Applying Local Search to Structural Constraint Satisfaction

Alexander Nareyek

GMD — FIRST
German National Research Center for Information Technology
Research Institute for Computer Architecture and Software Technology
Rudower Chaussee 5
D - 12489 Berlin, Germany

alex@first.gmd.de

Abstract

Although domains like continuous workflow planning are promising application candidates for constraint programming techniques, conventional constraint satisfaction representations and algorithms are not expressive and reactive enough. Thus, the present paper extends and combines the concept of structural constraint satisfaction with the use of local search and global constraints to satisfy the advanced requirements.

1 Introduction

Most workflow management systems [Hollingsworth, 1995] have modules to support manual modeling and simulation. Automation of workflow planning is rare, and even less common is continuous planning[1].

Constraint programming (CP) focuses on complex combinatorial search problems. It is well suited to provide a framework for workflow planning as all kinds of constraints can easily be integrated, e.g., temporal, spatial or resource-oriented restrictions of the plan's activities.

A constraint satisfaction problem (CSP) consists of a set of variables $X = \{X_1, \ldots, X_n\}$, where each variable is associated with a domain $D_1, ..., D_n$, and a set of constraints $C = \{C_1, ..., C_m\}$ over these variables. The domains can be symbols as well as numbers, continuous or discrete (e.g., "13", "6.5", "TruckService, Inc."). Constraints are relations between variables (e.g., "$X_a < X_b \times X_c$", "X_a is a supplier of X_b") that restrict the possible value assignments. Constraint satisfaction is the search for a variable assignment that satisfies the given constraints.

[1]In the context of continuous workflow planning, we do not distinguish between a process definition and its process instances as we wish to stress the continuous adaptiveness. A process structure is called *plan* instead.

The application of constraint programming to continuous workflow planning is fairly intractable using conventional constraint satisfaction techniques. Sections 2 and 3 present extensions to the basic CP framework, which are necessary to handle continuous workflow planning. These concepts are combined in Section 4.

2 Structural Constraint Satisfaction

Figure 1 shows two alternative plans. We can either use a car to get to work and read a paper afterwards, or walk to the subway and travel on this while reading the paper. This example shows that there are options regarding not only the variable assignment of plans, but also the graph structure itself.

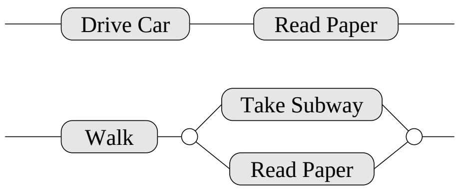

Figure 1: An Example of Plan Alternatives

Conventional constraint satisfaction is unable to handle the graph-generation task, as constraint satisfaction problem formulations are static. There is a given set of constraints and variables, and the structure of the constraint graph does not change. For the task of generating a plan, though, the search for the structure of the constraint graph must be part of the satisfaction process.

This problem should not be confused with the dynamic constraint satisfaction problem (see [Verfaillie and Schiex, 1994] for a brief survey). Dynamic constraint satisfaction tries to revise a variable assignment with given changes to the constraint graph and does not include structural changes as part of the search.

This is realized by the concept of structural constraint satisfaction [Nareyek, 1999]. The basic satisfaction idea is to iteratively change a graph (starting with an empty graph) based on an algebraic graph grammar, which is generically produced using the problem definition. Structural constraints allow us to formulate restrictions on admissible constraint graphs.

2.1 Graph Grammars

This section gives a fairly informal introduction to algebraic graph grammars (see [Ehrig *et al.*, 1973; Rozenberg *et al.*, 1997; Habel *et al.*, 1996] for a detailed overview).

Algebraic graph grammars are a generalization of Chomsky grammars. A graph signature $GSig$ consists of the sorts of vertices V, edges E, and a label alphabet L. The operations of $GSig$ provide source and target vertices for every edge, $s, t : E \to V$, and map a label to every vertex and edge, $l_v : V \to L$ and $l_e : E \to L$.

A match m of graph g_1 to graph g_2 is a partial graph morphism that maps the vertices and edges of g_1 to g_2 such that the graphical structure and the labels are preserved. We use injective matches only.

A production P is a partial morphism between a left-hand side P_l and a right-hand side P_r, which provides information about which elements are preserved, deleted and created in the case of an application of the production. The identity of objects is marked by appended identifiers like `:1` (e.g., in the production in Figure 8). A production is applicable to a graph g if there is a match of P_l to g.

