The OAKPLAN Case-Based Planner*

Ivan Serina
Free University of Bozen-Bolzano
Viale Ratisbona, 16
I-39042 Bressanone, Italy
ivan.serina@unibz.it

Abstract

Case-based planning can take advantage of former problem-solving experiences by storing in a plan library previously generated plans that can be reused to solve similar planning problems in the future.

In this paper we describe an innovative case-based planning system, called OAKPLAN, which is able to efficiently retrieve planning cases from plan libraries with more than ten thousands elements, heuristically choose a suitable candidate (possibly the best one) and adapt it to provide a good quality solution plan similar to the one retrieved from the case base.

Overall, we show that OAKPLAN is competitive with state of the art plan generation systems in terms of number of problems solved, CPU time, plan difference values and plan quality when cases similar to the current planning problem are available into the plan library.

Introduction

Planning is a process which usually involves the use of a lot of resources. The efficiency of planning systems can be improved by avoiding the repetition of the planning effort whenever it is not strictly necessary. In *C*ase-*B*ased *P*lanning (CBP), previously generated plans are stored as cases in memory and can be reused to solve similar planning problems in the future. CBP can save considerable time over planning from scratch, thus offering a potential (heuristic) mechanism for handling intractable problems. Similarly to other *C*ase-*B*ased *R*easoning (CBR) systems, CBP is based on two assumptions on the nature of the world (Leake 1996). The first assumption is that the world is regular: similar problems have similar solutions; as a consequence, solutions for similar problems are a useful starting point for new problem-solving. The second assumption is that the types of problems an agent encounters tend to recur; hence future problems are likely to be similar to current problems.

In this paper we present some data structures and new matching functions that efficiently address the problem of matching planning instances, which is NP-hard in the general case. These functions lead to a new case-based planner called OAKPLAN (acronym of *O*bject *A*ssignment *K*ernel case-based *plan*ner), which is competitive with state of the art plan generation systems when sufficiently similar reuse candidates can be chosen.

In the following sections we examine the different steps required by the Retrieval, Evaluation and Adaptation phases in detail, in particular we present our *Optimal Assignment Kernel*[1] as a symmetric and positive definite similarity measure for directed graph structures. Then we examine the results produced by OAKPLAN in comparison with four state of the art plan generation systems. Finally, we give the conclusions and indicate future work.

*An extended version of this paper has been published in the Artificial Intelligence Journal vol 174 (2010), pp. 1369-1406.

[1]For an introduction to kernel functions related concepts and notation, the reader is referred to Scholkopf and Smola's book (Scholkopf & Smola 2001).

Plan Retrieval

Although the plan adaptation phase is the central component of a CBP system, the retrieval phase critically affects the system performance too. As a matter of fact the retrieval time is a component of the *total adaptation time* and the *quality* of the retrieved plan is fundamental for the performance of the successive adaptation phase. To the end of applying a reuse technique, it is necessary to provide a plan library from which "sufficiently similar" reuse candidates can be chosen. In this case, "sufficiently similar" means that reuse candidates have a large number of initial and goal facts in common with the new instance. However, one may also want to consider the reuse candidates that are similar to the new instance after the objects of the selected candidates have been systematically renamed. Following Nebel & Koehler's formalisation (Nebel & Koehler 1995), we will have a closer look at this matching problem.

Object Matching We assume that the operators are ordinary STRIPS operators using variables, moreover we use *a many-sorted logic* (Chien, Hudli, & Palakal 1998) in order to reduce the search space for the matching process. If there are two planning instances

$$\Pi' = \langle \mathcal{P}_r(\mathbf{O}', \mathbf{P}'), \mathcal{I}', \mathcal{G}', \mathcal{O}p' \rangle \qquad \Pi = \langle \mathcal{P}_r(\mathbf{O}, \mathbf{P}), \mathcal{I}, \mathcal{G}, \mathcal{O}p \rangle^2$$

such that (without loss of generality)

$$\mathbf{O}' \subseteq \mathbf{O} \qquad \mathbf{P}' = \mathbf{P} \qquad \mathcal{O}p' \subseteq \mathcal{O}p$$

then a *mapping*, or *matching function*, from Π' to Π is a function

$$\mu : \mathbf{O}' \rightarrow \mathbf{O}$$

The mapping is extended to ground atomic formulae and sets of such formulae in the canonical way, i.e.,

$$\mu(p(c_1 : t_1, ..., c_n : t_n)) = p(\mu(c_1) : t_1, ..., \mu(c_n) : t_n)$$

$$\mu(\{p_1(..), ..., p_m(..)\}) = \{\mu(p_1(..)), ..., \mu(p_m(..))\}$$

If there exists a bijective matching function μ from Π' to Π such that $\mu(G') = G$ and $\mu(I') = I$, then it is obvious that a solution plan π' for Π' can be directly reused for solving Π since Π' and Π are identical within a renaming of constant symbols, i.e., $\mu(\pi')$ solves Π. Even if μ does not match all goal and initial-state facts, $\mu(\pi')$ can still be used as a starting point for the adaptation process that can solve Π.

In order to measure the similarity between two objects, it is intuitive and usual to compare the features which are common to both objects (Lin 1998). The Jaccard similarity coefficient used in information retrieval is particularly interesting. Here we examine an extended version that considers two pairs of disjoint sets:

$$simil_{\mu}(\Pi', \Pi) = \frac{|\mu(\mathcal{G}') \cap \mathcal{G}| + |\mu(\mathcal{I}') \cap \mathcal{I}|}{|\mathcal{G}| + |\mu(\mathcal{I}')|}. \qquad (1)$$

[2]$\mathcal{P}_r$ is a finite set of ground atomic propositional formulae, $\mathcal{I} \subseteq \mathcal{P}_r$ is the initial state, $\mathcal{G} \subseteq \mathcal{P}_r$ is the goal state and $\mathcal{O}p$ is a finite set of operators.

Using $simil_\mu$ we obtain a value equal to 1 when there exists a mapping μ s.t. $\forall f \in I'$, $\mu(f) \in I$ (to guarantee the applicability of π') *and* $\forall g \in G$, $\exists g' \in G'$ s.t. $g = \mu(g')$ (to guarantee the achievement of the goals of the current planning problem).

It should be noted that this matching problem has to be solved for each potentially relevant candidate in the plan library to select the corresponding best reuse candidate. For this reason, the efficiency of the matching component is crucial for the overall system performance. Unfortunately, similarly to Nebel & Koehler's analysis (Nebel & Koehler 1995), it is quite easy to show that this matching problem is an NP-hard problem.

