called Museums. The Museums domain was inspired by the real-world operations of acquiring and relocating art objects among a set of museums around the world. The domain combines aspects of the well-known Logistic and Tower of Hanoi (ToH) problems. The objective is to use trucks to move various art objects from museums to other museums. When a truck comes to a museum to load or unload objects, there are three places to put the objects: the truck, and two pallets at the museum's loading dock. An object's placement depends on its fragility: fragile art objects must be placed on less-fragile ones. Thus, loading and unloading correspond to solving ToH problems.

The rationale for using the Museums domain in the evaluation of Duet was that we observed that it is challenging for state-of-the-art planners and it includes two kinds of subproblems: domain-specific knowledge isn't needed to plan the truck movements, but is needed to plan the loading and unloading operations, since the ToH problem is hard for many domain-independent planners including LPG.

In our experiments, we varied the amount of HTN-based domain-specific knowledge available to Duet and compared its performance with LPG's and SHOP2's performance as stand-alone planners. Even with just a small amount of domain-specific knowledge (e.g., "choose the least-fragile object and move it to the target museum"), Duet usually generated solutions faster than LPG. With more domain-specific problem-solving knowledge (e.g., how to properly stack art objects on top of each other), Duet ran faster and solved more problems than both LPG and SHOP2. Although SHOP2's performance could have been improved, this would have required much more time for hand-crafting its knowledge base.

2 Preliminaries

Our definitions of classical states, planning operators, planning domains and problems are based on those in [9]. Below we'll summarize the definitions at the semantic level; for syntactic details see [9].

[†]Dipartimento di Elettronica per l'Automazione, Universitá degli Studi di Brescia, Via Branze 38, I-25123 Brescia, Italy.

[‡]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742, USA.

[1] In that sense, it is closely related to the recently-proposed "Model-Lite Planning" approach [12, 16], which aims to develop techniques that do not require intensive domain knowledge but still are practical.

In addition to classical planning operators and actions (i.e., ground instances of planning operators), we define an *abstract planning operator* as a triple $(t, \mathsf{Pre}, \mathsf{Eff})$, where Pre and Eff are the preconditions and the effects of the abstract operator (described as logical formulas over literals), and t is an expression $(name, arg_1, \ldots, arg_n)$, where $name$ is the abstract operators' name and $arg_1, \ldots, arg_n$ are the arguments (variables and/or constant symbols). An *abstract action* is a ground instance of an abstract planning operator. A *plan* is a sequence of actions that are either classical or abstract.

A planning domain is a triple $\Sigma = (S, A, \gamma)$ where S and A are the sets of states and actions (classical and abstract), and $\gamma : S \times A \rightarrow S$ is the state-transition function, with $\gamma(s, a)$ defined iff a is applicable to s. $\Gamma(s, \pi) = \gamma(\gamma(...\gamma(s, a_1), a_2), \ldots, a_n)$ is the state generated by applying the plan $\pi = \langle a_1, \ldots, a_n \rangle$ in the state s. If some action a_i is inapplicable in $\Gamma(s, \langle a_1, \ldots, a_{i-1} \rangle)$ then π is inapplicable in s and $\Gamma(s, \pi)$ is not defined.

A *planning problem* is a pair $P = (s_0, g)$ in the planning domain $\Sigma = (S, A, \gamma)$, where $s_0 \in S$ is the initial state and g is the *goals* represented as a conjunction of logical atoms (i.e., g represents a set of goal states $G \subseteq S$). A *solution* for a classical planning problem P is a plan $\pi = \langle a_1, \ldots, a_k \rangle$ such that each a_i in π is a classical action and the state $s' = \Gamma(s_0, \pi)$ satisfies the goals g.

LPG's plan representation is based on linear action graphs [8], which are variants of the well-known planning graphs [2]. A *linear action graph* [8] is a directed acyclic leveled graph alternating between a proposition level, i.e., a set of domain propositions, and an action level, i.e., one ground domain action and a set of special dummy actions, called "no-ops", each of which propagates a proposition of the previous level to the next one. If an action is in the graph, then its preconditions and positive effects appear in the corresponding proposition levels of the graph. Moreover, a pair of propositions or actions can be marked as mutually exclusive at every graph level where the pair appears (for a detailed description, see [8]).