A derivation g_2 from g_1 is the so-called *push-out* graph of an application of an applicable production P. The new graph g_2 is similar to g_1, but the elements of P_r that are not in P_l are added, and elements of P_l that are not in P_r are deleted.

The application of a production may also require application conditions. A negative application condition (NAC) is a morphism $P_l \to n$ that is satisfied if there is no morphism $n \to g$ such that $P_l \to n \to g = P_l \to g$. An NAC is represented by a convex dark area (e.g., in production P_{Less} in Figure 7). For multiple NACs, such as in production P_{Mouse_e} in Figure 9, the conjunction of the conditions must hold. A positive application condition (PAC) is a morphism $P_l \to p$ that is satisfied if there is a morphism $p \to g$ such that $P_l \to p \to g = P_l \to g$. A PAC is represented by a convex light area (e.g., in the left-hand side of production P_{Mouse_r} in Figure 9).

2.2 Graph Elements

A variable of the CSP is represented by a vertex with the label *Variable*. It is graphically depicted by a circular vertex. Constraints could be represented by edges, but there are constraint types that allow a variable number of variables to be included. These constraint types will be called *extensible constraints* (C_e) in contrast to *nonextensible constraints* (C_n), such that $C_e \cup C_n = C$.

As the number of variables incorporated for extensible constraints may vary throughout the search, there must be a simple mechanism to include/exclude variables for constraints. Hence, a constraint is also represented by a vertex, new edges to variables being added in order to incorporate them. A constraint vertex's label corresponds to the type of the constraint. A constraint is graphically depicted by a rectangular vertex.

An SCSP allows the existence of so-called object constraints. These constraints do not restrict the variables' values, but provide structural context information. For example, it must be known which two start and duration variables together form a task object. Otherwise, a NON-OVERLAP constraint, to check if the included tasks are overlapping, might combine start and duration values of different tasks for the check. Object constraints act as a kind of structural broker between variables and conventional constraints. They are represented by a rectangular vertex with a dashed outline. Object constraints can be nonextensible as well as extensible, $O_e \cup O_n = O$, $C \cap O = \emptyset$. Constraints of C can be connected to variables and object constraints, whereas object constraints can be connected to constraints of C as well.

Edges are used to connect vertex elements. The role/position of a variable (or constraint when speaking about objects) within a constraint is often very important. Thus, an edge's label and direction can be used to express its role/position. An edge's direction is indicated by an arrow and the label is displayed in the edge's middle (*NoLabel* if omitted).

Figure 2 shows an example graph with two nonextensible ACTION TASK object constraints, an extensible SUM constraint that restricts the sum of ($\circ \to \square$)-connected variables to equal a ($\square \to \circ$)-connected variable, and a nonextensible LESS constraint that restricts the ($\circ \to \square$)-connected variable to being less than the ($\square \to \circ$)-connected variable. The extensible NON-OVERLAP constraint is redundant because the other constraints already forbid the tasks' overlap.

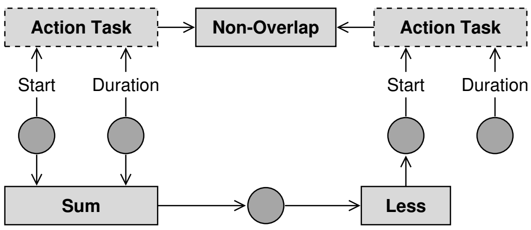

Figure 2: An Example Graph

A general vertex (variable, conventional constraint or object constraint) is graphically depicted by a flat ellipse (see Figure 6).

2.3 Structural Constraints

Structural constraints allow us to formulate restrictions on admissible constraint graphs, e.g., that an ordering activity requires a communication medium to be existent and available, or that driving a car and reading a paper at the same time is forbidden (see Figure 3). If the docking part S_d of a structural constraint S matches the constraint graph, an alternative of the testing part S_t has to match too. A graph g is structurally consistent if there exists a morphism $a \to g$, $a \in S_t$ for every structural constraint S and every possible match $S_d \to g$ such that $S_d \to a \to g = S_d \to g$.

