Construction and Repair: A Hybrid Approach to Search in CSPs

Konstantinos Chatzikokolakis, George Boukeas, and Panagiotis Stamatopoulos

Department of Informatics and Telecommunications, University of Athens
Panepistimiopolis, Athens 15784, Greece
{c.chatzikokolakis,boukeas,takis}@di.uoa.gr

Abstract. In order to obtain a solution to a constraint satisfaction problem, *constructive* methods iteratively extend a consistent partial assignment until all problem variables are instantiated. If the current partial assignment is proved to be inconsistent, it is then necessary to backtrack and perform alternative instantiations. On the other hand, *reparative* methods iteratively repair an inconsistent complete assignment until it becomes consistent. In this research, we investigate an approach which allows for the combination of constructive and reparative methods, in the hope of exploiting their intrinsic advantages and circumventing their shortcomings. Initially, we discuss a general hybrid method called CR and then proceed to specify its parameters in order to provide a fully operational search method called CNR. The reparative stage therein is of particular interest: we employ techniques borrowed from local search and propose a general cost function for evaluating partial assignments. In addition, we present experimental results on the open-shop scheduling problem. The new method is compared against specialized algorithms and exhibits outstanding performance, yielding solutions of high quality and even improving the best known solution to a number of instances.

Keywords: constraint satisfaction, search, heuristics

1 Introduction

A great number of interesting combinatorial problems can be viewed as constraint satisfaction problems (CSPs), involving a finite set V of variables and a finite set C of constraints between the variables. Given a CSP, the goal is to obtain a *complete consistent assignment*, that is to assign a value to every variable (complete) so that all constraints are satisfied (consistent). Search methods for obtaining solutions to CSPs can be broadly characterized as *constructive* or *reparative*. Constructive (global) methods iteratively extend a consistent partial assignment until a consistent complete assignment is obtained. If the current partial assignment is proved to be inconsistent, it is then necessary to backtrack and explore alternative assignments. On the other hand, reparative (local) methods iteratively modify an inconsistent complete assignment until a consistent complete assignment is obtained. The methods in each category exhibit certain features which are, in fact, complementary. In this research, we investi-

G.A. Vouros and T. Panayiotopoulos (Eds.): SETN 2004, LNAI 3025, pp. 342–351, 2004.

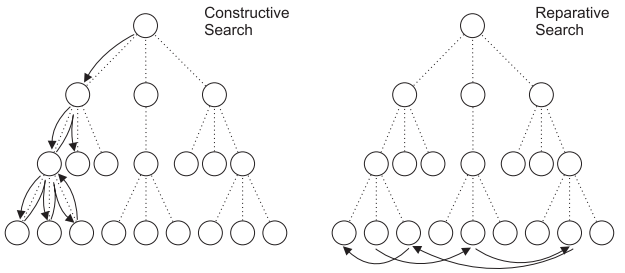


Fig. 1. Constructive methods extend consistent partial assignments or otherwise backtrack. Reparative methods modify inconsistent complete assignments.

gate a framework which allows for the combination of constructive and reparative methods, in the hope of exploiting their intrinsic advantages and circumventing their shortcomings. In the generic CR framework we propose, search is performed in two alternating stages: in the constructive stage a consistent partial assignment is iteratively extended. However, if the current partial assignment is proved to be inconsistent, backtracking is replaced by a reparative stage, in which the inconsistent partial assignment is iteratively modified until it becomes consistent. In order to evaluate the CR framework experimentally, we specify its parameters and provide a fully operational method called CNR-search. The reparative stage therein is of particular interest: we employ techniques borrowed from local search and propose a general cost function for evaluating partial assignments.

The rest of this paper is organized as follows: In Sects. 2 and 3, we examine constructive and reparative search methods separately, identifying some of their most characteristic features. In Sect. 4, we discuss the generic CR search framework. In Sect. 5 we proceed to provide implementations for both the construction and repair operators of the CR framework, thus obtaining a concrete search algorithm called CNR-search. In Sect. 6, we present experimental results from the application of CNR to the *open-shop* scheduling problem. A discussion regarding these excellent results, as well as the method in general, can be found in the concluding Sect. 7.