While in the original definition, action levels contain only classical actions [8], here we use an extended representation where an action level contains either a classical action or an abstract action. An (extended) action graph can have two types of flaws: unsatisfied action preconditions and abstract actions. LPG uses a stochastic local search process that iteratively modifies the current graph until there is no flaw or a certain search limit is exceeded [8]. LPG deals with an unsatisfied precondition by inserting into or removing from the graph a new or existing action, respectively. We modified LPG in order to recognize abstract actions as flaws resolvable by running an HTN planner, as described below. An action graph with no flaws represents a solution for the input planning problem.

An HTN planner formulates a plan by decomposing tasks (i.e., symbolic representations of problem-solving activities to be performed) into smaller and smaller subtasks until tasks are reached that can be performed directly. An *HTN* is a pair (T, C), where T is a set of tasks and C is a set of partial ordering constraints on the tasks. The *empty* HTN is the pair (T, C) such that $T = \emptyset$ and $C = \emptyset$.

An HTN planner uses an *HTN domain description* that contains three kinds of knowledge artifacts: *axioms*, *operators*, and *methods.* The axioms are similar to logical Horn-clause statements; the planner uses them to infer conditions about the current state. The operators are like the planning operators used in any classical planner. The names of these operators are designated as *primitive tasks* .

Each method in an HTN domain description is a prescription for how to accomplish a nonprimitive task by decomposing it into subtasks (which may be either primitive or nonprimitive tasks). A method consists of (1) the task that the method can be used to accomplish, (2) the set of preconditions which must be satisfied for the method to be applicable, and (3) the subtasks to accomplish, along with some constraints over those tasks that must be satisfied.

For example, consider the task of moving a collection of items from one location to another. One method might be to move them by truck. For such a method, the preconditions might be that the truck is in working order and is present at the first location. The subtasks might be to open the door, put the items onto the truck, drive the truck to the other location, and unload the items.

We assume that each abstract action in a planning domain corresponds to a nonprimitive task, which must be decomposed into smaller tasks using HTN methods (if available).[2] In addition to primitive and nonprimitive tasks, we also define a class of special-purpose tasks called *achieve-goals tasks*. An *achieve-goals task* specifies a set of goals, as in a classical planning problem, that need to be achieved in the world before the task decomposition-process can progress during HTN planning. An HTN planner would not have any methods to decompose an achieve-goals task t Instead, an achieve-goals task triggers the invocation of a classical planner to generate a plan π such that the state $\Gamma(s, \pi)$ satisfies the specified goals of t, which we denote as $\mathsf{GoalsOf}(s, t)$, given the input set of actions. The achieve-goals task is an important component of our planning system Duet that incorporates LPG and SHOP2 in a unified planning process, as described in the next section.

3 Duet = LPG + SHOP2

This section describes our planning procedure, called Duet, that incorporates local-search planning as in LPG [8] and HTN planning as in SHOP2 [14]. The LPG and SHOP2 planning procedures that we use in Duet are slightly modified versions of the originals reported in [8] and [14], respectively. Below, we first describe the Duet planning procedure, and subsequently we briefly describe our modifications to LPG and SHOP2 to adapt them to work within Duet.

Figure 1 shows a high-level description of the Duet planning procedure. Duet's input includes the initial state s_0 and the goal condition g of a classical planning problem, as well as a possibly empty initial task network specified for achieving the goals g and a possibly empty set M of HTN methods. Duet first initializes the current state s to s_0 and the current partial plan to the empty plan. At Line 1, n is a counter for the number of search steps performed by the planner; that is, n is the total number of graph modifications performed by LPG to fix the flaws plus the number of task decompositions done by SHOP2. Duet also uses a *tabu list*, τ, that keeps the abstract actions that cannot be decomposed into smaller tasks given the HTN methods in M, and therefore, must be avoided during local search in LPG. The tabu list τ is initialized to the empty list at Line 1.

Duet successively generates and resolves subgoals for the input planning problem until it generates a solution plan. A *subgoal* of the planning problem is either a goal to achieve using domain-independent search heuristics via LPG, or an abstract action (i.e., a task) that needs to be decomposed into smaller tasks via SHOP2. Duet performs this iterative procedure for a maximum predefined number of search steps. If a solution cannot be found during these iterations, the procedure returns failure.