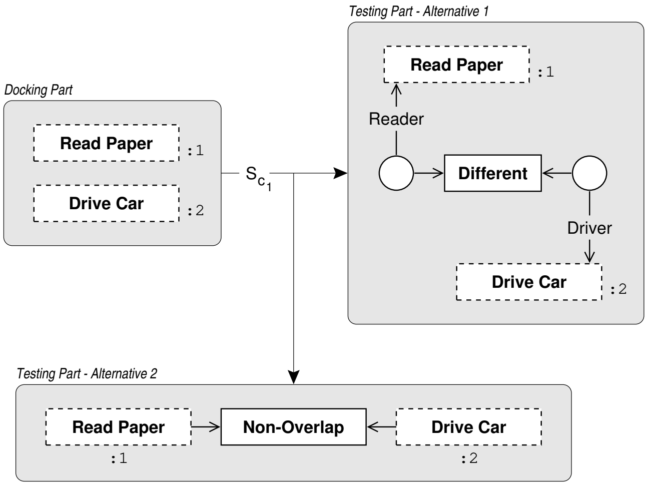

Figure 3: An Example of a Structural Constraint

2.4 Structural Constraint Satisfaction Problems

A structural constraint satisfaction problem $SCSP = (\mathcal{CD}, \mathcal{S})$ consists of a tuple of sets of constraint descriptions $\mathcal{CD} = (\mathcal{C}_n, \mathcal{C}_e, \mathcal{O}_n, \mathcal{O}_e)$ and a set of structural constraints $\mathcal{S}$. The constraint descriptions of $\mathcal{C}_n$ and $\mathcal{O}_n$ are pairs (c, p_{base}) with a nonextensible conventional (or object) constraint c and its embedding graph p_{base}. The constraint descriptions of $\mathcal{C}_e$ and $\mathcal{O}_e$ are quadruple $(c, p_{base}, E, p_{max})$ with an extensible conventional (or object) constraint c, its minimal embedding graph p_{base}, a set of extension graphs E, and the constraint's maximal embedding graph p_{max}.

An embedding graph shows the constraint with all its directly connected neighbor vertices. If an extensible constraint has no maximal embedding, p_{max} is the empty graph. An extension graph shows the constraint connected to the vertices that can be added in one step. Figure 4 shows some components of an example SCSP.

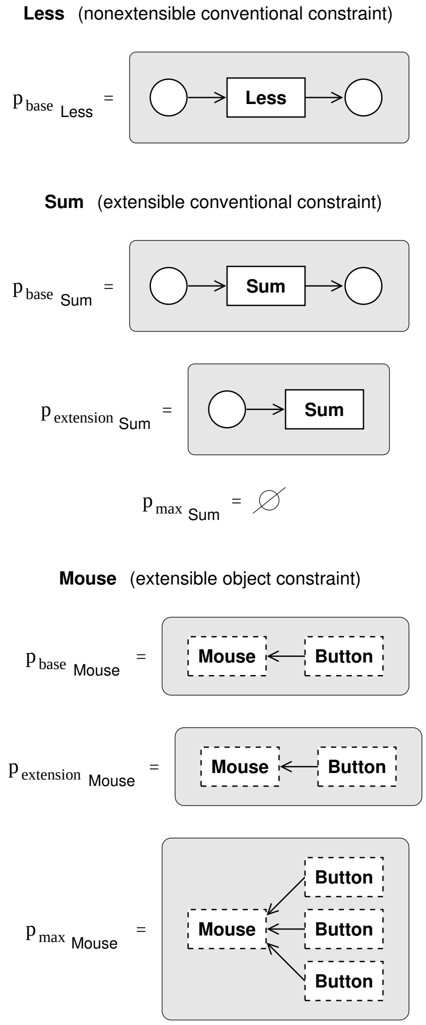

Figure 4: Some Components of an Example SCSP

3 Local Search

The explicit consideration of all alternatives is almost impossible as there may be endless options when designing a plan. In addition, possible changes of the environment, e.g., a machine breakdown or a withdrawal of an order, may necessitate a rescheduling or even a restructuring of the process instances. As not all possibilities can be considered in advance, process modeling and process enactment must be interleaved.

Conventional constraint satisfaction uses **refinement search** (a stepwise narrowing process) by assigning a value to a variable in each refinement. Propagation techniques are used to eliminate inconsistent domain values. In contrast, **local search** techniques perform a search by iteratively changing a complete assignment of variables. For each iteration, a *neighborhood* of potential successor states is considered. A state's quality can be computed by a *cost function*. The basis for the selection of a successor state is mostly the expected improvement with respect to the current state's costs. The term *inconsistency* will be used below as a synonym for *costs*.

