A Case-Based Reasoning approach to knowledge transfer in Tabu Search

Stephan Grolimund Jean-Gabriel Ganascia

University of Paris IV
Laforia - IBP
4, place Jussieu
F-75232 PARIS Cedex 5

email: {ganascia, grolimund}@laforia.ibp.fr

Abstract

Tabu search, a class of local search techniques, relies explicitly on knowledge transfer between search episodes in order to improve performances as search progresses. For this purpose, relatively simple, statistical learning techniques are used. On a problem class close to combinatorial optimisation, i.e. plan generation, analytical learning techniques like EBL or derivational analogy have shown to perform and scale up well.

In order to make tabu search-like techniques benefit from analytical learning, we propose a case-based reasoning approach for learning and reusing control knowledge for combinatorial optimisation problems. A move decomposition technique is used to reduce neighbourhood sizes without the need to call on candidate sets. It permits to cope with computational complexity due to learning, and to allow problem modelling with complex moves. This second point is especially interesting concerning diversification, a control strategy of tabu search, as well as concerning real life problems.

Résumé

Les méthodes de recherche tabou, une sous-classe des méthodes de recherche locales, font explicitement usage d'un transfert de connaissance entre des étapes successives de la recherche. Pour ce faire, elles se fondent sur des techniques d'apprentissage relativement simples et statistiques. Concernant une classe de problèmes proche de l'optimisation combinatoire, à savoir la génération de plans, les techniques d'apprentissage analytique comme l'apprentissage par explication se sont montrées performantes même sur des problèmes de grande taille.

Nous nous proposons de faire bénéficier les méthodes de recherche tabou des résultats obtenus en apprentissage analytique. Cela nous conduit à introduire une technique de raisonnement à partir de cas pour apprendre des connaissance de contrôle lors de la résolution de problèmes d'optimisation combinatoire. Une technique de décomposition de transitions est proposée afin de réduire la taille des voisinages sans avoir à recourir aux ensembles candidats classiques. Il est alors possible de maintenir notre méthode à un niveau de complexité acceptable. Tout comme, il devient possible d'utiliser des transitions plus complexes lors de la modélisation de problèmes. Cela est particulièrement intéressant lors de la spécification de la phase de diversification de la méthode tabou ainsi que pour la modélisation de problèmes d'optimisation industriels.

Contents

1. Introduction

Analytical learning techniques allow a system to improve performance according to its past experiences. Most of them apply to plan generation problems, where a plan generator has to produce an operator sequence, leading from a given initial state to a final state satisfying a set of initially given goals. The resulting operator sequence is called *plan* and forms the solution of the planning problem. In case the plan generator has a learning component associated, this one is responsible for capturing the plan's important features to allow the plan generator to reuse them in subsequent planning problems. Examples of such techniques are EBL [Minton, 1989], derivational analogy [Veloso, 1992], and chunking in SOAR [Laird et al, 1986]).

Combinatorial optimisation problems have different nature than planning problems. In general, a quality measure in form of an objective function is associated with each state belonging to the solution space. The optimisation task consists in finding the state for which this function is minimised (or maximised) over the whole search space. In this context, the desired final state has to satisfy a property (i.e. optimality) with respect to every state of the search space, whereas in plan generation, any state satisfying the goals can be a solution state. Nevertheless, plan generation could be formulated as optimisation problem, stating the task such as "finding the plan with minimal length". However, finding such an optimal plan is beyond scope for classical planning systems [Georgeff 1987], and only some specialised analytical learning techniques are concerned with optimisation problems [Tambe & Rosenbloom, 1993] [Ruby & Kibler, 1992].

Combinatorial optimisation is well-known in operations research. There exists a great number of techniques, exact and heuristic ones, dealing with it [Papadimitriou & Steiglitz, 1982]. One important class of methods, called *local search*, explores the search space by performing iteratively transformations on solution states in order to explore as complete as possible the search space[1]. Recently, two variations of the classical local search scheme have shown to be successful: simulated annealing [Kirkpatrick et al, 1983] and tabu search [Glover, 1989, 1990]. Whereas simulated annealing relies on an analogy with physics, tabu search explicitly incorporates a transfer of knowledge from past search episodes to future search.

Although in its function similar to analytical learning in planning, knowledge transfer in tabu search acquires control knowledge by primarily relying on simple statistical properties of the solution space. It does not use symbolic learning techniques that rightly showed to have significant impact in the resolution of planning problems.

In this paper, we present a domain independent analytical learning approach to knowledge transfer in combinatorial optimisation problems, using a case-based reasoning technique for extracting and storing control knowledge during past search, and for reusing it during conflict resolution, as search goes on. In order to improve efficiency of this, we introduce move decomposition, a technique that slices local search moves into their basic components. By this

---

[1] These methods initially influenced the first generation AI planning systems, based on state space search [Tate et al, 1990].

means, the move's neighbourhood sizes are reduced and learning can be focused on the move's basic components instead of moves as a whole. Similar to tabu search, and in opposite to analytical learning in general, our approach does not intend to transfer knowledge between different problem instances, but yields to reuse control knowledge during optimisation of the same instance the knowledge has been extracted from.

First, we give a brief introduction to local search and tabu search. Second, we describe our move decomposition technique, valid for local search in general. Third, our case-based reasoning technique for knowledge transfer is outlined.

2. Local Search

Local search optimisation techniques group a variety of methods for a large panorama of different problems. We give here only a brief introduction to local search for combinatorial optimisation problems as well as to tabu search, pointing out the aspects which are of concern for our work. On tabu search, the interested reader finds a covering introduction together with representative research papers in [Glover et al, 1993a].

2.1. The Local Search framework

For an instance of a combinatorial optimisation problem, local (or neighbourhood) search methods explore the *solution space $\mathcal{S}$*, which contains the feasible problem solutions, by repeatedly making *moves* m, leading from one *solution* S to another solution S' = m(S). The aim of this exploration is to identify good solutions (optimal or near-optimal ones) according to the minimisation[2] of an *objective function* f(S). For every state S, its *neighbourhood* N(S) is defined as containing all solutions S' reachable by the moves applying to S. For improving maximally the objective function of the obtained solution after each iteration, the method selects as successor in each step the best element belonging to N(S).

Such iteratively improvement methods usually get trapped in *local minima*, i.e. during search, they will reach a solution S^{LM} whose neighbourhood $N(S^{LM})$ contains no successor that will improve the objective function :

$$\forall S \in N(S^{LM}), f(S) \geq f(S^{LM})$$

As there is no general way to identify such solutions as being the globally optimal ones, the search goes on, starting from a new initial solution. There are different ways to determine this new solution state: either a new starting point is built from scratch, randomly or systematically, or search continues from the local minima by performing one or several *decreasing moves*, i.e. moves that decrease solution quality.

In practical applications, the evaluation of a state's neighbourhood meets computational problems: neighbourhoods are often too large, so it is too complex to evaluate the objective

---

[2] For simplicity, the presentation is restricted to minimisation problems, but remains valid for maximisation as well.

function for each of its elements during every iteration. Even worse, it is possible that the evaluation of the objective function itself is very time consuming [Crainic et al, 1993].

These two factors together often lead to a computational infeasible neighbourhood evaluation. To remedy, several restrictions concerning problem modelling are mainly used:

a) Restrict the neighbourhood size by using only moves of relatively small complexity.
b) Limit the investigation of the neighbourhood to a *candidate set* V(S), $V(S) \subseteq N(S)$. While the definition of V(S) is problem dependent, its size can be as small as $|V(S)| = 1$.
c) Limit the computational complexity of the objective function if necessary. This can be achieved by either constructing special, problem-dependent state evaluation schema, or, for problem domains where this is not available, by using some fast estimations.

According to these restrictions, neighbourhoods used in practical, large scale application are frequently have complexities of about $O(n^2)$, n representing the problem size (Section 2.3).