[2] Note that a *macro-action* [4, 7] is a special case of an abstract action: a macro-action decomposes directly into a sequence of primitive actions, whereas an abstract action may be decomposed into a combination of both primitive actions and other nonprimitive tasks that need to be decomposed further. This allows us, for example, to write HTNs that perform the standard recursive decomposition of a Towers of Hanoi task in the Museum domain.

Procedure Duet(s_0, g, w_0, M)
Input: The problem initial state s_0, the set of problem goals g, the initial task network w_0 and a set of HTN-methods,
Output: A solution plan or failure.

1. $n \leftarrow 0$; $s \leftarrow s_0$; $w \leftarrow w_0$; $\pi \leftarrow \tau \leftarrow g_{shop2} \leftarrow g_{lpg} \leftarrow \emptyset$;
2. while n does not exceed a predefined number of steps
3. if π is a solution (all subgoals satisfied) then **return** π;
4. else if there exists an abstract action g_{SHOP2} then
5. $\langle \pi', s', g_{\text{LPG}}, w', n \rangle \leftarrow \text{SHOP2}(s, g_{\text{SHOP2}}, \pi_{nil}, n, M)$;
6. if $\pi' =$ *failure*
7. then $\tau \leftarrow \tau \cup \langle g_{\text{SHOP2}}, n \rangle$; $w \leftarrow (w - g_{\text{SHOP2}})$;
8. else $\pi \leftarrow \pi + \pi'$; $w \leftarrow w' + (w - g_{\text{SHOP2}})$; $s \leftarrow s'$;
9. $g_{\text{SHOP2}} \leftarrow \emptyset$;
10. else if there exists an achieve-goals task g_{LPG} then
11. $\langle \pi', g_{\text{SHOP2}}, n \rangle \leftarrow \text{LPG}(s, g_{\text{LPG}}, \pi_{nil}, n, \tau)$;
12. $\pi \leftarrow \pi +$ prefix of π' up to the first abstract action;
13. $w \leftarrow$ the rest of $\pi' + (w - g_{\text{LPG}})$;
14. $s \leftarrow \Gamma(s_0, \pi)$;
15. $g_{\text{LPG}} \leftarrow \emptyset$;
16. else if $w \neq \emptyset$ then
17. $\langle \pi, s, g_{\text{LPG}}, w, nil \rangle \leftarrow \text{SHOP2}(s, w, \pi, n, M)$;
18. if $\pi =$ *failure* then **return** *failure*;
19. else
20. $\langle \pi', g_{\text{SHOP2}}, n \rangle \leftarrow \text{LPG}(s_0, g, \pi, n, \tau)$;
21. $\pi \leftarrow$ prefix of π' up to the first abstract action;
22. $w \leftarrow$ the rest of π';
23. $s \leftarrow \Gamma(s_0, \pi)$;
24. **return** *failure*.

Figure 1. Pseudocode of the Duet planning algorithm. "+" is the operator concatenating two plans, π_{nil} is the empty plan, s is the world state, w is the task network, τ is the tabu-list, g_{LPG} represents the goals specified in an achieve-goals task, and g_{SHOP22} is an abstract action.

If Duet returns failure, we re-start it from the beginning with the same input for a predefined number of times, in order to search for possible solutions again. The rationale behind these restarts is that since LPG, and therefore Duet, is a randomized search algorithm, there is a possibility that different restarts of the planner will produce different search paths in the search space and the planner will generate a solution plan.

At each iteration of the while loop (Lines 2–23), Duet first checks whether the current partial plan π is a solution for the input planning problem. If so, Duet returns this plan and terminates successfully. Otherwise, if there is an abstract action (or an HTN of abstract actions) to be accomplished, Duet invokes SHOP2 on this HTN, which is called g_{SHOP2} in Line 5. Using the input HTN methods M, SHOP2 attempts to generate a solution plan for the HTN g_{SHOP2}.

Figure 2 shows the modified version of SHOP2 [14] that Duet uses. The planning procedure is the same as in [14], except for Lines 10–12. In Line 10, if the current task to be decomposed is an achieve-goals task, then our adaptation of SHOP2 returns the GoalsOf(s, t) in the current state s. As described above, the Duet then invokes LPG on these goals to achieve them and updates the current partial plan.