The production of a solution by refinement takes a long time because numerous labeling/propagation (and maybe backtracking) cycles have to be performed. Local search is an anytime algorithm, and improvement iterations can mostly be computed very fast, resulting in a temporally tight sequence of improvement steps. This enables initial results to be computed much faster. In addition, tight time limits and large problems usually result in a superiority of local search over complete refinement methods [Wallace and Freuder, 1996; Gent *et al.*, 1997].

A dynamic environment promotes local search as well because the search heuristics do not normally bother about modifications of the search space. This is in contrast to complete methods, which have to update their memory of the already explored search space.

As we wish to have continuous workflow planning with fast reactions to changes in the environment (worker is too late → reschedule and perhaps restructure process instances), local search techniques are an appropriate choice. Moreover, the advantage of refinement search's termination property would be nullified by a combination with structural constraint satisfaction because SCSPs are undecidable.

We focus on the concept of using *global constraints* for local search [Nareyek, 1998]. A global constraint is a substitute for a set of lower-level constraints, additional domain knowledge allowing the application of specialized data representations and algorithms to guide and accelerate search. Each global constraint can compute its subjective satisfaction/costs on its own. Heuristics for building neighbors and for selecting a successor can be directly encapsulated in the global constraints, where the appropriate domain knowledge is available.

Coefficients weight the constraints' subjective costs for the overall cost function. By way of this cost information, a *global search control* selects a constraint in each iteration, which is responsible for the next state improvement (see Figure 5).

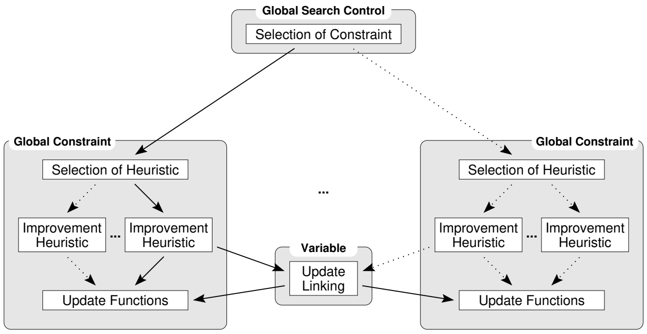

Figure 5: Local Search with Global Constraints

A global constraint for local search must feature a start function to construct its initial structures and to compute its initial inconsistency, an improvement procedure for determining a promising successor state, and an update function for its internal structures and the constraint's cost contribution in case of variable changes.

4 Integration of the Techniques

Given an SCSP, valid constraint graphs must somehow be constructed. The structural constraint satisfaction of [Nareyek, 1999] is a refinement search, and a constraint graph can only be expanded during search. The previous section argued that local search is more appropriate for the continuous workflow planning domain. This section therefore adapts structural constraint satisfaction to the local search approach with global constraints.

4.1 Allowing for Deletion

To search for a structurally consistent graph, the satisfaction approach of [Nareyek, 1999] applies addition productions only and expands the graph in a stepwise manner. This is reasonable for refinement search but not for local search, as all possible states should be reachable from any state. To realize this, there must be deletion productions to reverse the effects of the addition productions. This section describes how generic deletion productions can be created using the SCSP formulation.

The creation of the addition productions is not explained, but the addition productions for the examples are included in the figures as well. The reader is referred to [Nareyek, 1999] for a detailed description. The creation of additional structural constraints for a prevention from redundant extensible constraints is the same as in refinement search (see also [Nareyek, 1999]).

The productions of the following subsections impose certain requirements on the SCSP formulation:

- Nonextensible constraints are not allowed to appear in graphs of other constraints, as the addition and deletion productions of constraints partly incorporate their graphs.
- Constraint-usage cycles are not allowed for the p_{base} graphs of extensible constraints, e.g., that p_{base_A} includes a B constraint and p_{base_B} includes the A constraint.
- If the p_{max} graph of an extensible constraint is nonempty, no sequence of productions that is applied to the constraint's p_{base} graph can produce a graph that includes the p_{max} graph without previously producing the p_{max} graph.