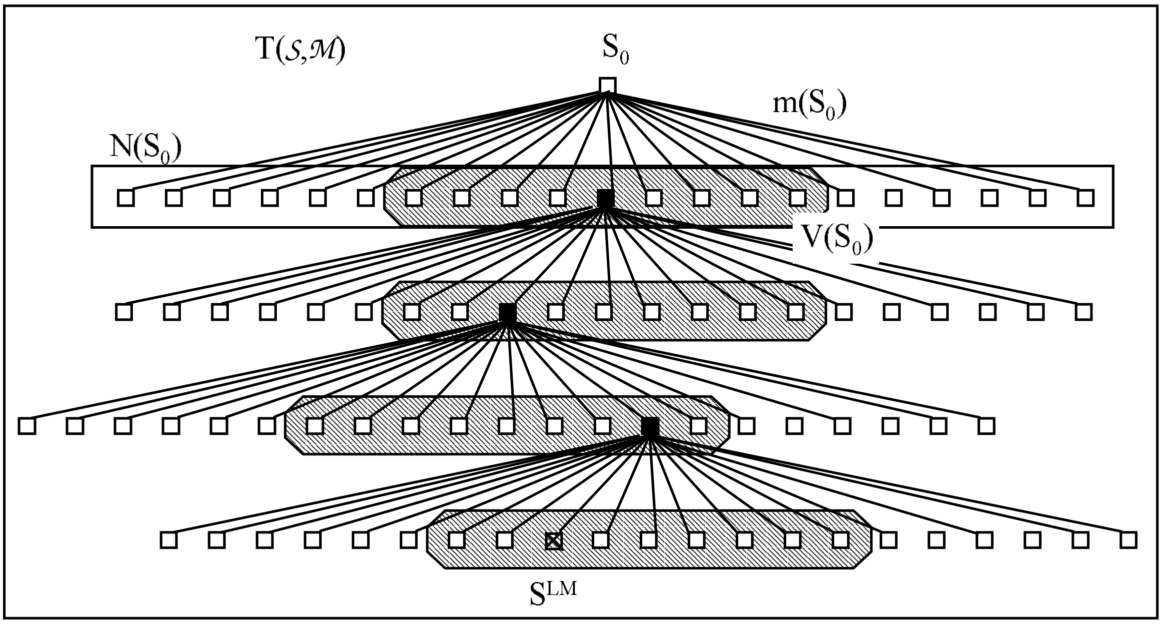

Figure 1 : search tree for $G(\mathcal{S},\mathcal{M})$

The search space can be represented by a directed graph $G(\mathcal{S}, \mathcal{M})$, where the solution states represent the nodes, and the arcs belong to the set $\mathcal{M}$ covering all possible moves m. Commonly, $G(\mathcal{S}, \mathcal{M})$ has the property of being strongly connected. Exploring $G(\mathcal{S}, \mathcal{M})$ by starting in solution S_O and applying moves iteratively can be seen as walking through a search tree $T(\mathcal{S}, \mathcal{M})$ with root S_O, as illustrated in figure 1. For each node $S \in T(\mathcal{S}, \mathcal{M})$, the arcs correspond to the applicable moves and the white boxes represent the elements of N(S). An exemplar candidate set V(S) is grey underlaid. $T(\mathcal{S}, \mathcal{M})$ is explored following a strategy called select and commit, which does not allow to backtrack on a move already applied. The effectively visited successor of a node, after applying a move, is marked with a black box. As defined above, each node of this tree refers to a possible solution $S \in \mathcal{S}$ of the optimisation problem.

2.2. Tabu Search

Tabu-search is a high-level control heuristic guiding the search space exploration of local optimisation techniques. It copes with the problem of local minima by allowing the use of decreasing moves during search. As this ability introduces the possibility of cycling, i.e. the search may reach solutions which have just been visited before, the performed moves are inscribed in a *tabu-list* and will be forbidden to apply for a fixed number of iterations, in the hope that meanwhile the search left the attraction region of the local minimum.

But tabu-search has a second, perhaps more important aim than just treating local minima. The idea is that it is possible, for one and the same problem instance, to guide the search to good solution by "acquiring, storing and processing knowledge" gained during the past search[3]. This assumption leads to two concurrent search control strategies, *intensification* and *diversification*, which both modify the standard search space exploration with regard to solution states visited before. Intensification aims to focus the search on promising areas of the search space, while diversification tries to direct the search to yet unexplored regions.

Tabu search is a flexible framework allowing to adapt the technique to a wide variety of different optimisation problems. For illustration purposes, we list some commonly made design choices in actual tabu search applications [Glover et al, 1993b]

Search control in tabu search is organised following a three level architecture. The lower level uses one or several tabu-lists of fixed or variable length, working like first-in-first-out queues, in which moves are inscribed that are not authorised to apply during the current iteration. The reasons why moves are selected to be inscribed in the list is problem-dependent, but, in order to reduce the probability of cycling, the choice is based on the recently applied moves as well as their reverse moves.

The control architecture's middle level, intensification, aims to identify common properties of good solutions visited so far. Often in form of variables fixed to a unique value within several solutions, these properties are used to focus the search to solutions respecting them. A way to do this is to fix some variables and to keep them fixed during some iterations by eliminating all moves that would affect them. This has the valuable side-effect of reducing the neighbourhoods.

The upper level of control, diversification, is complementary to intensification, in that the search is guided to solutions with properties that were rare during past search. Several techniques are used to accomplish this. Either some variables are fixed to their complements, either rarely applied moves are favoured, or even particular classes of moves are used in special situations only. These moves are often too complex to be applied at every step of the search, so they are used only during diversification.

Like local search in general, tabu search's performance in particular depends crucially on the design of appropriated moves. For complex moves, there is a general trade-off like: the more a

---

[3] Tambe and Rosenbloom (1993) implicitly make a similar assumption: for their optimisation problem, they suppose that past search episodes contain knowledge which can be reused for identifying good solutions to close problem instances.

move is complex, its application modifies the current solution and pushes search further, but the larger are the neighbourhoods it produces. As illustrated for diversification, a possible compromise is to use complex moves only in special situations.

2.3. Examples on Tabu Search

For illustration purposes, three typical applications of tabu search are skilled out below. Each one is characterised by its search space definition and the different control strategies used.

2.3.1. Single machine production scheduling problem [Barnes & Laguna, 1993]

Problem: n jobs have to be placed on a single machine. All jobs arrive at time 0. Each job i requires t_i units of time for production, and receives a penalty of p_i for each time unit it is scheduled after arrival time. There is a set-up cost s_{ij} for scheduling job i after job j. The objective is to minimise the sum over penalties and set-up-costs. This problem is a "permutation problem", where the desired solution is one of n! possible sequences.

Search space: two types of moves are defined: `swap` moves exchange two jobs i and j, and `insert` moves move one job to another position. Each type of move defines a neighbourhood size of $O(n^2)$. As neighbourhoods are too large to be evaluated at each iteration, a candidate set is introduced.

Local control: one tabu-list of fixed size is used. Jobs affected by `insert` moves are not allowed to change position as long as their move is inscribed in the tabu-list, and the earlier of two swapped jobs cannot be placed before its origin position as long as the `swap` move is in the tabu-list.

Diversification: on the leave of a local minimum, frequently executed moves are penalised in order to proliferate the use of rarely applied ones.

2.3.2. Quadratic assignment problem [Taillard, 1991]

Problem: in order to arrange a factory, n items (workshops) have to be assigned to n different locations, with a_{ij} the distance between location i and location j, b_{pq} the flow between item p and item q, and x_{ip} a boolean variable designing that item p is placed at location i. The objective is to place the items in a way as to minimise the sum of the products between flows and distances :

$$\min\left(\sum_i \sum_j \sum_p \sum_q a_{ij} \cdot b_{pq} \cdot x_{ip} \cdot x_{jq}\right)$$

Search space: `swap` moves are defined to exchange items p and q. The cost of the exploration of the entire neighbourhood is of complexity $O(n^2)$, which, in this case, allows to explore the neighbourhood completely.