When SHOP2 returns, there are three cases:

- SHOP2 generates a plan π' for g_{SHOP2} successfully using the methods in M. In this case, the returned successor HTN w' is the empty HTN and there are no successor goals for LPG (i.e., g_{LPG} is the empty set in Line 5).
- SHOP2 generates an achieve-goals task t_{LPG} for Duet to invoke LPG in the next iteration. In this case, π' is the partial plan that SHOP2 generated until the task t_{LPG} in the decomposition process, s' is the state in which LPG must be called, g_{LPG} is the goals for LPG specified by t_{LPG}, w' is the HTN that still needs to be accomplished once Duet generates a plan that achieves the goals g_{LPG}, and n is the updated number of search steps.
- SHOP2 returns failure. SHOP2's failure on g_{SHOP2} means that there are no possible ways to decompose g_{SHOP2} given the current domain knowledge and the input initial state, and therefore, LPG should not consider the particular abstract action g_{SHOP2} in its later planning invocations. In this case, Duet inserts g_{SHOP2}, along with the number of search steps generated so far, in the tabu list, and removes g_{SHOP2} from the current task network (Line 7).

Procedure SHOP2(s, w, π, n, M)
Input: a world state s, a task network w, a (partial) plan π, a number of search steps n and a set of HTN-methods,
Output: a plan, its final state, a task that has no method, a task network and a number of search steps.

1. while w is not empty do
2. nondeterministically choose a task t from w that has no predecessors and remove it;
3. $n \leftarrow n + 1$;
4. if t is primitive then
5. $\pi \leftarrow \pi + t$; $s \leftarrow \gamma(s, t)$;
6. else if t is nonprimitive then
7. choose an applicable method m for t (or if there's
8. no such method then **return** failure)
9. add decomposition to the front of tasks;
10. else if t is an achieve-goals task then
11. **return** $\langle \pi, s,$ GoalsOf$(s, t), w, n \rangle$;
12. **return** $\langle \pi, s, nil, nil, n \rangle$;

Procedure LPG($s, g, \pi, n_{init}, \tau$)
Input: an initial world state s, a set of goals g, a (partial) plan π, a number of search steps n_{init} and a tabu-list τ,
Output: a plan, the first abstract action in the plan and a number of search steps.

1. $\mathcal{A} \leftarrow$ an action graph with the first fact level defined by s, the action levels by π and the last fact level by g
2. for $n = n_{init}$ to a predefined number of steps do
3. $\pi \leftarrow$ the plan represented by $\mathcal{A}$
4. if $\mathcal{A}$ is a solution graph then **return** $\langle \pi, nil, n \rangle$;
5. $\sigma \leftarrow$ the flaw at the lowest level of A
6. if σ is an abstract action then **return** $\langle \pi, \sigma, n \rangle$;
7. else
8. $N \leftarrow$ set of actions that are not in τ and whose insertion to/removal from $\mathcal{A}$ fixes σ;
9. select an element from N and modify $\mathcal{A}$ with it
10. **return** $\langle nil, nil, n \rangle$.

Figure 2. Pseudocode of Duet's modified SHOP2 and LPG procedures.

If SHOP2 returns a plan π', Duet inserts it into the current plan π, and updates the HTN w that still needs to be accomplished. Note that at Line 8, if SHOP2 could successfully accomplish g_{SHOP2} without returning any goals to LPG, the returned HTN w' would be the empty HTN, and there would be no update to the HTN w.

If there is a goal g_{LPG} for LPG (see Lines 10–15), Duet invokes LPG with this goal, the current state, the empty plan, and the current values of the tabu list and number of search steps. The modified LPG procedure (Figure 2) is essentially the same stochastic local search procedure of [8] with the following differences: the action graph is initialized using a (possibly non-empty) plan; the initial number of

search steps is an input number instead of zero; the action graphs can contain a new type of flaw (an abstract action), which is handled by just returning it to Duet together with the current plan and number of search steps (Line 6); the search neighborhood is restricted to forbid the insertion of an abstract action in the input tabu list (Line 8). Note that at Line 5 the unsupported preconditions of an abstract action are selected before the action and that, as in [8], the neighborhood selection at Line 9 is randomized and uses a heuristic function.