In order to perform an efficient matching between the objects of a planning case and the objects of the current planning problem we define a particular labeled graph data structure called *Planning Encoding Graph* which encodes the initial and goal facts of a single planning problem Π. The *Planning Encoding Graph* of a planning problem $\Pi(I, G)$ is built using the corresponding initial and goal facts. In particular for each propositional initial fact $\mathbf{p} = p(c_1 : t_1, ..., c_n : t_n) \in I$ we define a data structure called *Initial Fact Encoding Graph* which corresponds to a graph that represents $\mathbf{p}$. More precisely:

Definition 1 *Given a propositional typed initial fact* $\mathbf{p} = p(c_1 : t_1, ..., c_n : t_n) \in I$ *of* Π*, the* **Initial Fact Encoding Graph** $\mathcal{E}^I(\mathbf{p}) = (V_\mathbf{p}, E_\mathbf{p}, \lambda_\mathbf{p})$ *of fact* $\mathbf{p}$ *is a directed labeled graph where*

- $V_\mathbf{p} = \{I_p, c_1, ..., c_n\} \subseteq \mathbf{V}_\Pi$;
- $E_\mathbf{p} = \{[I_p, c_1], [c_1, c_2], [c_1, c_3], ..., , [c_{n-1}, c_n]\} =$

$$= [I_p, c_1] \cup \bigcup_{i=1,...,n;\ j=i+1,...,n} [c_i, c_j]$$

- $\lambda_\mathbf{p}(I_p) = \{I_p\}, \lambda_\mathbf{p}(c_i) = \{t_i\}$ `with` $i = 1, ..., n$;
- $\lambda_\mathbf{p}([I_p, c_1]) = \{I_p^{0,1}\}; \quad \forall [c_i, c_j] \in E_\mathbf{p},\ \lambda_\mathbf{p}([c_i, c_j]) = \{I_p^{i,j}\};$

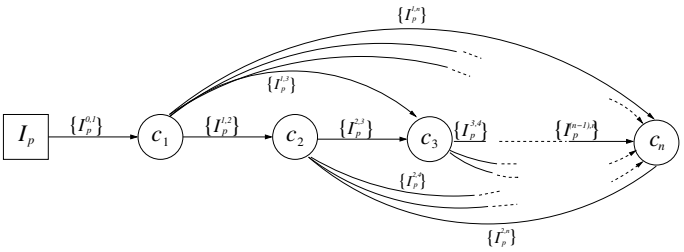
where $V_\mathbf{p}$ is the set of vertices of $\mathcal{E}^I(\mathbf{p})$, $E_\mathbf{p} \subseteq V_\mathbf{p} \times V_\mathbf{p}$ is the set of directed edges and $\lambda : V_\mathbf{p} \cup E_\mathbf{p} \rightarrow \wp_s(L_\lambda)$ is a function assigning labels to vertices and edges. I.e. the first node of the graph $\mathcal{E}^I(\mathbf{p})$, see Figure 1, is the initial fact relation node I_p labeled with the multiset $\lambda_\mathbf{p}(I_p) = \{(I_p, 1)\} = \{I_p\}$,[3] it is connected to a direct edge to the second node of the graph, the concept node c_1, which is labeled by sort t_1 (i.e. $\lambda_\mathbf{p}(c_1) = \{(t_1, 1)\} = \{t_1\}$); the node c_1 is connected with the third node of the graph c_2 which is labeled by sort t_2 (i.e. $\lambda_\mathbf{p}(c_2) = \{(t_2, 1)\} = \{t_2\}$) and with all the remaining concept nodes, the third node of the graph c_2 is connected with c_3, c_4, ...,c_n and so on. The first edge of the graph $[I_p, c_1]$ is labeled by the multiset $\{I_p^{0,1}, 1\} = \{I_p^{0,1}\}$, similarly a generic edge $[c_i, c_j] \in E_\mathbf{p}$ is labeled by the multiset $\{I_p^{i,j}\}$.

Similarly to Definition 1 we define the **Goal Fact Encoding Graph** $\mathcal{E}^G(\mathbf{q})$ of the fact $\mathbf{q} = q(c'_1 : t'_1, ..., c'_m : t'_m) \in G$ using $\{G_q\}$ for the labeling procedure.

Given a planning problem Π with initial and goal states I and G, the **Planning Encoding Graph** of Π, that we indicate as $\mathcal{E}_\Pi$, is a directed labeled graph derived by the encoding graphs of the initial and goal facts:

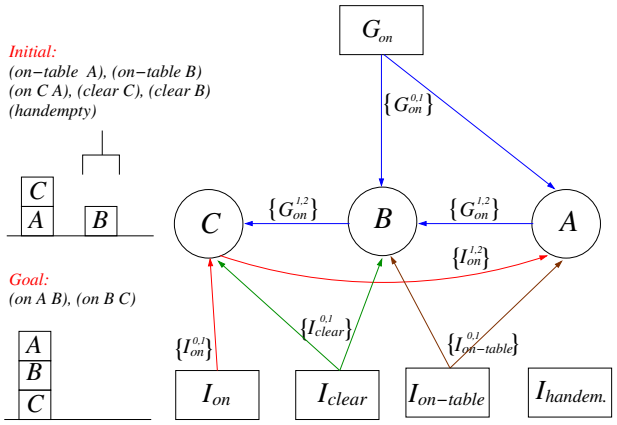
$$\mathcal{E}_{\Pi(I,G)} = \bigcup_{\mathbf{p} \in I} \mathcal{E}^I(\mathbf{p}) \cup \bigcup_{\mathbf{q} \in G} \mathcal{E}^G(\mathbf{q}) \qquad (2)$$

[3] In the following we indicate the multiset $\{(x, 1)\}$ as $\{x\}$ for sake of simplicity.



$$\lambda(I_p) = \{(I_p, 1)\} = \{I_p\}, \quad \lambda(c_1) = \{(t_1, 1)\} = \{t_1\},$$
$$..., \lambda(c_n) = \{(t_n, 1)\} = \{t_n\}$$

Figure 1: Initial Fact Encoding Graph $\mathcal{E}^I(\mathbf{p})$ of the propositional initial fact $\mathbf{p} = p(c_1 : t_1, ..., c_n : t_n)$



$$\lambda(A) = \{(Obj, 3)\}, \lambda(B) = \{(Obj, 4)\} \text{ and}$$
$$\lambda(C) = \{(Obj, 3)\}$$

Figure 2: Planning Encoding Graph for the Sussman Anomaly planning problem in the BlocksWorld domain.

i.e. the Planning Encoding Graph of $\Pi(I, G)$ is a graph obtained by merging the Initial and Goal Fact Encoding Graphs. For simplicity in the following we visualise it as a three-level graph. The first level is derived from the predicate symbols of the initial facts, the second level encodes the objects of the initial and goal states and the third level shows the goal fact nodes derived from the predicate symbols of the goal facts.[4]

Figure 2 illustrates the Planning Encoding Graph for the Sussman anomaly planning problem in the BlocksWorld domain. The nodes of the first and third levels are the initial and goal fact relation nodes: the vertices I_{on}, I_{clear} and $I_{on-table}$ are derived by the predicates of the initial facts, while G_{on} by the predicates of the goal facts. The nodes of the second level are concept nodes which represent the objects of the current planning problem A, B and C, where the label "Obj" corresponds to their type. The initial fact "*(on C A)*" determines two arcs, one connecting I_{on} to the vertex C and the second connecting C to A; the labels of these arcs are derived from the predicate symbol "*on*" determining the multisets $\{I_{on}^{0,1}\}$ and $\{I_{on}^{1,2}\}$ respectively. In the same way the other arcs are defined. Moreover since there is no overlapping among the

[4] Following the conceptual graph notation, the first and third level nodes correspond to initial and goal fact relation nodes, while the nodes of the second level correspond to concept nodes representing the objects of the initial and goal states.

edges of the Initial and Goal Fact Encoding Graphs, the multiplicity of the edge label multisets is equal to 1; on the contrary the label multisets of the vertices associated to the objects are: $\lambda(A) = \{(Obj, 3)\}$, $\lambda(B) = \{(Obj, 4)\}$ and $\lambda(C) = \{(Obj, 3)\}$.