Local control: it can be considered as using n^2 tabu-lists, one for each item and each location. Each list has a fixed length of 1 and contains the number of the latest iteration at which the item was at the corresponding location. A move will be tabu, if both items to be swapped ought to be placed in positions they occupied within the last t iterations. The parameter t is changed randomly during the search.

Diversification: select a move if it will place both items in locations that they did not occupy for a fixed number of iterations.

2.3.3. Location/allocation transportation problem [Crainic et al, 1993]

Problem: there are N warehouses which deliver P products to M customers where D_{ip} designs the demand of product p by customer i and the variable x_{ijp} designs the flow of product p from warehouse j to client i. The objective is to minimise the sum over transportation costs and costs affecting the opening and operating of the warehouses.

Search space: the boolean variable y_j describes whether a warehouse j is open or closed. For y_j fixed, $\forall j$, and assuming linear cost functions, this problem is a uncapacitated multicommodity minimum cost network flow problem. It can be solved efficiently using special network algorithms. By this means, the remaining problem is to determine the warehouse configurations by fixing the boolean variables y_j. For each configuration, the optimal network will then be produced by the network algorithm, affecting clients to open warehouses and deciding on warehouse sizes.

Three types of moves are defined: `add` moves open a warehouse, `drop` moves close a warehouse, and `swap` moves close one warehouse and open another one. As neighbourhoods are too large, several restrictions on them are defined in order to get interesting candidate sets.

Local control: two tabu-lists of fixed size, one for `add/drop` moves, and one for `swap` moves, are used to prevent cycling.

Intensification: the exclusive use of strictly improving `swap` moves is forced after an interesting sequence of `add/drop` moves, in order to explore several warehouse configurations without changing the number of opened warehouses.

Diversification: after a fixed number of intensification steps, a certain number of warehouses are set to the complement values (opened or closed) they had according to the best solution found yet. This operation can be seen as an application of several `add` and `drop` moves together. These are inscribed in a special tabu-list, which concerns only the diversification phases.

3. A structured search space exploration

In practical applications, local search techniques, and especially tabu search, often suffer from computational problems due to the complexity of member evaluation in potentially large neighbourhoods. This section proposes a generally valid method that reduces neighbourhood sizes significantly. As direct consequences, the method allows to give up the somewhat unsatisfying notion of candidate set. It permits as well a more accurate modelling of problem domains that require complex transformations between solution states.

3.1. Move decomposition

As outlined, in classical local search all states of the search space $\mathcal{S}$ are feasible problem solutions, and moves describe complete transformation from one solution to another one. But often, moves do not describe elementary transitions, but quite complex solution state transformations. The decomposition of a move into its basic components reduces neighbourhood size, as these directly depend on the complexity of the applicable moves. To make such a decomposition possible, it is necessary to allow *infeasible states* to be part of the search space.

Consider the single machine production scheduling problem. The `insert` move has two basic components. First, a job is retracted from the schedule, and second, it is put back to another position, again. This leads to the decomposition of `insert` into two basic actions, `retract` a job from the schedule, followed by `place` that same job at another position. The intermediate state between `retract` and `place` is infeasible, as after `retract`, one job will not be scheduled anymore. The neighbourhood size of `insert` is $O(n^2)$, as n jobs can be inserted at n positions. The simpler moves `retract` and `place` generate both a neighbourhood of O(n), for n jobs the are retractable, and n positions where the retracted job can be placed again.

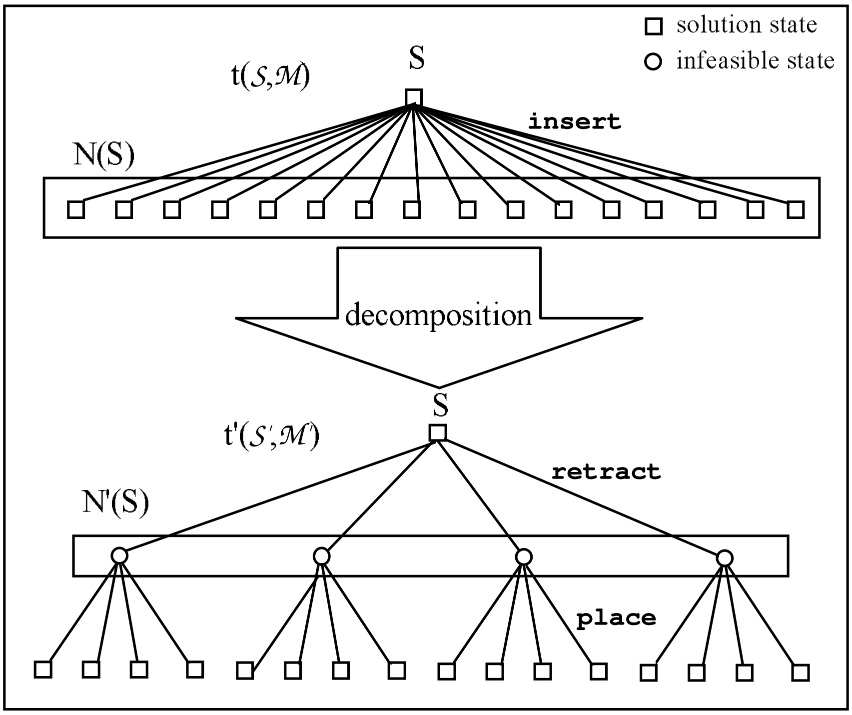

Figure 2 : move decomposition in the single machine production scheduling domain

Figure 2 illustrates move decomposition, with $t(\mathcal{S}, \mathcal{M})$, the subtree of $T(\mathcal{S}, \mathcal{M})$ corresponding to the states and moves leading from a state S to the first feasible solution succeeding to S. The deepness of $t(\mathcal{S}, \mathcal{M})$ is 1, and the degree of S, $d(S) = |N(S)|$, is of size $O(n^2)$. Decomposing `insert` into `retract` and `place` results in a search tree $t'(\mathcal{S}', \mathcal{M}')$ with deepness = 2, whereas the degree of S' and that of the members of N(S') is O(n). The decomposition does not affect the number of feasible solutions accessible from the root, both search trees have the same number of leaves. The total number of investigated neighbourhood members on a solution path changes from $O(n^2)$ to $2 \cdot O(n)$.

3.2. Search space formalism

3.2.1. State space

The *search space* $\mathcal{E}$ is composed of states $E \in \mathcal{E}$. It contains the subspace $\mathcal{S}$ of *feasible states*, $\mathcal{S} \subseteq \mathcal{E}$. The membership of E in $\mathcal{S}$ can be decided using the *characteristic function* $f_{\mathcal{S}}$ from $\mathcal{E}$ to $\{0, 1\}$, such that

$$f_{\mathcal{S}}(E) = \begin{cases} 1, \text{iff } x \in \mathcal{S} \\ 0, \text{iff } x \notin \mathcal{S} \end{cases}$$

For each state $E \in \mathcal{E}$, the *objective function* $q\colon \mathcal{E} \rightarrow \Re^+$ associates a quality to E. The space of the *optimal states* $\mathcal{E}^*$, $\mathcal{E}^* \subseteq \mathcal{S}$, $|\mathcal{E}^*| \geq 1$, a subspace of $\mathcal{S}$, holds all feasible states which minimise the objective function globally, such that :

$$\forall E_1 \in \mathcal{E}^*, E_2 \in \mathcal{S}, q(E_1) \leq q(E_2)$$

In general, no characteristic function concerning $\mathcal{E}^*$ is known.

3.2.2. Domain theory

The *domain theory* $\mathcal{O}$ holds the set of *operators* $o : \mathcal{E} \rightarrow \mathcal{E}$ that define transitions between states. Together with an initial state, the domain theory defines implicitly the graph $G(\mathcal{E},\mathcal{O})$ for every problem instance (formerly $G(\mathcal{S},\mathcal{M})$). Operators are instances of an operator schema [Fikes & Nilsson, 1971]. An *Operator schema* os describes a class of operators. It contains variables that will be replaced by constant expressions in order to produce a specific, instanciated operator.