There are three possible cases when LPG terminates:

- LPG tries to fix a flaw corresponding to an abstract action during its search and needs SHOP2 to decompose this abstract action into smaller tasks. In this case, LPG returns the current partial plan it has (π'), the abstract action for SHOP2 (g_{SHOP2}), and the updated number n of performed search steps.
- LPG generates a solution plan with no abstract actions for the input goals g_{LPG}. In this case, LPG's g_{SHOP2} output is empty.
- LPG fails because the search increases the input number of search steps n above the predefined maximum. In this case, Duet will return failure and can be restarted.

After the run of LPG, Duet updates the current plan π, the current task network w and the current world state s (Lines 12–14).

If there are no immediate goals for SHOP2 or LPG (i.e., if both $g_{\text{SHOP2}} = \emptyset$ and $g_{\text{LPG}} = \emptyset$), then Duet checks whether there are more tasks that need to be decomposed by SHOP2 (Lines 16–18) or any remaining flaws in the current plan that need to be fixed by LPG (Lines 19–23). In the former case, Duet invokes SHOP2 to plan for the HTN w that still needs to be accomplished. Note that, in this case, Duet gives SHOP2 the current partial plan as input (instead of the empty plan as in the above case). This is because if SHOP2 generates a plan for the input abstract action then that plan must be a part of the solution. If the task network becomes empty and the current plan contains a flaw, Duet invokes LPG in its next iteration (see Line 20) with the initial planning problem, except that this time LPG starts with the current partial plan and attempts to generate a solution based on it, rather than starting from the empty plan.

The following theorem establishes Duet's soundness (we omit the proof due to space limitations).

Theorem 1 *Let $P = (s_0, g)$ be a classical planning problem, w_0 be a (possibly empty) HTN to accomplish the goals g, and M be a set of HTN methods. Suppose* Duet(s_0, g, w_0, M) *returns a plan π. Then, π is a solution for the planning problem P.*

Duet is not a complete planner (i.e., it may not find a solution to an input planning problem, although there is one) for two reasons: (1) LPG, as a stochastic local search procedure, may return failure without finding any solution given the number of restarts and the bound parameter on the number of search steps; and (2) the HTNs provided as input for SHOP2 may not be complete, and even if they are, they may prune the solution away.

4 Experimental Evaluation

We compared LPG and SHOP2 with two versions of Duet, one supplied with extremely sparse domain knowledge, and the other with more detailed knowledge of one facet of the Museums domain. The planning operators for LPG in this domain are DRIVE-TRUCK, MOVE-TO-TRUCK, MOVE-FROM-TRUCK, and MOVE. The three move operators define a ToH subdomain where the pegs are the truck area and the two museum pallets. Duet

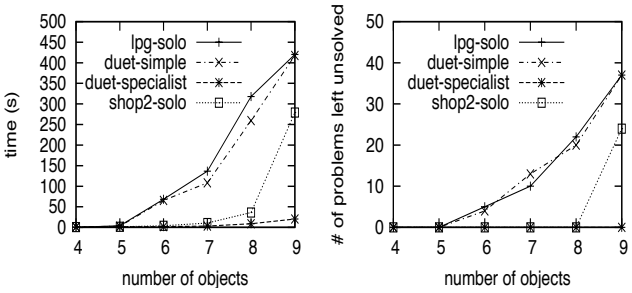

Figure 3. In the first graph, each data point is the average running time on 50 randomly generated problems. The second graph shows how many times the planners failed to return plans within our 500-second deadline; each such failure was scored at 500 seconds in the first graph.

with sparse domain-knowledge, denoted as $\text{Duet}_{\text{Simple}}$, used SHOP2 to choose the order in which to relocate the objects, and LPG to plan how to move each object. Duet with rich domain-knowledge of object-stacking, denoted as $\text{Duet}_{\text{Specialist}}$, provided LPG with abstract actions LOAD and UNLOAD in place of the three primitive move operators. In this version, LPG controls the trucks and chooses which objects to pick up and drop off, where each pick up/drop off request is an abstract action handled by SHOP2.

Table 1. Sizes of the human-generated Museum domain descriptions for LPG, $\text{Duet}_{\text{Simple}}$, and $\text{Duet}_{\text{Specialist}}$, and a SHOP2 HTN.