This graph representation can give us a detailed description of the "topology" of a planning problem without requiring any a priori assumptions on the relevance of certain problem descriptors for the whole graph. In the following we examine a procedure based on graph *degree sequences* that is useful to derive an upper bound on the size of the Maximum Common Edge Subgraph (MCES) of two graphs in an efficient way. Then we present an algorithm based on *Kernel Functions* that allows to compute an approximate matching of two graphs in polynomial time.

Screening Procedure As explained previously, the retrieval phase could be very expensive from a computational point of view; so we have developed a screening procedure that can be used in conjunction with an object matching algorithm.

First, the set of vertices in each graph is partitioned into l partitions by label type, and then sorted in a non-increasing total order by degree.[5] Let L_1^i and L_2^i denote the sorted degree sequences of a partition i in the planning encoding graphs G_1 and G_2, respectively. An upper bound on the number of vertices $Vertices(G_1, G_2)$ and edges $Edges(G_1, G_2)$ of the MCES graph can be computed as follows:

$$Vertices(G_1, G_2) = \sum_{i=1}^{l} min\left(|L_1^i|, |L_2^i|\right) \tag{3}$$

$$Edges(G_1, G_2) = \left\lfloor \sum_{i=1}^{l} \sum_{j=1}^{min\left(|L_1^i|, |L_2^i|\right)} \frac{min\left(|E(v_1^{i,j})|, |E(v_2^{i,j})|\right)}{2} \right\rfloor \tag{4}$$

where $v_1^{i,j}$ indicates the j-th element (vertex) of the L_1^i sorted degree sequence and $E(v_1^{i,j})$ indicates the set of arcs connected to the vertex $v_1^{i,j}$. An upper bound on the similarity between G_1 and G_2 can be expressed using Johnson's similarity coefficient (Johnson 1985):

$$simil^{ds}(G_1, G_2) = \frac{(Vertices(G_1, G_2) + Edges(G_1, G_2))^2}{(|V(G_1)| + |E(G_1)|) \cdot (|V(G_2)| + |E(G_2)|)} = \\ = \frac{(Vertices(G_1, G_2) + Edges(G_1, G_2))^2}{|G_1| \cdot |G_2|} \tag{5}$$

For screening purposes, it is only necessary to specify a minimum acceptable value for the MCES based graph similarity measure. If the value determined by $simil^{ds}(G_1, G_2)$ is less than the minimum acceptable similarity, then the object matching comparison can be avoided. This procedure can be performed by using the quick sort algorithm in $O(n \cdot logn)$ time, where $n = max_i\left(|L_1^i|, |L_2^i|\right)$.

Kernel Functions for Object Matching As previously exposed `obj_match` is an NP-hard problem and its exact resolution is infeasible from a computational point of view also for a limited number of candidates in the case base. In the following we present an approximate evaluation based on kernel functions. Our kernel functions are inspired by Fröhlich et al.'s work (Fröhlich *et al.* 2006; Fröhlich *et al.* 2005) on kernel functions for molecular structures. Their goal is to define a kernel function which measures the degree of similarity between two chemical structures which are encoded as undirected labeled graphs. Our goal is to define a matching function among the objects of two planning problems encoded as directed graphs.

Let us assume we have two graphs G and G', which have vertices $v_1, ..., v_n$ and $u_1, ..., u_m$ respectively. Let us further assume we have a kernel function k, which compares a pair of vertices v_i, u_j from both graphs, including information on their neighbourhoods. We now want to assign each vertex of the smaller of both graphs to exactly one vertex of the bigger one such that the overall similarity score, i.e., the sum of kernel values between individual vertices, is maximised. Mathematically this can be formulated as follows: let ζ denote a permutation of an n-subset of natural numbers $1, ..., m$, or a permutation of an m-subset of natural numbers $1, ..., n$, respectively. Then we are looking for the quantity

$$\mathcal{K}(G, G') = \begin{cases} max_\zeta \sum_{h=1}^{m} k(v_{\zeta(h)}, u_h) & \texttt{if } n \geq m \\ max_\zeta \sum_{h=1}^{n} k(v_h, u_{\zeta(h)}) & \texttt{otherwise} \end{cases} \tag{6}$$

$\mathcal{K}$ is a valid kernel function, as shown by (Fröhlich *et al.* 2005), and hence a similarity measure for graphs. Implicitly it computes a dot product between two vector representations of graphs in some Hilbert space.

We now have to define the kernel function k. For this purpose let us suppose we have two kernel functions k_v and k_e which compare the vertex and edge labels $\lambda(\cdot)$, respectively. In the following $e_j(v)$ denotes the j-th edge of the vertex v, while $n_j(v)$ denotes the node adjacent to the vertex v associated to the j-th edge $e_j(v)$. In the same way $e_j^{i/o}(v)$ denotes the j-th incoming/outgoing edge, while $n_j^{i/o}(v)$ denotes the direct predecessor/successor of the vertex v associated to the j-th incoming/outgoing edge $e_j^{i/o}(v)$. $\mathcal{N}(v_j)$ denotes the set of vertices adjacent to the vertex v_j, while $E(v_j)$ denotes the set of incoming and outgoing edges of vertex v_j. Similarly $\mathcal{N}^{i/o}(v_j)$ denotes the set of direct predecessor/successor vertices of the vertex v_j.

Given a pairs of vertices v and u, we use the kernel function $k_v(v, u) = \gamma_0(v, u) \cdot \frac{|\lambda(v) \cap \lambda(u)|}{|\lambda(v) \cup \lambda(u)|}$, where $\gamma_0(v, u)$ is equal to 1.1 if u and v correspond to the same object (it is verified considering the names of the objects represented by vertices u and v), otherwise it is equal to 1.0 . The γ_0 coefficient has been introduced in our kernel functions in order to allow a greater stability in the activity assignment which is useful especially when human agents are handled by the planner. For example, in a logistic domain, we would like the drivers to be assigned the same set of activities as much as possible. While for pairs of edges we use $k_e(e_k(v), e_j(u)) = \frac{|\lambda(e_k(v)) \cap \lambda(e_j(u))|}{|\lambda(e_k(v)) \cup \lambda(e_j(u))|}$ if $e_k(v)$ and $e_j(u)$ are both incoming or outgoing edges of the vertices v and u, otherwise $k_e(e_k(v), e_j(u))$ is equal to 0. Formally, this corresponds to the multiplication by a so-called δ-kernel.