For example, `insert(?X, ?P)` is an operator schema with variables `X` and `P`, while `insert(job_1,position_17)` is an instance of the operator schema. The *substitution* $\sigma = \{{}^{\text{constant}}/\text{variable}\}$ stands for the replacement of a variable by a constant expression. The substitution used in the example is $\sigma = \{{}^{\texttt{job\_1}}/\texttt{?X},\ {}^{\texttt{position\_17}}/\texttt{?P}\}$.

A *transformation* τ is a finite, totally ordered sequence of operators o^τ leading from one feasible state E_0^τ to another feasible state E_f^τ, $E_0^\tau, E_f^\tau \in \mathcal{S}$, passing by 0, 1 or several intermediate, infeasible states E_i^τ, $E_i^\tau \notin \mathcal{S}$ (Figure 3). E_0^τ is called *initial state* of τ, and E_f^τ is called its *final state*. Transformations play the role of moves in local search, whereas operators are similar to components of decomposed moves. When there is no ambiguity, the suffix τ will be omitted for state and operator denomination.

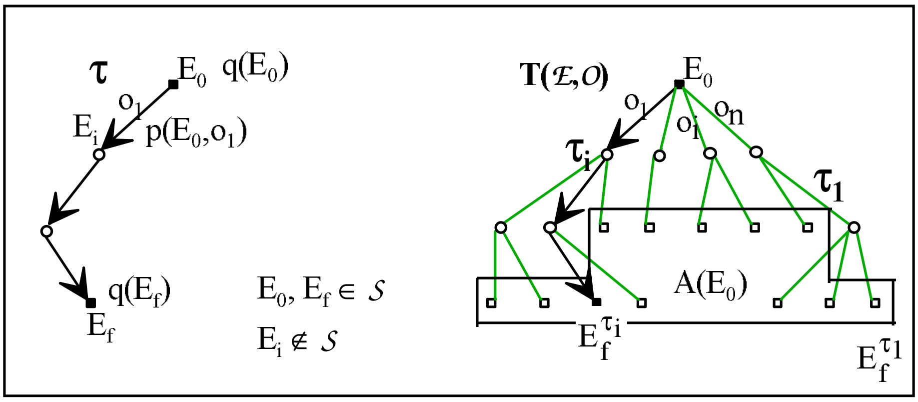

Figure 3 : transformation τ, and search tree T(ℰ,𝒪)

For a state E_0, several different transformations τ_i may lead to different final states $E_f^{\tau_i}$. Called *accessible states*, these final states can be seen as being the leaves of the search tree T(ℰ,𝒪) with root E_0. They form the set of accessible states $A(E_0)$ of E_0. An operator typically belongs to several transformations, as the state it leads to may have different applicable operators leading to different final states.

3.3. Search space exploration and performance considerations

3.3.1. State transition diagram

Operator schemata are linked between each other by some sort of functional relations, inherited from their status as decomposed moves. These relations define a total order on the operator schemata. A suitable representation for the relations are standard state transition diagrams. As example, figure 4 shows the state transition diagram for the single machine production scheduling domain.

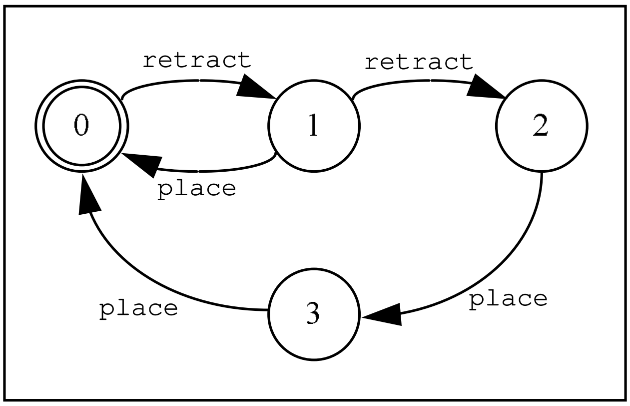

Figure 4: State transition diagram for the single machine scheduling domain

In `state_0` search is in a feasible problem instance state. Applying `retract` to `state_0` moves the search to an infeasible state, `state_1`, where either `place` or `retract` can be applied. Applying `place` accomplishes the transformation, guiding search back to the feasible state, `state_0`. This transformation is similar to an `insert` move. Applying a second `retract` to `state_1`, followed by two `place` performs a transition similar to previous `swap` moves.

As transformations are defined to start and finish in feasible states and are of finite length, they necessarily represent a finite circuit passing through `state_0`. This property assures that every leave of the search tree $T(\mathcal{O},\mathcal{M})$ is a feasible state, and the deepness of $T(\mathcal{O},\mathcal{M})$ is finite.

3.3.2. Operator evaluation and estimation

The choice between applicable operators o^{τ_i} in a state E should be based on the objective function evaluation of τ_i's final state, $q(E_f^{\tau_i})$. This would correspond to move evaluation in local search. However, $q(E_f^{\tau_i})$ is in general unknown without exploring τ_i completely. But doing this for every transformation leads to an enumerative exploration of A(E), a too complex operation because it would cause the lost of all benefits from the reduced neighbourhood. However, it is not possible to judge operators on the quality of the intermediate states they would lead to, because these states are infeasible.

This problem of complex objective functions is met in some practical applications of local search techniques [Crainic et al, 1993]. It leads to the introduction of a heuristic estimation of the objective function value of a transformations final state.

So, a function for *quality estimation* $p: \mathcal{E} \times \mathcal{O} \rightarrow \Re$, associated to each state E and each operator o^{τ_i}, is used to forecast the change of the objective function between the actual state E (i.e. q(E), known) and the *best* final state of all the transformation τ_i to which belongs o^{τ_i} (i.e. $\overline{q}(E_f^{\tau_i})$, unknown).

$$p(E, o^{\tau_i}) = q(E) - \overline{q}(E_f^{\tau_i})$$

Of course, the better these estimations, the better the choice of the operator to apply, and the better the investigation of a state's accessible states. However, there are limits on the complexity of the estimation function, as the complexity of neighbourhood evaluation depends on both neighbourhood size and computational complexity of the estimation function for each neighbourhood member.

3.3.3. Search space exploration

Similar to local search, the search space is explored following a select and commit strategy without backtracking, in the aim to push search further with each step. Starting in an initially feasible state, operators are applied until the next feasible state is reached, accomplishing a transformation. The properties of the state transition diagram guarantee that there will be no failure branch in the search tree, which ever operator will be applied. Search will go on, starting a new transformation, until some stopping condition is met. Possible stopping conditions are solution quality, number of iterations or computation time. Once the search stopped, the best feasible state found is considered as solution to the problem instance.

In each state E, there are usually different operators applicable. For that an operator is applicable, it has to belong to an operator schema authorised to act on E, according to the state transition diagram. The choice of the operator to apply is based on each operator's quality estimation p(E,o). Between the whole set of applicable operators, the one with maximal p is selected. Like in tabu search, decreasing moves are allowed.

3.3.4. Performance considerations

Move decomposition replaces a single step, high complex neighbourhood evaluation, found in classical local search, by a sequence of less complex ones. For large problem instances, this should produce a remarkably gain, allowing the introduction of a symbolic learning technique for knowledge transfer like the one outlined in Section 5.

There is another point worth to be skilled out. This formalism can be used to construct transformations that are more complex than those based on the relatively simple moves used in local search. The only conditions that transformations have to respect in order to maintain computational feasible neighbourhood evaluations are a) the transformation decomposition must produce operators that are simple enough, and b) quality estimation p of these operators has to be of low complexity.