Planner	*Total lines*	*Total characters*	*Total no. of tokens*
LPG	34	1658	426
$\text{Duet}_{\text{Simple}}$	70	2893	694
$\text{Duet}_{\text{Specialist}}$	157	6573	1534
SHOP2	238	9549	2254

To measure the complexity of the domain knowledge needed by the various planners, Table 1 gives several different measures of the sizes of the domain descriptions used by the various planners. LPG requires only a description of the operators, while SHOP2 requires the operators and HTN methods to solve the Museum planning problems. $\text{Duet}_{\text{Simple}}$ and $\text{Duet}_{\text{Specialist}}$ use a partial set of HTN methods: these methods can be used to generate plans for parts of a Museums planning problem but they cannot solve the problem entirely.

There are three parameters affecting problem difficulty in Museums domain: the number of museums, the connectivity of the museums, and the number of art objects to transport. We performed experiments for each case in which we fixed two of the parameters above and vary the other. In the cases where we varied the first two parameters above, we did not observe a significant change in the relative performance of the planners since these two cases emphasized the truck-movement subproblems in the Museums domain and all of our planners were able to solve truck-movement subproblems easily.

All of our operator and HTN descriptions and other input files regarding our experimental setup are available online.[3]

Figure 3 shows the results of our experiments with varying number of objects where we fixed the number of museums as 3 and generated complete graphs of museums. Each data point in this figure is the average of 50 randomly-generated planning problems. We set the time limit of 500 seconds for the planners and we scored those runs that did not return a plan within the limit at 500 seconds.

With increasing numbers of objects, LPG's local search became frequently trapped into local minima and was unable to produce any

[3] See http://www.cs.umd.edu/~waisbrot/Duet

plan within the given CPU-time limit. For example, LPG began to struggle when the number of objects at any one museum went beyond 4, and out of the 50 9-object problems, it failed on 37.

$\text{Duet}_{\text{Simple}}$ outperformed LPG slightly when they both solved a problem, but generally failed on most of the same problems as LPG, for the same reasons. One advantage of $\text{Duet}_{\text{Simple}}$ over LPG was an increase in reliability. Some of the plans produced by LPG included repetition of actions: picking an object up and then putting it back in the same place multiple times. LPG can be configured to do more planning iterations and produce an improved plan, but $\text{Duet}_{\text{Simple}}$ was able to produce a more directed plan in a single pass, saving time.

$\text{Duet}_{\text{Specialist}}$ dramatically outperformed both $\text{Duet}_{\text{Simple}}$ and LPG because it used domain-specific HTNs to solve the parts of the problem that involve object-stacking. While the object-stacking HTNs required human authoring, we did not give $\text{Duet}_{\text{Specialist}}$ any HTNs for navigating between museums, choosing when objects should be picked up, or choosing where to place objects. $\text{Duet}_{\text{Specialist}}$ solved all of the problems, and in most cases solved them faster than LPG.

To run SHOP2 by itself, we needed to give it HTN methods both for stacking art objects and navigating the truck. It suffered from two major failings, due to the inexperience of the domain writer. First, the HTN methods focused on moving one art object at a time, rather than loading multiple objects onto the truck before attempting delivery. Second, the HTN methods were deeply recursive, so large problems caused the stack to overflow. Although the SHOP2 methods could be improved with additional time and experience, Duet produces good results with less effort on the part of the domain writer.

One exception to Duet's performance was that LPG outperformed it in the easiest problems. This is because of Duet's loose coupling between SHOP2 and LPG, which made Duet easy to implement but made the communication from SHOP2 to LPG very expensive. Duet and SHOP2 are both written in LISP, so calls to SHOP2 to decompose a task were inexpensive, but calls to LPG, which is written in C, required spawning and later destroying a separate shell and process. Because of this expense, the easiest problems were completely solved by LPG before Duet was able to complete the necessary calls between planners. If both planners were packaged as libraries, this inter-planner communication cost would be significantly decreased.