Variables

One production is responsible for reversing the addition of a variable (production P_{v_d} in Figure 6). This requires that the variable not be connected to any constraint because a constraint's minimal p_{base} specification might be violated in consequence. This requirement is ensured by the two NACs.

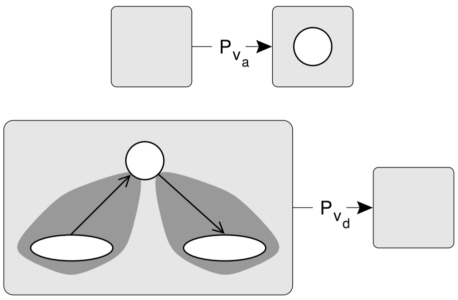

Figure 6: Productions for Variables

Nonextensible Constraints

Corresponding to the addition productions, there must be one deletion production per nonextensible constraint.

The left-hand side of a deletion production is equal to the constraint's embedding graph p_{base}. The right-hand side of the production contains the vertices of the left-hand side without the constraint itself. Production P_{Less_d} in Figure 7 shows the deletion production for the LESS constraint.

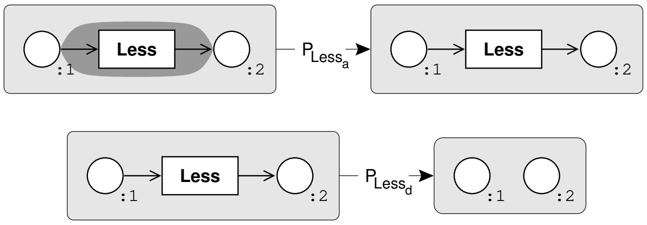

Figure 7: Productions for Nonextensible Constraints

Extensible Constraints

There must also be one deletion production per extensible constraint. The left-hand side of the deletion production is equal to the embedding graph p_{base}. The right-hand side of the production contains the vertices of the left-hand side without the constraint itself. The left-hand side contains two NACs, one with a general vertex with an edge to the constraint, and one with a general vertex with an edge from the constraint. The NACs prevent the constraint from being deleted before the constraint's extensions are deleted. Production P_{Sum_d} in Figure 8 shows an example of the SUM constraint.

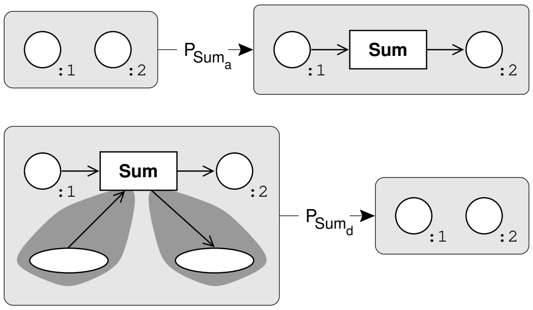

Figure 8: Productions for Extensible Constraints

Constraint Extensions

For every extension of a constraint, there must be a production to remove the extension. The production's left-hand side is similar to the extension graph of E, including a PAC that contains the p_{base} graph, such that the constraint is unified with the constraint of the extension graph (PAC without the constraint itself) to avoid falling below the base graph. For any other constraint vertices of the extension graph, a similar PAC must save from falling below the constraint's base graph. A PAC is not necessary if a base graph contains only the constraint.

The right-hand side of the production is the left-hand side without the PACs and the edges. Production P_{Mouse_r} in Figure 9 shows an example of the reduction of the computer MOUSE object constraint.

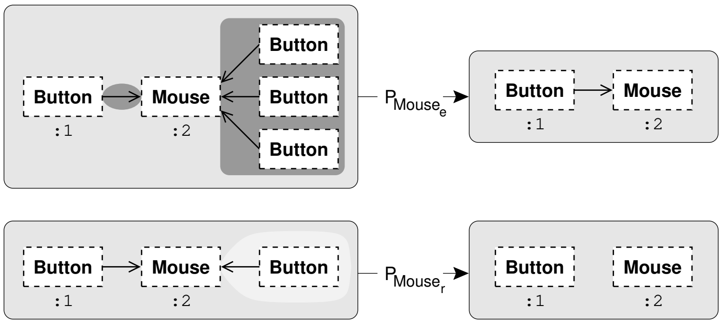

Figure 9: Productions for Constraint Extensions

4.2 Managing Structural Constraints

If the graph is changed by a production, the graph's consistency must be verified again. It would be very costly to test each time for all matches of possible structural constraints. Instead of this, *instances* of structural constraint *types* are memorized. These instances stay matched to a certain part of the constraint graph. Then, the graph must only be reverified with respect to the changes.