1.1 Related Work

The notion of merging constructive and reparative features into a hybrid search framework has been investigated in various forms. In some cases, the coupling between the two approaches is loose: constructive and reparative modules exchange information but essentially operate independently [1, 2]. In other frameworks, where the integration is of a higher degree, the reparative process employs constructive methods in order to (systematically) explore the neighborhood [3, 4]. However, our research is more closely related to the approaches described in [5, 6], where repair operators are applied on partial inconsistent assignments, obtained by construction. In both cases, the repair operator undoes previous instantiations, essentially performing some form of non-systematic dynamic backtracking.

Especially in [6], the reparative process is guided by conflict-based heuristics and is coupled with tabu search. Our general CR framework encompasses such approaches while not being restricted to a specific repair operator. In CNR-search, the repair operator does not employ backtracking.

2 Constructive Search

Constructive methods iteratively extend a consistent partial assignment until a consistent complete assignment is obtained. If the current partial assignment is proved to be inconsistent, it is then necessary to backtrack and explore alternative assignments. The fact that search is performed in the space of partial assignments is a defining feature. Proving that a particular partial assignment is inconsistent promptly removes the need to explicitly enumerate all the inconsistent complete assignments in the subtree below it. Consistency techniques can be exploited for reasoning about partial assignments and pruning the search space. On the other hand, operating on partial assignments essentially restricts the search process to a subset of the search space at any given time. Poor search decisions can confine the search process to unproductive branches and are computationally expensive to overcome.

All constructive methods can be described using the generic algorithm of Fig. 2(a). The list L employed therein comprises all partial assignments which remain to be explored and can potentially be backtracked to, in case the current one is proved inconsistent. The use of L allows for a systematic exploration of the search space and endows constructive methods with *completeness*: it is guaranteed that the entire search space will eventually be explored. The `extend` function generates possible extensions of the current partial assignment α, returns one of them and inserts the rest into L so that they can be backtracked to. The `backtrack` function selects and returns an assignment out of L.

3 Reparative Search

Reparative methods iteratively modify an inconsistent complete assignment until a consistent complete assignment is obtained. Because of the fact that search is performed exclusively in the space of complete assignments, the features of reparative methods are complementary to those of constructive ones. The search process is endowed with flexibility, since it is possible to perform arbitrary leaps to complete assignments throughout the search space. However, systematic exploration of the search space and completeness are forsaken, consistency techniques can no longer be exploited and the search process is particularly sensitive to the existence of local optima.

All reparative methods can be described using the generic algorithm of Fig. 2(b). The `repair` function applies a repair operator on assignment α and returns the resulting modified assignment.

```
constructive-search (α) {
    L ← ∅
    while (not solution(α)) {
        if (consistent(α))
            α ←extend(α, L)
        else if (L ≠ ∅)
            α ←backtrack(α, L)
        else
            return no-solution
    }
    return α
}
```

(a)

```
reparative-search (α) {
    while (not solution(α))
        α ←repair(α)
    return α
}
```

(b)

Fig. 2. General algorithms for (a) constructive and (b) reparative search.

4 Search by Construction and Repair

In this section, we discuss a generic search framework which incorporates both construction and repair operators. Intuitively, search is to be performed in two alternating stages:

Constructive Stage. A construction operator is iteratively applied on the current consistent partial assignment, extending it until it becomes inconsistent or until a complete consistent assignment is obtained.

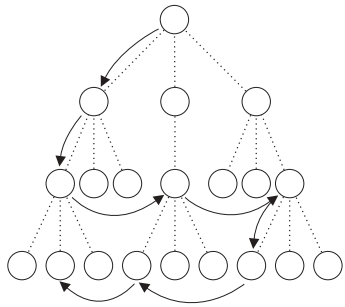
Repair Stage. A repair operator is iteratively applied on the current inconsistent assignment, modifying it until it becomes consistent.

The exploration of the search space using the hybrid CR-search algorithm, is depicted in Fig. 3(b), where the alternating search stages are apparent. Search with CR is performed in the space of both partial and complete assignments. In the constructive stage, it is possible to employ consistency techniques in order to prune the search space and discover inconsistencies. The repair operator on partial assignments allows leaps to distant areas of the search space, overcoming poor search decisions made during partial assignment construction. The variable domain information maintained in the constructive stage can potentially be exploited for guiding the reparative stage, as is explained in Sect. 5.