5 Conclusions

We have described Duet, a new planner that incorporates adaptations of two well-known planners, LPG [8] and SHOP2 [14]. Duet combines LPG's domain-independent local search techniques with hierarchical domain knowledge in the form of SHOP2's *Hierarchical Task Networks (HTNs)*. Duet starts with a planning problem consisting of an initial state, a goal condition, and a possibly empty set of tasks. During planning, Duet uses SHOP2 to decompose tasks into smaller subtasks, and LPG to satisfy goal conditions.

Our experiments with Duet in the Museums domain showed that even when Duet had only a small amount of domain-specific knowledge (e.g., "choose the least-fragile object and move it to the target museum first"), it still solved planning problems faster, on average, than LPG. With more problem-solving knowledge (e.g., how to properly manipulate stacks of art objects), Duet outperformed both LPG and SHOP2, in terms of both speed and the number of successfully solved problems. To get SHOP2 to perform better, significantly more human effort would have been needed to improve its knowledge base.

We are currently starting a further experimental evaluation of Duet. So far, we have run experiments using the Storage domain from the 2006 International Planning Competition and obtained similar results to those shown here.

Although the Duet planning procedure we described in this paper is based on SHOP2 and LPG, the ideas we described here could be easily generalized to combine any planner that uses domain-specific knowledge with any domain-independent classical planner. Thus, a possible future direction is to extend Duet to work with planners such as FF [11], FastDownward [10], and SGPlan [5].

Another direction is a tighter integration of SHOP2 and LPG, which would probably yield more efficient planning in Duet. Not only would this reduce the communication overhead between the planners, it would allow Duet to provide a richer form of "knowledge transfer;" the decisions that one of the planners make during its planning time will be more closely dependent on the domain knowledge that the other one could provide.

Acknowledgments. This work was supported in part by DARPA's Transfer Learning and Integrated Learning programs and NSF grant IIS0412812. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

REFERENCES

[1] F. Bacchus and F. Kabanza, 'Using temporal logics to express search control knowledge for planning', *Artificial Intelligence*, **116**(1-2), 123–191, (2000).
[2] A. L. Blum and M. L. Furst, 'Fast planning through planning graph analysis', *Artificial Intelligence*, **90**(1-2), 281–300, (1997).
[3] B. Bonet and H. Geffner, 'Planning as heuristic search: New results', in *ECP*, Durham, UK, (1999).
[4] Adi Botea, Markus Enzenberger, Martin Muller, and Jonathan Schaeffer, 'Macro-ff: Improving ai planning with automatically learned macro-operators', *JAIR*, **24**, 581–621, (2005).
[5] Y. Chen, C. Hsu, and B. Wah, 'Temporal planning using subgoal partitioning and resolution in SGPlan', *JAIR*, **26**, 323–369, (2006).
[6] K. Currie and A. Tate, 'O-Plan: The open planning architecture', *Artificial Intelligence*, **52**(1), 49–86, (1991).
[7] R. E. Fikes and N. Nilsson, 'Strips: A new approach to the application of theorem proving to problem solving', *Artificial Intelligence*, **5**(2), 189–208, (1971).
[8] A. Gerevini, A. Saetti, and I. Serina, 'Planning through Stochastic Local Search and Temporal Action Graphs', *JAIR*, **20**, 239–290, (2003).
[9] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.
[10] M. Helmert, 'The Fast Downward planning system', *JAIR*, **26**, 191–246, (2006).
[11] J. Hoffmann and B. Nebel, 'The FF planning system: Fast plan generation through heuristic search', *JAIR*, **14**, 253–302, (2001).
[12] S. Kambhampati, 'Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain theories', in *AAAI*, Vancouver, Canada, (2007).
[13] J. Kvarnström and P. Doherty, 'TALplanner: A temporal logic based forward chaining planner', *Annals of Mathematics and Articial Intelligence*, **30**, 119–169, (2001).
[14] D. Nau, T. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman, 'SHOP2: An HTN planning system', *JAIR*, **20**, 379–404, (2003).
[15] N. Nguyen, S. Kambhampati, and R. Nigenda, 'Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search', *Artificial Intelligence*, **135**(1-2), 73 – 124, (2002).
[16] S. Yoon and S. Kambhampati, 'Towards Model-lite Planning: A Proposal For Learning & Planning with Incomplete Domain Models', in *Proc. ICAPS-07 Workshop on AI Planning and Learning*, Providence, RI, (2007).
[17] D. E. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann, San Mateo, CA, 1988.