Whenever new elements are added by a production, there must be new structural constraint instances for all possible constraints' docking-part matches that include the new elements. Since there might be negative application conditions in the docking parts of existing structural constraints, those with an unsatisfied negative application condition due to the new elements must be excluded.

Whenever elements are deleted by a production, all structural constraint instances that matched these elements with their docking part must be deleted. If the deleted elements were part of a PAC in the docking part, the constraint instances must only be deleted if other elements to satisfy the PAC are not available. The deletion of elements may also mean that NACs of potentially applicable constraints' docking parts are no longer applicable. Structural constraint instances must be added in these cases.

4.3 Using Global Constraints

The approach of using global constraints for local search covers only the satisfaction of conventional global constraints. Besides conventional global constraints, there are now global object and structural constraints as well. Conventional and object constraints are added/deleted by productions, whereas there must be a check after each application of a production to decide on the addition/deletion of global structural constraints (see Section 4.2). After application of a production, the inconsistency of affected structural constraints must be updated.

Extensible global (conventional or object) constraints must feature additional update functions to integrate/disintegrate variables and object constraints.

Global Conventional Constraints
The heuristics of global conventional constraints can now change the graph structure by the application of a production or a sequence of productions. For example, the graph contains a structure like the first test alternative of Figure 3, and the DIFFERENT constraint's heuristic would decide on a structural change because it is not able to enforce the inequality of the reader and the driver variable. In consequence, the DIFFERENT constraint might apply the production $P_{Different_d}$ to delete itself.

Global Structural Constraints
The global structural constraints are very similar to global conventional constraints. One difference is that their costs are not associated with variable assignments, but with the existence or nonexistence of graph elements. Thus, their heuristics can only apply productions. A change of variable values is not allowed (apart from an initial value assignment that they have to provide for created variables).

For example, after deletion of the DIFFERENT constraint, the global structural constraint that corresponds to the structural constraint in Figure 3 may become inconsistent. The structural constraint's heuristic might decide to satisfy its second alternative because the first alternative has become inconsistent. This could be accomplished by an integration of the tasks into an existing NON-OVERLAP constraint. Other options are to create a new NON-OVERLAP constraint between the tasks, or to delete one of the tasks (and thereby the structural constraint itself).

Global Object Constraints
Global object constraints are only a reference structure and do not have cost or improvement functions. Their role is to maintain a correct linking of conventional constraints and variables such that the variable updates can be passed correctly and the conventional constraints are aware of the variables' structural connection.

A conventional constraint has to inform connected object constraints about the variables it is interested in, as not all connected variables may be important. For example, a NON-OVERLAP constraint needs information about an involved DRIVE CAR task's start and duration, but not about the task's driver.

Global Search Control
If a structural constraint becomes unsatisfied by a graph change, the global search control must ensure a correct graph structure before proceeding, as conventional constraints cannot handle inconsistent structures. Thus, the global search control always selects a structural constraint to improve the current state if there is an inconsistent structural constraint (see Figure 10).

Goal Optimization
So far, there has only been a search for a consistent plan. But we wish to go beyond satisfaction and optimize the

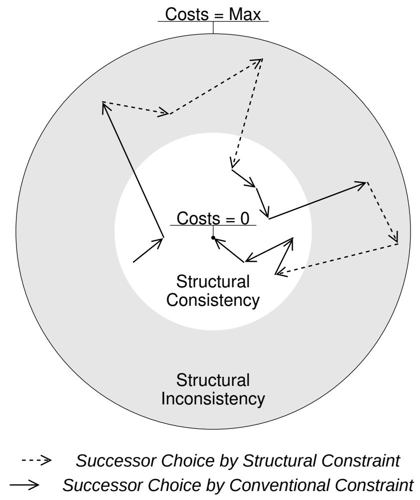

Figure 10: Exploration of the Search Space

plan according to given planning goals. These goals can have a conventional as well as a structural nature. Thus, conventional and structural constraints can have a second cost contribution for a second cost function (*goal function*).