Consider the single machine production scheduling domain. A swap move involving 3 jobs, `swap3`, would produce a neighbourhood of complexity $O(n^3)$. However, `swap3` can be decomposed into three `retract` operators followed by three `place`, each one of complexity $O(n)$. Whereas `swap3` is not interesting for large problem instances because of its complexity, replacing it by a sequence of six low complex operators could show profitable on the overall system performance.

3.4. Transformation decomposition examples

3.4.1. Single machine production scheduling problem

As already introduced, the moves `swap` and `insert` can be decomposed into the operators `retract` and `place` where `retract` retracts one job from the schedule, and `place` inserts this job again.

For performing `insert`, one `retract` and one `place` operator have to be applied. `Insert` has a neighbourhood of $O(n^2)$, while each of the two operators has a neighbourhood of $O(n)$, and their combination remains of linear complexity $O(n)$. The `swap` move can be reproduced by performing two `retract` operators followed by two `place` operators. The resulting complexity is still of $O(n)$.

3.4.2. Quadratic assignment problem

It is possible to use the operators `retract` and `place` described above to substitute the swap moves.

3.4.3. Location/allocation transportation problem

The `add` and `drop` moves can directly be used as `add` and `drop` operators, and there will be no more need for defining `swap` moves (which, in the cited example, are already a combination of `add` and `drop` moves).

This problem domain in particular has another interest. The presented model relies on an efficient resolution of a underlying network flow problem, supposing a network with uncapacitated arcs and nodes together with linear cost functions. If these assumptions do not hold anymore, the network flow problem cannot be solved efficiently and the presented model looses its force.

4. What can be learned during the search?

The overall goal of the search is to identify globally optimal or near-optimal solutions for a given problem instance. For rule-based systems in AI, this goal is somewhat unusual, as in optimisation the desired final state is required to satisfy a constraint (i.e. optimality) relative to *every* element of the search space, whereas usually, a solution state has "only" to respect a set of goal literals, so that *any* state doing so can be acceptable as solution.

Problem solving has to cope with execution problems like goal interferences and success or failure on branches of the search tree. So, analytical learning techniques[4] focus on these phenomena and acquire domain knowledge and especially control knowledge in order to cope with them [Minton, 1993].

However, consider the search space $\mathcal{E}$ introduced in the previous section. Its exploration meets only success paths in the search tree, as search is guided following a finite, interaction free circuit described in the state transition diagram. Neither an impasse branch nor goal-interference exist. So, learning on failures is irrelevant. Likewise, as there are only well-defined success paths, learning to capture them is neither interesting.

Optimisation problems face a control problem during conflict resolution, too. However, it does not affect the solution *path*, but the solution *state*. As operators o^{τ_i} are selected according to the estimation of the objective function at the end of transformation τ_i, and as backtracking is not allowed, selecting an operator eliminates definitively the whole set of final states not accessible from the selected operator's successor state.

This uncertainty concerning operator selection could be seen as prediction problem, for which different learning techniques have been elaborated [Sutton, 1988]. However, we will not deal with this problem. Following the tabu search hypothesis, past search episodes, done on a problem instance, contain knowledge concerning the way how to do search on that same problem instance. Further, it is possible to extract this knowledge and to reuse it to guide future search (on the same problem instance) to obtain good solutions. While this hypothesis leads to intensification and diversification strategies in tabu search, we use it in order to attribute credit or blame to each applied operator together with the state it applied in. This control knowledge is reused during operator selection in future search, in order to discriminate between operators that seem promising in the current state, and operators that showed deceiving performances.

Similar to tabu search, and in opposite to analytical learning in general, this approach does not intend to transfer knowledge between different problem instances.

---

[4] At least in what concerns analytical learning techniques that suppose perfect domain theories. As pointed out by Tambe & Rosenbloom (1993), domain refinement methods are not appropriate to optimisation problems, as these do not suffer from classical domain imperfections, even when operators can be considered as being over-generalised.

5. Learning during Search

Reaching a transformation's final state every applied operator is analysed and judged whether it produced "interesting" effects during the transformation ("interesting" is defined below). The analysis is based on the accuracy of the operator's quality estimation p, as now it is possible to compare the estimation with the objective function evaluation of the really achieved solution state. The result of the analysis will lead to distinct reward for every operator. The reward, together with the context of the operators application, will be memorised, constituting some kind of search experience.

When search goes on, this memory is consulted to investigate whether it contains experiences concerning the applicable operators. If there are, the operator's quality estimation p will be modified according to these experiences. In other words, p will be adapted relative to the reward stored in the experience and relative to the similarity between the current state and the state when the experience was produced. By this way, the selection of an operator to apply is not only based on its estimated contribution to the whole transformation, but on the operator's history, too.

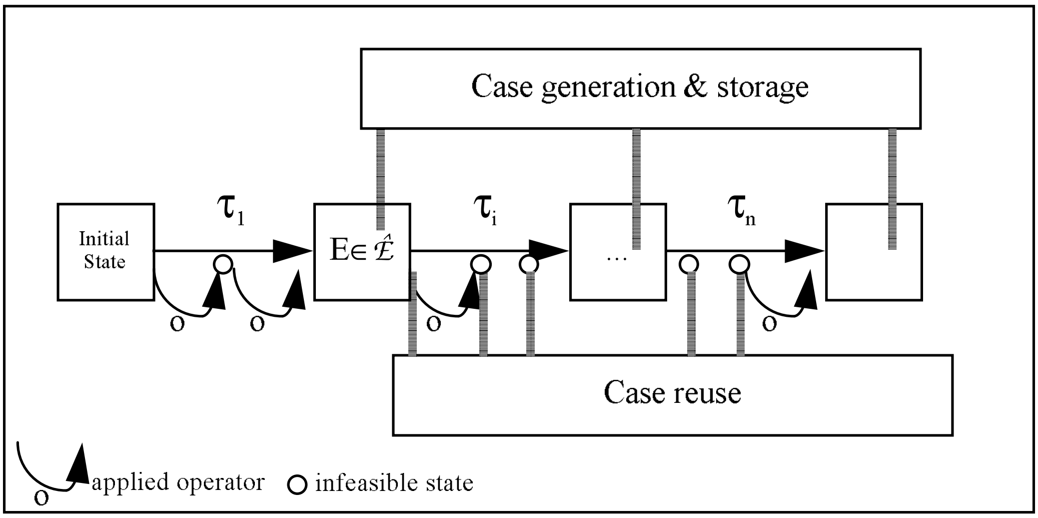

Figure 5: Knowledge transfer principle

A case-based reasoning technique is used to store, retrieve and reuse the operator's experiences in form of cases (Sections 5.4, 5.6). In order to be able to decide whether an application of an operator is similar to another one stored in memory, a similarity measure has to be defined (Section 5.5). As combinatorial optimisation problems have both, numerical and symbolic characteristics, an adapted state representation language has to be defined (Section 5.1). For practical applications, state descriptions may be very large. To be able to compare whether to situations are similar, the notion of instanciation context is defined for applicable operators to capture the state's features that are important to the operator (Section 5.2). The way reward is attributed is outlined in Section 5.3.

5.1. State representation language

Combinatorial optimisation problems are in general described by both, symbolic and numeric characteristics. For example in the single machine scheduling domain, placing a job A at position B is typically represented by a symbolic variable, whereas the set-up cost between two succeeding jobs A and C is better described with a numeric attribute. To capture symbolic and numeric characteristics, symbolic languages like first order logic solely meet some difficulties. To cope with such problems, Bisson (1991) introduced a language based on a subset of first order logic, augmented with the capability to represent numeric attributes. This same approach is used by us.

A state is described by a set of literals of the form `(predicate attribute1 attribute2 ...)`. Attributes have to belong to either of the two classes: objects or values. Furthermore, attributes have to be typed. Consider a blocks world example with the three literals `(on A B)`, `(box A empty)`, `(size A 25)`. Here `on`, `box` and `size` are the predicates, `A` and `B` are attributes of class object, and `empty` and `25` are attributes of class value. Concerning types, the attributes `A` and `B` could be of type `Box`, `empty` of type `FillingType`, and `25` of type `Integer`.