We define the *base kernel* between two vertices v and u, including their direct neighbourhoods as

$$k_{base}(v, u) = k_v(v, u) + \tag{7}$$
$$+ \frac{1}{|\mathcal{N}^i(v)| \cdot |\mathcal{N}^i(u)|} \sum_{h,h'} k_v\left(n_h^i(v), n_{h'}^i(u)\right) \cdot k_e\left(e_h^i(v), e_{h'}^i(u)\right) +$$
$$+ \frac{1}{|\mathcal{N}^o(v)| \cdot |\mathcal{N}^o(u)|} \sum_{h,h'} k_v\left(n_h^o(v), n_{h'}^o(u)\right) \cdot k_e\left(e_h^o(v), e_{h'}^o(u)\right)$$

[5] The degree or valence of a vertex v of a graph G is the number of edges which touch v.

This means that the similarity between two vertices consists of two parts: first the similarity between the labels of the vertices and second the similarity of the neighbourhood structure. It follows that the similarity of each pair of neighbours $n_h^{i/o}(v), n_{h'}^{i/o}(u)$ is weighed by the similarity of the edges leading to them. The normalisation factors before the sums are introduced to ensure that vertices with a higher number of arcs do not automatically achieve a higher similarity. Hence we divide the sums by the number of the addends in them. It is also interesting to point out that the previous definition is just a classical convolution kernel as introduced by Haussler (Haussler 1999).

In the following we define a more accurate kernel R_1, which compares all the direct neighbours of the vertices (v, u) as the optimal assignment kernel between all the neighbours of v and u and the edges leading to them so that we can to improve the similarity values that can be obtained simply using k_{base}; more precisely $R_1(v, u)$ is equal to:

$$\begin{array}{l} \frac{1}{|E(v)|} \max_{\zeta} \sum_{i=1}^{|E(v)|} k_v\left(n_{\zeta(i)}(v), n_i(u)\right) \cdot k_e\left(e_{\zeta(i)}(v), e_i(u)\right) \texttt{ if } |E(v)| \geq |E(u)| \\ \frac{1}{|E(u)|} \max_{\zeta} \sum_{i=1}^{|E(u)|} k_v\left(n_i(v), n_{\zeta(i)}(u)\right) \cdot k_e\left(e_i(v), e_{\zeta(i)}(u)\right) \texttt{ otherwise} \end{array} \tag{8}$$

Similarly to the graph kernel $\mathcal{K}$ of equation (6), the intuition behind this kernel function is that the similarity between two nodes depends not only on the nodes structure but also on the matching of the corresponding neighbourhoods; i.e., two nodes are more similar if their neighbourhood elements are connected in a more similar way in both nodes.

Of course it would be beneficial not to consider the match of direct neighbours only, but also that of indirect neighbours and vertices having a larger topological distance. For this purpose we can evaluate R_1 not at (v, u) only, but also at all pairs of neighbours, indirect neighbours and so on, up to some topological distance L. The weighed average of all these values corresponds to the weighed mean match of all indirect neighbours and vertices of a larger topological distance. Adding them to $k_v(v, u)$ leads to the following definition of the neighbourhood kernel $k_{\mathcal{N}}$:

$$k_{\mathcal{N}}(v, u) = k_v(v, u) + \gamma(1) R_1(v, u) + \sum_{l=2}^{L} \gamma(l) R_l(v, u) \tag{9}$$

where $\gamma(l)$ denotes a decay parameter which reduces the influence of neighbours that are at topological distance l.[6] Similarly, R_l denotes the average of all R_1 evaluated for neighbours at distance l and it is computed from R_{l-1} via the recursive relation and we define $R_l(v, u)$ as:

$$\begin{array}{l} \frac{1}{|\mathcal{N}^i(v)| \cdot |\mathcal{N}^i(u)|} \sum_{h,h'} R_{l-1}\left(n_h^i(v), n_{h'}^i(u)\right) \cdot k_e\left(e_h^i(v), e_{h'}^i(u)\right) + \\ + \frac{1}{|\mathcal{N}^o(v)| \cdot |\mathcal{N}^o(u)|} \sum_{h,h'} R_{l-1}\left(n_h^o(v), n_{h'}^o(u)\right) \cdot k_e\left(e_h^o(v), e_{h'}^o(u)\right) \end{array} \tag{10}$$

i.e., we can compute $k_{\mathcal{N}}(v, u)$ by iteratively revisiting all direct neighbours of v and u. The first addend in equation (9) takes into account the nodes (v, u), while the second addend takes into account the direct neighbours of (v, u) computing the $R_1(v, u)$ kernel function, then the next addend (i.e. $\gamma(2) \cdot R_2(v, u)$) computes the average of the match of all neighbours which have topological distance 2 by evaluating R_1 for all direct neighbours of (v, u). The fourth addend (i.e. $\gamma(3) \cdot R_3(v, u)$) does the same for all neighbours with topological distance 3. Finally, the last addend considers all neighbours which have topological distance L by evaluating R_1 for all neighbours at topological distance $L - 1$.

To briefly summarise, our approach works as follows: we first compute the similarity of all vertex and edge features using the kernels k_v and k_e. Having these results we can compute the match of direct neighbours R_1 for each pair of vertices from both graphs by means of equation (8). From R_1 we can compute $R_2, ..., R_L$ by iteratively revisiting all direct neighbours of each pair of vertices and computing the recursive update formula (10). Having k_v and $R_1, ..., R_L$ directly gives us $k_{\mathcal{N}}$, the final similarity score for each pair of vertices, which includes structural information as well as neighbourhood properties. With $k_{\mathcal{N}}$ we can finally compute the optimal assignment kernel between two graphs G and G' using Equation (6). Moreover (6) can be calculated efficiently by using the *Hungarian method* (Kuhn 1955) in $O(n^3)$, where n is the maximum number of vertices of both graphs.

The kernel functions k_{base} and $k_{\mathcal{N}}$ can be used in equation (6) to define the optimal assignment kernels $\mathcal{K}_{base}$ and $\mathcal{K}_{\mathcal{N}}$ respectively. Our optimal assignment kernel functions also define a permutation ζ that allows to easily determine the matching function μ associating each object in the smaller planning problem to exactly one object in the other planning problem.

As it will be described in the next section, in OAKPLAN both $\mathcal{K}_{base}$ and $\mathcal{K}_{\mathcal{N}}$ have been used; $\mathcal{K}_{base}$, which has a lower computational complexity, has been used in order to prune unpromising case base candidates. It allows to define a first matching function μ_{base} and the corresponding similarity function $simil_{\mu_{base}}$, as described in the following section. On the other hand $\mathcal{K}_{\mathcal{N}}$ has been used to define a final matching function μ and the corresponding similarity function $simil_{\mu}$.

[6] The $\gamma(\cdot)$ function used in our experimental evaluation is defined in the experimental evaluation section.