For conventional constraints, it is pretty obvious how the goal function can be influenced, e.g., the completion time of a task, the workload of a resource, or monetary resource costs. The contribution of structural constraints to the goal function focuses mostly on graph minimization, e.g., a structural constraint that forbids redundant structures like the NON-OVERLAP constraint in Figure 2, or one that looks after unconnected object constraints.

The global search control handles this second cost function as well. The functions may compete such that an improvement of one function value may lead to a deterioration in the other function value. Thus, various options for the selection of a function for improvement should be customizable, e.g., alternating phases of a predefined length or with special abort criteria, or a general cost measure consisting of the addition of the single function values multiplied by static (or even variable) coefficients.

5 Conclusion

Representations like Graphplan's Planning Graph [Blum and Furst, 1997] are tailored to conventional action planning only and do not include constraints with respect to time, space or other resources. Unlike these planning formalisms, the presented approach allows us to formulate all kinds of constraints in a natural and coherent way.

Moreover, conventional action-planning formalisms focus on optimizing a plan's length. This property is mostly irrelevant for real-world problems. For optimization goals with non-linear behavior with respect to a plan's length, the approach of structural constraint satisfaction allows us to explore the search space much more appropriately.

Composite CSPs [Sabin and Freuder, 1996] aim at a similar extension of the conventional constraint satisfaction paradigm as SCSPs. A composite CSP expresses subgraph alternatives in a hierarchical manner. This allows optimized search guidance, but requires manual preprocessing to build the hierarchy. The creation of the hierarchy is often problematic, as completeness and appropriate structure are not always obvious.

The approach of considering maximal structures for plans, where substructures can be deactivated if they are superfluous, is suitable for problems with a few variations only. Structural constraint satisfaction provides a very general and declarative way to handle the problem of an infinite number of potential graphs.

The use of local search with global constraints allows real-time workflow planning and provides a mechanism to include domain-dependent knowledge. This allows us to revise a current state of the search on a more global level with domain-specific knowledge, while preserving features like declarativeness, reusability and maintenance. Furthermore, the application of local search allows the easy online integration of environmental changes because of its continuous repair approach.

Acknowledgments

The underlying EXCALIBUR project is being supported by the German Research Foundation (DFG), NICOSIO, Conitec Datensysteme GmbH, and Cross Platform Research Germany (CPR).

References

[Blum and Furst, 1997] A. L. Blum and M. L. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 90, pages 281–300, 1997.

[Ehrig *et al.*, 1973] H. Ehrig, M. Pfender and H. J. Schneider. Graph Grammars: An Algebraic Approach. In *Proceedings of the 14th Annual Symposium on Switching and Automata Theory (SWAT)*, pages 167–180, 1973.

[Gent *et al.*, 1997] I. P. Gent, E. MacIntyre, P. Prosser and T. Walsh. The Scaling of Search Cost. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 315–320, Providence, Rhode Island, July 1997. American Association for Artificial Intelligence.

[Habel *et al.*, 1996] A. Habel, R. Heckel and G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae*, Vol. 26, No. 3 & 4, 1996.

[Hollingsworth, 1995] David Hollingsworth. The Workflow Management Coalition Specification: the Workflow Reference Model. Technical Report TC00-1003, Workflow Management Coalition, Hampshire, UK, January 1995.

[Nareyek, 1998] Alexander Nareyek. Using Global Constraints for Local Search. In *Proceedings of the DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimization*, P5: 1-18, Piscataway, New Jersey, September 1998. Center for Discrete Mathematics and Theoretical Computer Science. Revised version available via http://www.first.gmd.de/concorde/excalibur/.

[Nareyek, 1999] Alexander Nareyek. Structural Constraint Satisfaction. AAAI'99 Workshop on Configuration, to appear.

[Rozenberg *et al.*, 1997] G. Rozenberg (ed.). *The Handbook of Graph Grammars. Volume I: Foundations*, Reading, 1997. World Scientific.

[Sabin and Freuder, 1996] D. Sabin and E. C. Freuder. Configuration as Composite Constraint Satisfaction. In *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

[Verfaillie and Schiex, 1994] G. Verfaillie and T. Schiex. Solution Reuse in Dynamic Constraint Satisfaction Problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 307–312, Seattle, Washington, August 1994. American Association for Artificial Intelligence.

[Wallace and Freuder, 1996] R. J. Wallace and E. C. Freuder. Anytime Algorithms for Constraint Satisfaction and SAT problems. *SIGART Bulletin* 7(2), 1996.