5.2. Operator instanciation context

We suppose that each applicable operators depends only on a delimited subset of the current state's properties. It is in this sense local. The instanciation context aims to capture these local features in order to characterise the operators application in this state without having to retain the state as a whole.

Let $\Omega(o, E)$ denote the set of objects included in the literals on which interacts[5] operator o in state E. The *instanciation context* IC(o, E) is the set of literals l which belong to state E, $l \in E$, and in which any object attribute a, $a \in \Omega(o, E)$, is referenced.

Following this definition, the instanciation context holds all literals that describe the objects intervening in the applicable operator for the current state.

Consider the blocks world example in figure 6. Operator o_1 is applicable in state E_1. We suppose that o_1 interacts on the literals `(on A B)`, `(box A empty)`, `(size A 25)`. The object attributes in these literals are `A` and `B`, so $\Omega(o, E) = \{A, B\}$ and IC(o, E) contains all literals where the object attributes `A` and `B` are referenced.

---

[5] The definition of "interact" is system-dependent. In a production rule context, it means a literal is referenced in the production rule's instanciated precondition.

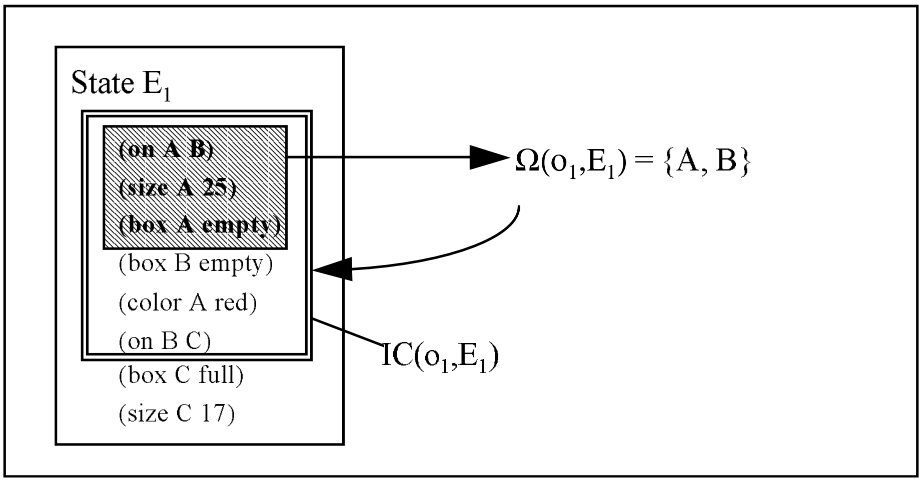

Figure 6: instanciation context for a block's world operator

5.3. Reward attribution

Reward attribution means assigning credit to each member of a sequence of operators (or to a sequence of transformations) according to a scalar reward received from an analysis of the effects resulting from the application of the operator sequence (or transformation sequence).

In the present context of combinatorial optimisation, the nature of reward is ambiguous, especially in what concerns positive reward or credit attributed to operators. One question that has to be answered is: Does the transformation earn credit or blame? A transformation can be considered as being of interest if it did lead to the discovery of a new best solution. However, this interest is only temporal, it will smelt away as soon as a new solution with a better objective function value is found. Likewise, at the beginning of the search it is easier to improve the best solution found so far, but it will get the more and more difficult as search progresses.

Similar arguments concern credit or blame accorded to applied operators. As it is not known, how a *not-applied* operator would have modified the search, the applied one can not be identified for sure to by the best alternative.

As conclusion, for efficiency reasons and because of the ambiguity of reward, a simple, heuristic credit/blame assignment technique is adopted to attribute a *boolean reward*, stating only its existence or non-existence, to every operator applied during each transformation. An operator receives reward in form of credit or blame from two origins: a) global reward, according to the transformation it belongs to, and b) local reward, according to the accuracy it predicted its own contribution to the transformation. The way these rewards are used is presented in Section 5.6.

For every state $E_i \in \tau$ the objective function value $q(E_i)$ is explicitly known. We define $\Delta q_i = q(E_i) - q(E_f^{\tau})$, the real evolution of the objective function between E_i and E_f^{τ}. For $i = 0$, Δq_0 is the evolution over the whole transformation. τ is said to be positive, iff $\mathrm{sgn}(\Delta q_0) > 0$, else τ is negative.

5.3.1. Global reward

A positive (negative) transformation is interpreted in the way that the operator sequence as a whole did produce an interesting (uninteresting) modification to a solution state. According to the type of the transformation, each operator receives reward in form of fixed credit or blame attributed to every operator equally. That is if τ was positive, the operators receive a `bonus`, if τ is negative, they get a `malus`.

5.3.2. Local reward

Local reward is given to each operator according to the difference between $p(o,E_i)$ and Δq_i, i.e. the difference between the estimation and the real evolution of the objective function from E_i to E_f^{τ}. In the case where $sgn(p(o,E_i)) = sgn(\Delta q_i)$, the estimation did design the real direction of the evolution. It can be considered as being almost correct and does not need to be adjusted. So no local reward is assigned in this case.

On the other hand, if $sgn(p(o,E_i)) \neq sgn(\Delta q_i)$, the estimation failed to predict the correct evolution of the objective function, and local reward is assigned to correct this error. Two cases have to be distinguished: `strengthening` is assigned in the case where the prediction was not optimistic enough, i.e. $p(o,E_i)<0$ while $\Delta q_i>0$; whereas `weakening` is assigned when the prediction was too optimistic, i.e. $p(o,E_i)>0$ while $\Delta q_i<0$.

5.3.3. Example

Consider the previously introduced blocks world example. Let us suppose that o_1 belongs to a transformation τ_1 with $q(E_0^{\tau_1}) = 165$ and $q(E_f^{\tau_1}) = 147$, which gives $\Delta q_0 = 18$, an improvement of the objective function over the whole transformation, and designing τ_1 as positive. So, the global reward `bonus` is accorded to o_1.

For local reward, suppose $q(E_1^{\tau_1}) = 135$ and $p(o,E_1) = 20$, which gives $\Delta q_1 = -12$, designing the prediction as being to optimistic and attributing `weakening` to o_1.

5.4. Case generation and indexing

Case-based reasoning [Riesbeck & Schank, 1989] solves problems by adapting solutions of similar problems stored in a case memory. As outlined, these techniques are used here in order to store, not solutions, but experiences concerning operator applications. Design of any case-based reasoning technique has to give answers to questions like: What is a case? How are cases discriminated (or indexed)? Confronted to new problems, which are the relevant cases? How are they reused to solve the new problem?

The case indexing technique outlined below is originally proposed by Veloso (1992) and showed to scale-up well for large case-libraries.

5.4.1. A case

A careful distinction has to be made between the contents of a case and its indexing. While the content holds what has to be memorised, indexing discriminates the case between others and captures the particular context to which the case is relevant. Here, in its simplest form, a *case* c is composed just of the rewards a previous operator's application received, grouping local and global reward in the same case. So, for each operator applied, a different case will be generated once the transformation finished.

5.4.2. Case indexing and storage

Cases represent "nude" reward while indexing will use the relevant characteristics describing the operator's application. As case retrieval will follow these indexes, once a case found, the relevant characteristics have guided retrieval to that particular case.

Before describing case indexing, the notion of "relevant characteristics" has to be explained. The application of an operator in a given state is summarised here with:

- the operator schema os of which the applied operator o was an instance,
- the substitution σ used to instanciate o from os, i.e. o = os σ.
- the instanciation context IC(o, E) of o in the state it applied, E.

Following Veloso (1992), efficiency considerations concerning scale up lead to parameterise the instanciation context IC(o, E) in order to replace constant expressions by parameters. Informal, each distinct occurrence of a constant expression is substituted by a unique, type-dependent parameter.