Algorithm EVALUATEPLAN
Input: a planning problem $\Pi = (I, G)$, an input plan π and an adaptation cost limit $Climit$
Output: a relaxed plan to adapt π in order to resolve Π

```
1.  CState= I;  Rplan = ∅
2.  forall a ∈ π_i do
3.      if ∃f ∈ Pre(a) s.t f ∉ CState then
4.          Rplan =RELAXEDPLAN(Pre(a), CState, Rplan)
5.      if |Rplan| > Climit then
6.          return Rplan
7.      CState = (CState/Del(a)) ∪ Add(a)
8.  if ∃g ∈ G s.t g ∉ CState then
        Rplan =RELAXEDPLAN(G, CState, Rplan)
9.  return Rplan
```

Figure 3: Algorithm to evaluate the ability of π to solve the planning problem Π.

Plan Evaluation Phase

The purpose of plan evaluation is that of defining the capacity of a plan π to resolve a particular planning problem. It is performed by simulating the execution of π and identifying the unsupported preconditions of its actions; in the same way the presence of unsupported goals is identified. The plan evaluation function could be easily defined as the number of inconsistencies in the current planning problem. Unfortunately this kind of evaluation considers a uniform cost in order to resolve the different inconsistencies and this assumption is generally too restrictive. Then our system considers a more accurate inconsistency evaluation criterion so as to improve the plan evaluation metric. The inconsistencies related to unsupported

Algorithm RETRIEVEPLAN
Input: a planning problem Π, a case base $C = \langle \Pi_i, \pi_i \rangle$
Output: candidate plan for the adaptation phase
1.1. π_R = EVALUATE_PLAN(Π, EMPTY_PLAN, ∞)
1.2. Define the set of initial relevant facts of Π using π_R:
$I_{\pi_R} = I \cap \bigcup_{a \in \pi_R} pre(a)$
1.3. Compute the Planning Encoding Graphs $\mathcal{E}_\Pi$ and $\mathcal{E}_{\Pi_R}$ of $\Pi(I, G)$ and $\Pi_R(I_{\pi_R}, G)$ respectively, and the degree sequences $L^j_{\Pi_R}$
1.4. `forall` $\Pi_i \in C$ `do`
1.5. $simil_i = simil^{ds}(\mathcal{E}_{\Pi_i}, \mathcal{E}_{\Pi_R})$
1.6. `push`$((\Pi_i, simil_i)), queue)$
1.7. $best\_ds = max(best\_ds, simil_i)$
2.1. `forall` $(\Pi_i, simil_i) \in queue$ s.t. $best\_ds - simil_i \leq limit$ `do`*
2.2. Load the Planning Encoding Graph $\mathcal{E}_{\Pi_i}$ and compute the matching function μ_{base} using $\mathcal{K}_{base}(\mathcal{E}_{\Pi_i}, \mathcal{E}_\Pi)$
2.3. `push`$((\Pi_i, \mu_{base}), q_1)$
2.4. $best\_\mu_{base} = max(best\_\mu_{base}, simil_{\mu_{base}}(\Pi_i, \Pi))$
3.1. `forall` $(\Pi_i, \mu_{base}) \in q_1$ s.t. $best\_\mu_{base} - simil_{\mu_{base}}(\Pi_i, \Pi) \leq limit$ `do`
3.2. Compute the matching function $\mu_\mathcal{N}$ using $\mathcal{K}_\mathcal{N}(\mathcal{E}_{\Pi_i}, \mathcal{E}_\Pi)$
3.3. `if` $simil_{\mu_\mathcal{N}}(\Pi_i, \Pi) \geq simil_{\mu_{base}}(\Pi_i, \Pi)$ `then` $\mu_i = \mu_\mathcal{N}$ `else` $\mu_i = \mu_{base}$
3.4. `push`$((\Pi_i, \mu_i), q_2)$
3.5. $best_S = max(best_S, simil_{\mu_i}(\Pi_i, \Pi))$
4.1. $best_C = \alpha_G \cdot |\pi_R|$; `best_plan` =EMPTY_PLAN
4.2. `forall` $(\Pi_i, \mu_i) \in q_2$ s.t. $best_S - simil_{\mu_i}(\Pi_i, \Pi) \leq limit$ `do`
4.3. Retrieve π_i from C
4.4. $cost_i = |$EVALUATEPLAN$(\Pi, \mu_i(\pi_i),\ best_C \cdot simil_{\mu_i}(\Pi_i, \Pi))|$
4.5. `if` $best_C \cdot simil_{\mu_i}(\Pi_i, \Pi) > cost_i$ `then`
4.6. $best_C = cost_i / simil_{\mu_i}(\Pi_i, \Pi)$
4.7. `best_plan` = $\mu_i(\pi_i)$
5.1. `return best_plan`

* We limited this evaluation to the best 700 cases of $queue$.

Figure 4: Algorithm to find a suitable plan for the adaptation phase from a set of candidate cases or the empty plan (in case the "generative" approach is considered more suitable).

facts are evaluated by computing a relaxed plan starting from the corresponding state and using the RELAXEDPLAN algorithm in LPG (Gerevini, Saetti, & Serina 2003). The number of actions in the relaxed plan determines the difficulty to make the selected inconsistencies supported; the number of actions in the final relaxed plan determines the accuracy of the input plan π to solve the corresponding planning problem.

Figure 3 describes the main steps of the EVALUATEPLAN function. For all actions of π (if any), it checks if at least one precondition is not supported. In this case it uses the RELAXEDPLAN algorithm (step 4) so as to identify the additional actions required to satisfy the unsupported preconditions. If $Rplan$ contains a number of actions greater than $Climit$ we can stop the evaluation, otherwise we update the current state *CState* (step 7). Finally we examine the goal facts G (step 8) to identify the additional actions required to satisfy them, if necessary.

Figure 4 describes the main steps of the retrieval phase. We initially compute a relaxed plan π_R for Π (step 1.1) using the EVALUATEPLAN function on the empty plan which is needed so as to define the *generation* cost of the current planning problem Π (step 4.1)[7] and an *estimate* of the initial state relevant facts (step 1.2). In fact we use the relaxed plan π_R so as to filter out the irrelevant facts from the initial state description.[8] This could be easily done by considering all the preconditions of the actions of π_R:

$$I_{\pi_R} = I \cap \bigcup_{a \in \pi_R} pre(a).$$

Then in step 1.3 the Planning Encoding Graph of the current planning problem Π and the degree sequences that will be used in the screening procedure are precomputed. Note that the degree sequences are computed considering the Planning Encoding Graph $\mathcal{E}_{\Pi_R}$ of the planning problem $\Pi_R(I_{\pi_R}, G)$ which uses I_{π_R} instead of I as initial state. This could be extremely useful in practical applications when automated tools are used to define the initial state description without distinguishing among relevant and irrelevant initial facts.

Steps 1.4 – 1.7 examine all the planning cases of the case base so as to reduce the set of candidate plans to a suitable number. It is important to point out that in this phase it is not necessary to retrieve the complete planning encoding graphs of the case base candidates $G_{\Pi'}$ but only their sorted degree sequences $L^i_{\Pi'}$ which are precomputed and stored in the case base. On the contrary the planning encoding graph and the degree sequences of the input planning problem are only computed in the initial preprocessing phase (step 1.3).

All the cases with a similarity value sufficiently close[9] to the best degree sequences similarity value ($best\_ds$) are examined further on (steps 2.1–2.4) using the $\mathcal{K}_{base}$ kernel function. Then all the cases selected at steps $2.x$ with a similarity value sufficiently close to the best $simil_{\mu_{base}}$ similarity value ($best\_\mu_{base}$) (step 3.1) are accurately evaluated using the $\mathcal{K}_\mathcal{N}$ kernel function, while the corresponding $\mu_\mathcal{N}$ function is defined at step 3.2. In steps 3.3–3.5 we select the best matching function found for Π_i and the best similarity value found until now.

We use the relaxed plan π_R in order to define an estimate of the *generation* cost of the current planning problem Π (step 4.1). The $best_C$ value allows to select a good candidate plan for adaptation (which could also be the empty plan). This value is also useful during the computation of the adaptation cost through EVALUATEPLAN, in fact if such a limit is exceeded then it is wasteful to use CPU time and memory to carry out the estimate and the current evaluation could be terminated. The computation of the adaptation cost of the empty plan allows to choose between an *adaptive* approach and a *generative* approach, if no plan gives an adaptation cost smaller than the empty plan.

For all the cases previously selected with a similarity value sufficiently close to $best_S$ (step 4.2) the adaptation cost is determined (step 4.4). If a case of the case base determines an adaptation cost which is lower than $best_C \cdot simil_{\mu_i}(\Pi_i, \Pi)$ then it is selected as the current best case and also the $best_C$ and the `best_plan` are updated (steps 4.5–4.7). Note that we store the encoded plan $\mu_i(\pi_i)$ in `best_plan` since this is the plan that can be used by the adaptation phase for solving the current planning problem Π. Moreover we use the $simil_{\mu_i}(\Pi_i, \Pi)$ value in steps 4.4 – 4.6 as an *indicator* of the effective ability of the selected plan to solve the current planning problem maintaining the original plan structure and at the same time obtaining low distance values.

[7] The α_G coefficient gives more or less importance to plan adaptation vs plan generation; if $\alpha_G > 1$ then it is more likely to perform plan adaptation than plan generation.

[8] In the relaxed planning graph analysis the negative effects of the domain operators are not considered and a solution plan π_R of a relaxed planning problem can be computed in polynomial time (Hoffmann & Nebel 2001).

[9] In our experiments we used $limit = 0.1$.

Plan Adaptation

As previously exposed, the plan adaptation system is a fundamental component of a case-based planner. It consists in reusing and modifying previously generated plans to solve a new problem and overcome the limitation of planning from scratch.

Our work uses the LPG-adapt system given its good performance in many planning domains but other plan adaptation systems could be used as well. It is important to point out that this paper relates to the description of a new efficient case-based planner, no significant changes were made to the plan adaptation component (for a detailed description of it see (Fox *et al.* 2006)).

Experimental Results

In this section, we present an experimental study aimed at testing the effectiveness of OAKPLAN in a number of standard benchmark domains. In the first subsection, we describe the experimental settings and then, in the second subsection, we present the system overall results. Finally, we compare our planner with four state-of-the-art planners.

Experimental Settings

Here we present and discuss the general results for the experimental comparison, moreover we examine the importance of the matching functions and the size of the case base in the overall performance of the system.

OAKPLAN is written in C++ and uses the SQLite3 library for storing and retrieving the data structures of the case base and the VFLIB library (Cordella *et al.* 2004) so as to create and elaborate our graph data structures. The OAKPLAN code, the benchmark planning problems, and a technical report containing all the experimental results are available from the OAKPLAN website `http://pro.unibz.it/staff/iserina/OAKplan/`.

We have conducted all the experimental tests using an AMD Sempron(tm) Processor 3400+ (with an effective 2000 MHz rating) with 1 Gbyte of RAM. Unless otherwise specified, the CPU-time limit for each run is 10 minutes for OAKPLAN and 30 minutes for all the other planners, after which termination is forced. In the following tests the maximum topological distance L considered for the computation of the $k_{\mathcal{N}}$ kernel function in equation (9) is set to half of the number of nodes of the smaller of the two graphs examined $\left(L = \left\lfloor \frac{min(|V_1|,|V_2|)}{2} \right\rfloor\right)$; since this value is sufficiently small to avoid convergence problems the $\gamma(l)$ coefficient of equation (9) is set equal to $\gamma(l) = \left(1 - \frac{1}{L}\right)^l$.

Since our planner and LPG use a randomised search algorithm, the corresponding results are median values over five runs for each problem considered. Moreover, since OAKPLAN and LPG are incremental planners we evaluate their performance with respect to three different main criteria: CPU-time required to compute a valid plan, the plan *stability* (Fox *et al.* 2006) of the generated plans with respect to the corresponding solutions of the target plans and the quality of the best plan generated within the given CPU-time limit.

In these tests the solution plans of the planning cases are obtained by using the *domain dependent* planner TLPLAN (Bacchus & Kabanza 2000) unless otherwise specified. Its use allows us to use a high quality input plan with comparatively low investment of initial computation time. Using a plan from a different planner also ensures that we are not artificially enhancing stability by relying on the way in which the planner explores its search space.

Results for OAKPLAN					
Domain	Solutions	Speed	Matching Time	Quality	Differences
BlocksWorld	187 (86%)	214244.8	121535.6	346.0	49.6
Logistics	213 (98%)	88928.4	69606.3	390.2	76.6
DriverLog	197 (91%)	112556.6	31107.4	230.2	23.6
ZenoTravel	211 (97%)	86722.1	34123.0	194.6	47.2
Rovers	214 (99%)	62421.3	53719.5	374.4	11.4
TPP	210 (97%)	26837.8	859.2	308.8	20.9
TOTAL	1232 (95%)	96162.0	50777.4	307.8	38.2

Table 1: Results of OAKPLAN in the different domains: number of solutions found, average CPU-time of the first solutions (in milliseconds) and corresponding average matching time, average best plan quality and average best plan differences.

Our tests are conducted on a series of variants of problems from different domains:

- BlocksWorld and Logistics Additionals (2nd International Planning Competition),
- DriverLog and ZenoTravel Strips (3rd IPC),
- Rovers-IPC5 and TPP Propositional (5th IPC).

These tests are generally performed by taking six problems from the benchmark test suite and then methodically generating a series of variants for those problems for a total of 216 planning problems for each domain using a procedure similar to the one proposed by Fox et al. (Fox *et al.* 2006). To confirm that the results are not an artifact of the particular problem instances chosen, we adopt a different problem generation strategy for creating problem instances in the Logistics domain. Thus we select problems randomly from the benchmark suites considering the "Additionals" planning problems created in the 2nd IPC for the Domain Dependent planners, distributed across the smallest and the largest problem instances, and generate variant problems for each case.