Consider the previous blocks world example (figure 6). The constants occurring in the instanciation context IC(o_1, E_1) will be replaced by parameters as follows (figure 7): `A`→`box_1`, `B`→`box_2`, `25`→`integer_1`, `empty`→`filling_1`, `red`→`color_1`, `C`→`box_3`. Remark that both occurrences of `empty` are replaced by the same parameter.

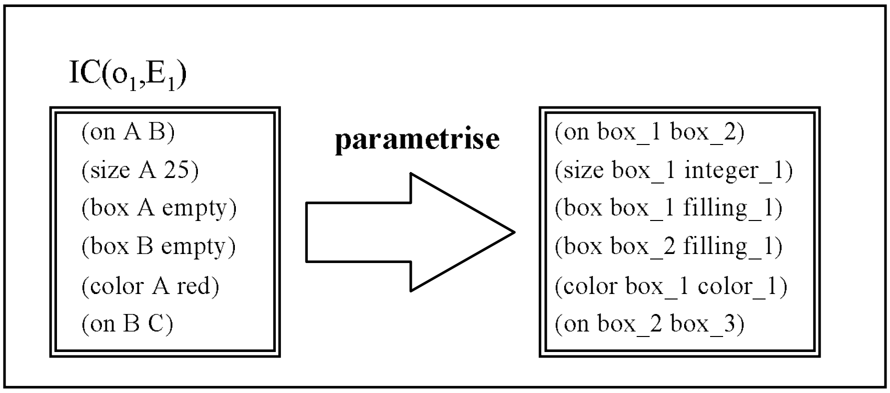

Figure 7: parametrise the instanciation context IC(o_1, E_1)

The cases are *indexed* according to the operator schema and the parametrised instanciation context, using a multi-level indexing schema based on a discrimination net. First level indexing is done by the operator schema, pointing to the entry nodes of the discrimination net where cases relevant to the schema are stored. The discrimination net itself is build with the parametrised instanciation context.

Consider again the blocks world example. Let's suppose that the operator o_1 is an instance of the operator schema os_1 and that the case library is empty. Introducing the case c_1 which represents the application o_1 leads to the discrimination net shown in figure 8. At the first level, os_1 points to a node, `node_0`, containing the parametrised IC(o_1, E_1), while `node_0` points directly to case c_1. Suppose the application of o_1 got reward in form of `bonus` and `weakening`. This is inscribed in the case's body.

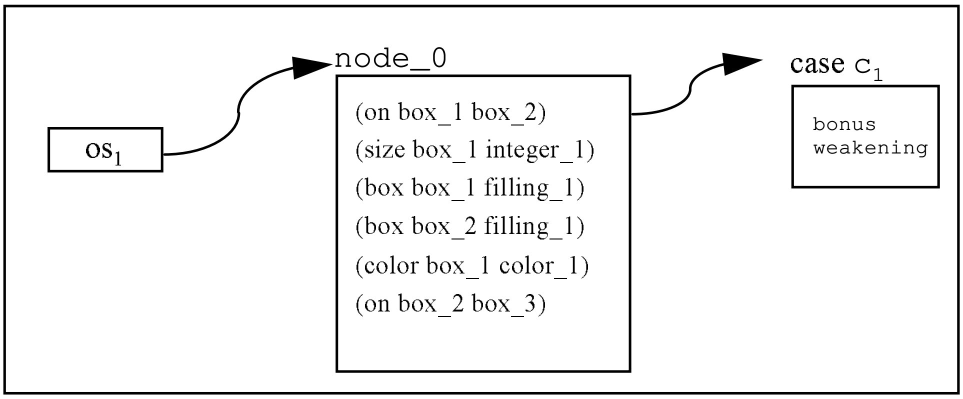

Figure 8: case library after storing c_1

Adding a second case modifies this discrimination net. Let operator o_2 be another instance of the operator schema os_1. Its instanciation context, the parametrisation as well as its introduction in the case library, as case c_2, are shown in figure 9. The common subset of literals between IC(o1) and IC(o2) remain in `node_0`, while the remaining sets of literals each construct a new node, `node_1` and `node_2`. These nodes now point to the cases. Reward of o_2 in inscribed in case c_2.

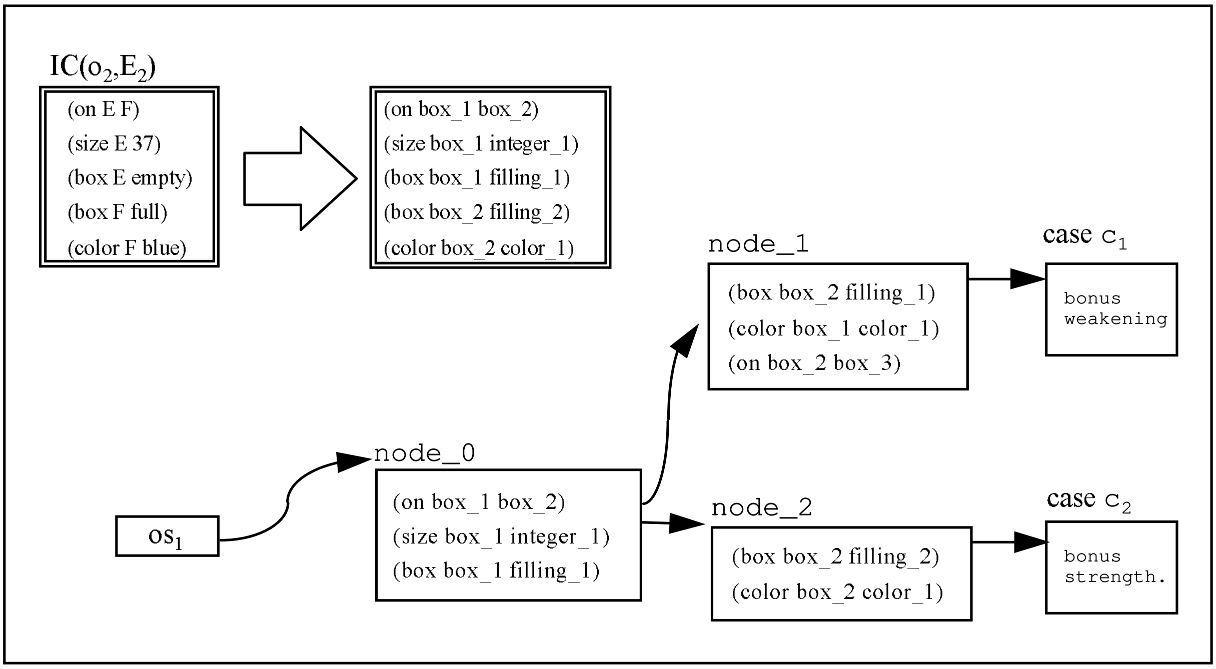

Figure 9: introduction of c_2 into the case library

5.5. Similarity measure

For case reuse, a similarity measure is needed in order to determine the resemblance between the current state of the search E and a case c found in the case library. More precisely, similarity has to be measured between the instanciation context IC of an operator o actually applicable, and the instanciation context IC' used to index case c. Both, IC and IC' are (parametrised) sets of literals.

Any valuable *similarity measure* s should satisfy the following characteristics for two sets of literals P and Q, $\forall P,Q \in \mathcal{L}$:

- s: $\mathcal{E} \times \mathcal{E} \rightarrow [0, 1]$, with s(P,P) = 1, and s(P, Q) = 0, iff $P \cap Q = \varnothing$,
- s(P, Q) = s(Q, P),
- s(P, Q) defined for $|P| \neq |Q|$.

Following Roux (1985), for simplicity and accuracy reasons, Jaccard's similarity is chosen:

$$s(P,Q) = \frac{c}{p+q-c},$$

with p = |P|, q = |Q|, and $c = |P \cap Q|$.

5.6. Case retrieval and reuse

Cases are retrieved in order to be reused during conflict resolution concerning operators applicable in the current state. For every applicable operator, all relevant cases are retrieved. A case is said to be relevant to an operator if the case is indexed by the operator schema of which the operator is an instance. Multiple cases are used to adapt the operator's quality estimation p, where the adaptation has to deal with the fact that different cases may contain absolutely contradictory rewards.