For each of the benchmark domains we build a case base library used by OAKPLAN. All the problems generated in the different IPCs belong to these libraries. Using the problem generators provided by the IPC organisers, a number of planning problems, with the same features as the IPC planning problems considered, are generated and added to the libraries, for a total of 10000 planning problems for each of the benchmark domains considered except TPP where we only use the original IPC planning problems since it is not possible to use TLPLAN to solve the planning problems of this domain; then we use SGPLAN-IPC5 to determine the solutions of the TPP planning cases.

Overall Results

In this section we report the overall results of OAKPLAN considering the number of solutions found, the CPU-time, the plan quality and the plan stability (Fox *et al.* 2006) of the solutions produced by the adaptation process with respect to the plan obtained by the RETRIEVEPLAN function ($best\_plan$), which is measured considering the *distance*, expressed in terms of number of different actions, between the source plan π and the target plan π_0.

In Table 1 we present the results of OAKPLAN in the different benchmark domains. Here we consider the average CPU-time and the Matching Time for the first solutions generated (in milliseconds). In the fifth column we present the average plan quality of the best solution generated in the different variants and finally the plan distance (in terms of number of differ-

ent actions) of the best solution produced with respect to the plan obtained by the RETRIEVEPLAN function ($best\_plan$). OAKPLAN solves 95% of the problems attempted and the average difference with respect to the target plans is 38.2, i.e. considering all the 1232 planning problems solved by OAKPLAN there are on average 38 actions introduced or removed with respect to the target plans. It requires 96 seconds to solve the different benchmark planning problems of which 51 seconds are required by the matching process. It is important to point out that more than 10000 cases belong to each plan library, which have to be considered by the matching process. We think that such a high number of cases is hardly required by real applications: in fact case base maintenance policies (Smyth & McKenna 1999) could be used in real word applications in order to reduce the number of cases that have to be handled by a case-based planner significantly.

OAKPLAN vs. State of the Art Planners

In this section we analyse the OAKPLAN behaviour with respect to four state-of-the-art planners, showing its effectiveness in different benchmark domains; in particular, we consider METRIC-FF (winner of the 2nd IPC), LPG (winner of the 3rd IPC), DOWNWARD (1st Prize, Suboptimal Propositional Track 4th IPC) and SGPLAN-IPC5 (winner of the 5th IPC).

In Table 2 we report the summary results of OAKPLAN compared to the other planners. In the second columns we report the number of the solutions found by the other planners, in the third columns we report the average speed of the problems solved (in milliseconds), then the average plan qualities produced and finally the average plan differences with respect to the solutions of the set of target plans produced by every single planner. In the brackets we report the *percent errors* with respect to OAKPLAN: we consider only the problems solved by both planners for this comparison, except for the column of the solutions found.

Results for DOWNWARD *and percent errors of* DOWNWARD *vs* OAKPLAN

Domain	Solutions	Speed	Quality	Differences
BlocksWorld	64.0 (-66%)	474335 (+437%)	572 (+155%)	375 (+634%)
Logistics	198 (-7.0%)	93899 (+19%)	353 (-4.8%)	242 (+217%)
DriverLog	76.0 (-61%)	41738 (+381%)	78.8 (+14%)	91.6 (+447%)
ZenoTravel	130 (-38%)	54752 (-27%)	128 (-23%)	85.7 (+72%)
TPP	211 (+0.47%)	148591 (+444%)	293 (-5.1%)	315 (+1403%)
TOTAL	679 (-45%)	133420 (+141%)	281 (+5.9%)	230 (+411%)

Results for LPG *and percent errors of* LPG *vs* OAKPLAN

Domain	Solutions	Speed	Quality	Differences
BlocksWorld	73.0 (-61%)	94078 (+4.3%)	238 (+12%)	149 (+160%)
Logistics	211 (-0.9%)	139416 (+58%)	451 (+16%)	396 (+413%)
DriverLog	122 (-38%)	108708 (+412%)	127 (+6.8%)	199 (+956%)
ZenoTravel	216 (+2.4%)	174570 (+95%)	202 (+2.3%)	332 (+591%)
Rovers	216 (+0.93%)	19440 (-69%)	335 (-11%)	489 (+4201%)
TPP	2.00 (-99%)	496110 (+41415%)	393 (+69%)	468 (+46650%)
TOTAL	840 (-32%)	110054 (+52%)	291 (+3.8%)	354 (+734%)

Results for METRIC-FF *and percent errors of* METRIC-FF *vs* OAKPLAN

Domain	Solutions	Speed	Quality	Differences
Logistics	171 (-20%)	307669 (+429%)	304 (-8.0%)	196 (+163%)
DriverLog	65.0 (-67%)	35598 (+345%)	72.2 (+16%)	124 (+655%)
ZenoTravel	164 (-22%)	219264 (+200%)	123 (-29%)	97.0 (+98%)
Rovers	198 (-7.5%)	745702 (+1089%)	299 (-19%)	391 (+3366%)
TPP	77.0 (-63%)	899049 (+7054%)	246 (-3.9%)	240 (+1202%)
TOTAL	675 (-45%)	455941 (+757%)	229 (-15%)	227 (+500%)

Results for SGPLAN-IPC5 *and percent errors of* SGPLAN-IPC5 *vs* OAKPLAN

Domain	Solutions	Speed	Quality	Differences
Logistics	216 (+1.4%)	462093 (+404%)	414 (+4.6%)	268 (+246%)
DriverLog	106 (-46%)	321346 (+2538%)	119 (+31%)	190 (+971%)
ZenoTravel	180 (-15%)	171353 (+126%)	142 (-23%)	137 (+175%)
Rovers	216 (+0.93%)	163457 (+162%)	343 (-8.4%)	467 (+4011%)
TPP	211 (+0.47%)	942278 (+3414%)	314 (+1.6%)	354 (+1593%)
TOTAL	929 (-25%)	429328 (+644%)	288 (-2.4%)	300 (+710%)

Table 2: Summary Tables of the different planners examined and a comparison with respect to the corresponding results produced by OAKPLAN.

Considering the total values we can see that OAKPLAN can solve the greatest number of variants, followed by SGPLAN-IPC5 and LPG. Regarding the CPU-time, we remark that DOWNWARD, LPG and OAKPLAN present similar computation time, while the CPU-time is more significant in METRIC-FF and SGPLAN-IPC5. However these average values are computed only by considering the problems solved by every single planner. In this case the SGPLAN-IPC5 planner solves 211 variants in the TPP domain requiring 942 seconds for them, whereas these variants only marginally influence the results of LPG. Regarding the difference values we can see that OAKPLAN clearly produces better results than the other planners. With respect to the plan quality we can note that METRIC-FF gives better results whereas OAKPLAN produces the worst results. We would like to point out that in OAKPLAN the optimisation process tries to balance between good quality and low distance values since we are much more interested in generating a plan with a limited number of differences with respect to the target plan than producing solutions of good quality. Moreover OAKPLAN is able to solve much more difficult planning problems than the other planners and these solutions weigh significantly on the average plan quality produced. In the following we examine the behaviour of each planner vs. OAKPLAN.

Downward cannot solve any problem in the Rovers domain. Globally it can solve 679 problems in comparison with the 1232 solved by OAKPLAN. DOWNWARD is 141% slower than OAKPLAN while their plan quality is comparable. The distance values of the plans generated by DOWNWARD with respect to the solutions produced by the same planner on the problems used to generate the variants is 411% greater than OAKPLAN. This high value is not particularly surprising since DOWNWARD and the other planners do not know the target plans used for this comparison. Moreover the search processes and the solution plans produced by a planner could be significantly different also for two planning instances that only differ in a single initial fact. These distance values are interesting since they are a clear indicator of the good behaviour of OAKPLAN and show that the generative approach is not feasible when we want to preserve the stability of the plans produced.

LPG can solve 840 of the 1296 variants, requiring 32% CPU-time more than OAKPLAN and the average distance of the solutions on target planning problems is 354 actions (which corresponds to $+734\%$ with respect to OAKPLAN). It is interesting to remark that the CPU-time needed by LPG to solve the Rovers variants (19.4 seconds) is significantly lower than in OAKPLAN (62.4 seconds) due to the additional CPU-time required by the matching process of OAKPLAN (53.7 seconds). The distance of the plans generated by LPG in this domain is 4201% greater than OAKPLAN.

Metric-FF cannot solve any variant in the BlocksWorld domain. Globally it can solve 675 problems and is 757% slower than OAKPLAN while its plan quality is 15% better. Finally the distance of the plans generated by METRIC-FF with re-

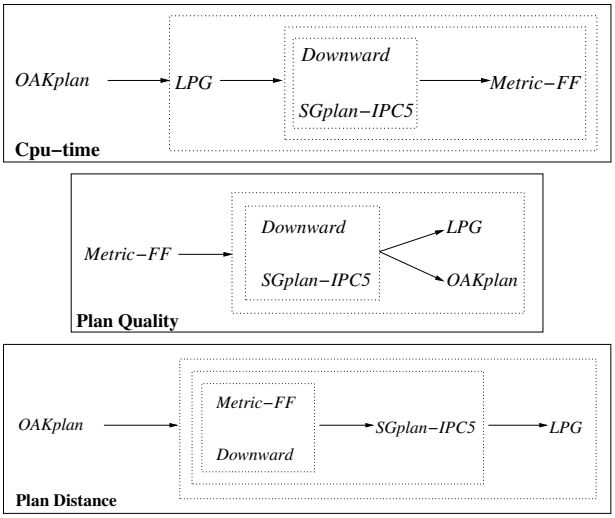


Figure 5: Partial order of the performance of OAKPLAN, DOWNWARD, LPG, METRIC-FF and SGPLAN-IPC5 according to the Wilcoxon signed rank test for our benchmark problems.

spect to the solutions produced by the same planner on the target problems is 500% greater than OAKPLAN.

SGPlan-ipc5 planner can solve 929 problems and is 644% slower than OAKPLAN, the plan qualities are very similar and considering the distance of the plans generated by SGPLAN-IPC5 are on average 710% greater than with OAKPLAN.

Figure 5 gives a graphical summary of the Wilcoxon results for the relative performance of OAKPLAN with DOWNWARD, LPG, METRIC-FF and SGPLAN-IPC5 in terms of CPU-time, plan quality and difference values for our benchmark problems. Here we can observe that OAKPLAN is statistically more efficient values than all the other planners in terms of CPU-time and plan distance. On the contrary OAKPLAN and LPG produce statistically worse plans from the quality point of view than the other planners, while METRIC-FF produces the highest quality plans.

Globally we can note that OAKPLAN is able to solve many more problems than the other planners and the first solution is usually generated in less time. In addition the distance values are significantly lower with respect to the target plans although the quality of the plans produced is slightly worse than that of the plans produced by the other planner; this is also related to the optimisation performed by OAKPLAN where we try to minimise not only the plan metric function but also the distance with respect to the solution plan of the planning case selected.

Conclusions

In this paper we have described a novel case-based planning system, called OAKPLAN, which uses ideas from different research areas showing excellent performance in many standard planning benchmark domains. To the best of our knowledge this is the first case-based planner that performs an efficient domain independent objects matching evaluation on plan libraries with thousands of cases.

We have examined OAKPLAN in comparison with four state of the art plan generation systems showing its extremely good performance in terms of the number of problems solved, CPU time, plan difference values and plan quality. Results are very encouraging and show that the case-based planning approach can be an effective alternative to plan generation when "sufficiently similar" reuse candidates can be chosen. This happens to different practical applications especially when the "world is regular" and the types of problems the agents encounter tend to recur. Moreover this kind of approach could be extremely appealing in situations in which the "stability" of the plan produced is fundamental. This is the case, for example, in mission critical applications where end users do not accept newly generated plans and prefer to use known plans that have already been successful in analogous situations and can be easily validated.

We believe that even more significant results will come from combining our approach with ideas and methods that have been developed in planning, case-based reasoning, graph theory and supervised learning research areas.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Chien, Y. P.; Hudli, A.; and Palakal, M. 1998. Using many-sorted logic in the object-oriented data model for fast robot task planning. *Journal of Intelligent and Robotic Systems* 23(1):1–25.

Cordella, L. P.; Foggia, P.; Sansone, C.; and Vento, M. 2004. A (sub)graph isomorphism algorithm for matching large graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26(10):1367–1372.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of International Conference on AI Planning and Scheduling (ICAPS)*. AAAI Press.

Fröhlich, H.; Wegner, J. K.; Sieker, F.; and Zell, A. 2005. Optimal assignment kernels for attributed molecular graphs. In De Raedt, L., and Wrobel, S., eds., *ICML*, volume 119 of *ACM International Conference Proceeding Series*, 225–232. ACM.

Fröhlich, H.; Wegner, J. K.; Sieker, F.; and Zell, A. 2006. Kernel Functions for Attributed Molecular Graphs – A New Similarity Based Approach To ADME Prediction in Classification and Regression. *QSAR Comb. Sci.* 25:317–326.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)* 20:pp. 239–290.

Haussler, D. 1999. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, UC Santa Cruz.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 14:253–302.

Johnson, M. 1985. *Relating metrics, lines and variables defined on graphs to problems in medicinal chemistry*. New York, NY, USA: John Wiley & Sons, Inc.

Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly* 2:83–97.

Leake, D. B., ed. 1996. *Case-Based Reasoning*. Cambridge, Massachusetts: The MIT Press.

Lin, D. 1998. An information-theoretic definition of similarity. In Shavlik, J. W., ed., *ICML*, 296–304. Morgan Kaufmann.

Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: A complexity-theoretic perspective. *Artificial Intelligence- Special Issue on Planning and Scheduling* 76:427–454.

Scholkopf, B., and Smola, A. J. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press.

Smyth, B., and McKenna, E. 1999. Footprint-based retrieval. In Althoff, K. D.; Bergmann, R.; and Branting, K., eds., *ICCBR*, volume 1650 of *Lecture Notes in Computer Science*, 343–357. Springer.