We propose adaptation according to the following characteristics:

- Distinction is made between each type of reward (`bonus`, `malus`, `strengthening`, `weakening`). For positive rewards, the case that matches best is reused, whereas for negative reward, all cases are considered.
- The operator's quality estimation is adapted by a correction factor α in order to remain proportional to its original value. We suppose: $p(o, E) := \alpha \cdot p(o, E)$

Continuing the blocks world example, case retrieval and reuse are introduced informally. Consider an operator o_3 applicable in the current state E. We suppose that o_3 is an instance of the operator schema os_1, too. Case retrieval first follows the operator schema os_1 pointing to the discrimination net's entry nodes concerning o_3, here `node_0` (figure 9). From `node_0`, the cases c_1 and c_2 are found, after passing at `node_1` and `node_2`. Each found case is added to the set of retrieved states C, here $C(o_3) = \{c_1, c_2\}$. For every retrieved case, its parametrised instanciation context $IC'(c_i)$ is reconstructed simultaneously. Likewise, o_3's instanciation context $IC(o_3, E)$ is constructed and parametrised (figure 10).

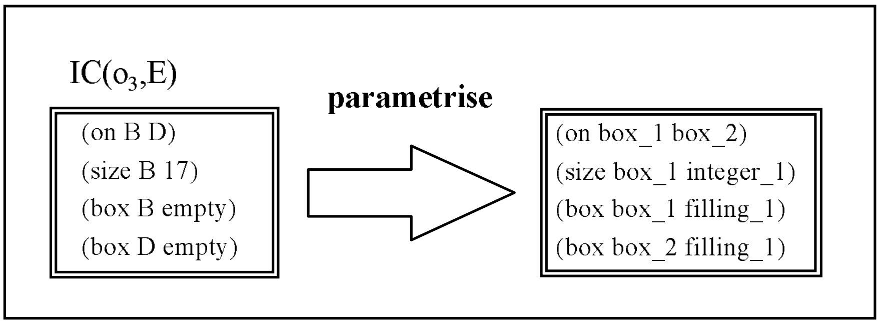

Figure 10: instanciation context $IC(o_3, E)$

Cases are reused to adapt the quality estimation p(o_3,E) according to the reward stored in the retrieved cases. Using the similarity measure s, the closeness between the operator's instanciation context and that of each case are determined.

$$s(IC(o_3,E),IC'(c\quad)) = \frac{4}{6+4-4} = \frac{2}{3}$$

$$s(IC(o_3,E),IC'(c\quad)) = \frac{3}{6+3-3} = \frac{1}{2}$$

To adapt the operator's quality estimation, we propose as correction factor α:

$$\alpha(C) = \frac{1 + \max\limits_{\substack{c \in C(o),\\ \text{bonus}}} \left[s(IC,IC'(c))\right] + \max\limits_{\substack{c \in C(o),\\ \text{strenght}}} \left[s(IC,IC'(c))\right]}{1 + \sum\limits_{\substack{c \in C(o),\\ \text{malus}}} \left[s(IC,IC'(c))\right] + \sum\limits_{\substack{c \in C(o),\\ \text{weakening}}} \left[s(IC,IC'(c))\right]}$$

With the similarities determined previously, we have:

$$\alpha(C) = \frac{1 + \max\left[\,/_3,\ /\,\right] + \max\left[\,/\,\right]}{1 + 0 + \left[\,/\,\right]} = \frac{13}{9} = 1{,}44$$

This means that the quality estimation p of o_3 will be increased by 44%, as there are cases in the case library that a) represent instanciations of the same operator schema, b) have been generate in states similar to the current one, and c) represent mostly positive reward.

6. Summary and future work

In this paper, we introduced a general formalism based on move decomposition in order to reduce the complexity of neighbourhood evaluations in local search techniques. This approach allows the use of a case-based reasoning technique to acquire control knowledge during search. It follows the basic idea of knowledge transfer in tabu search, modifying the notions of intensification and diversification. Intensification uses knowledge in form of cases in order to determine the interest of an applicable operator. Diversification can be achieved using more complex moves which, decomposed into basic operators, are not so computational costly any more.

Scaling up is important for learning techniques dealing with combinatorial optimisation problems, as search can go on for quite a number of iterations. The case storing technique

proposed is close to that used by Veloso (1992), which already showed to scale up satisfyingly.

The presented system is actually under implementation. In order to test the accuracy of move decomposition as well as the learning technique, we plan to apply it to production scheduling problems and to large size, industrial transportation network optimisation problems.

7. Bibliography

Barnes J., Laguna M., A tabu search experience in production scheduling, in: Glover F., Laguna M., Taillard E., de Werra D. (eds.), Tabu Search, *Annals of Operations Research 42* (1-4), Baltzer Science Publishers, Basel, Switzerland, 1993, p. 141-56

Bisson G., A similarity measure to learn in FOL, *Proceedings of RFIA-91*, Paris, France, 1991, p. 913-24

Crainic T., Gendreau M., Soriano P., Toulouse M., A tabu search procedure for multicommodity location/allocation with balancing requirements, in: Glover F., Laguna M., Taillard E., de Werra D. (eds.), Tabu Search, *Annals of Operations Research 42* (1-4), Baltzer Science Publishers, Basel, Switzerland, 1993, p. 359-83

Fikes R., Nilsson N., STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence 2*, 1971, p. 189-208

Georgeff M., Planning, *Ann. Rev. Comput. Sci. 2*, 1987, p. 359-400

Glover F., Tabu Search - Part 1, *ORSA Journal on Computing 1* (3), 1989, p. 190-206

Glover F., Tabu Search - Part 2, *ORSA Journal on Computing 2* (1), 1990, p. 4-32

Glover F., Laguna M., Taillard E., de Werra D. (eds.), Tabu Search, *Annals of Operations Research 42* (1-4), Baltzer Science Publishers, Basel, Switzerland, 1993

Glover F., Taillard E., de Werra D., A user's guide to tabu search, in: Glover F., Laguna M., Taillard E., de Werra D. (eds.), Tabu Search, *Annals of Operations Research 42* (1-4), Baltzer Science Publishers, Basel, Switzerland, 1993, p. 3-28

Kirkpatrick S., Gelatt J., Vecchi M., Optimization by Simulated Annealing, *Science 220* (4598), 1983, p. 671-80

Laird J., Rosenbloom P., Newell A., Chunking in SOAR: The Anatomy of a General Learning Mechanism, *Machine Learning 1*, 1986, p. 11-46

Minton S., *Learning Search Control Knowledge: An Explanation-Based Approach*, Kluwer Academic Press, Boston, 1989

Minton S. (ed.), *Machine Learning Methods for Planning*, Morgan Kaufmann, San Mateo, 1993

Riesbeck C. K., Schank R. C., *Inside case-based reasoning*, Lawrence Erlbaum Associates, Hillsdail, NJ, 1989

Roux M., *Algorithmes de classification*, Masson, Paris, 1985

Papadimitriou C., Steiglitz K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, New Jersey, Englewood Cliffs, 1982

Ruby D., Kibler D., Learning Episodes for Optimization, Proceedings of the International Conference on Machine Learning (ML-92), 1992, p. 279-84

Sutton, Learning to predict by the method of temporal differences, *Machine Learning 3*, 1988, p. 9-44

Taillard E., Robust taboo search for the quadratic assignment problem, *Parallel Computing 17*, 1991, p. 443-55

Tambe M., Rosenbloom P., On the Masking Effect, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1993, p. 526-33

Tate A., Hendler J., Drummond M., A review of AI planning techniques, in: Allen J., Hendler J., Tate A. (eds.), *Readings in Planning*, Morgan Kaufmann, San Mateo, 1990, p. 26-49

Veloso M., *Learning by analogical reasoning in general problem solving*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, 1992