The generic CR algorithm is described in Fig. 3(a). The `extend` function is inherited from constructive search and implements the construction operator, whereas the `repair` function is a generalized version of the operator encountered in reparative search which can also be applied on partial assignments. Different implementations of these abstract functions give rise to different specializations of the generic CR-search framework. Constructive search itself can be obtained from CR by using backtracking as the repair operator. Reparative search can also be obtained from CR by applying the repair operator only on complete assignments. This means that CR is strictly more general that constructive or reparative search alone.

```
cr-search (α) {
    L ← ∅
    while (not solution(α)) {
        if (consistent(α))
            α ←extend(α, L)
        else
            α ←repair(α, L)
    }
    return α
}
```

(a)



(b)

Fig. 3. Construction-Repair search. (a) The generic search algorithm and (b) an illustration of the manner in which the search tree is traversed.

```
CNR-extend(α, L) {
    v ← h_v(α)
    ℓ ← h_ℓ(α, v)
    return consistency(α ⊙_{v_i} ℓ)
}
```

(a)

```
CNR-repair(α, L) {
    α ← arg min_{α′∈N(α)} f(α′)
    return consistency(α)
}
```

(b)

Fig. 4. Implementation of the (a) constructive and (b) reparative stages. Applying the operator $\odot_{v_i}$ on an assignment α instantiates the i-th variable of α.

5 Search by Construction and Neighborhood Repair

In this section, we provide implementations for both the construction and repair operators of the CR framework, thus obtaining a concrete search algorithm called CNR-search (construction-neighborhood repair).

5.1 Constructive Stage

The implementation of the `extend` function in the constructive stage is contained in Fig. 4(a). The variable ordering heuristic function h_v selects an uninstantiated variable and the value ordering heuristic function h_ℓ selects a value out of the domain of a variable. The selected variable is instantiated with the selected value and the `consistency` function is invoked in order to propagate the effects of the instantiation to the domains of the other variables. Note that in our implementation, the `consistency` function implements a modified version of *arc consistency* which does not terminate when an inconsistency is detected. Instead, propagation continues without taking into account any variables with an empty domain. The remaining domain information is to be exploited during the reparative stage.

5.2 Reparative Stage

The *neighborhood* $N_R(\alpha)$ of an assignment α is the set of assignments which are accessible from α in a single application of the repair operator R. Formally,

a neighborhood is a mapping $N_R : \mathcal{A} \rightarrow 2^{\mathcal{A}}$, where $\mathcal{A}$ is the set of all assignments. In the implementation of the `repair` function in the reparative stage, the neighborhood of the current *partial* assignment is computed (given a repair operator R), a member of the neighborhood is selected greedily according to a cost function f described below and the `consistency` function is invoked in order to enforce consistency. Different repair operators give rise to different neighborhoods.

In CNR there is no restriction imposed on the neighborhood $N_R(\alpha)$ of an assignment α, which is determined by the repair operator R. There exist general neighborhoods which are applicable to arbitrary CSPs and are often encountered in the literature. In this research, the ALTER repair operator has been employed, yielding a neighborhood $N_a(\alpha)$ which contains any assignment obtainable from α by modifying the value of a single instantiated variable. The domains of all variables uninstantiated in α are reset to their original domains.

The cost function $f : \mathcal{A} \rightarrow \mathbb{R}$ evaluates the partial assignments in a neighborhood in order to guide the reparative process towards consistent partial assignments. Therefore, f must evaluate the *extent* of constraint violation in an assignment and must be minimized for consistent assignments. Along the same lines, a well-known function described in [7] returns the number of violated constraints but is not directly applicable on partial assignments. We propose that, given an assignment $\alpha = \langle a_1, a_2, \ldots, a_n \rangle$ where a_i is the domain of variable v_i, the cost function evaluating α be:

$$f(\alpha) = \rho \cdot |\{v_i \in V \mid a'_i = \emptyset\}| - \prod_{v_i \in V} \max\{|a'_i|, 1\}$$

where $\alpha' = \texttt{consistency}(\alpha)$, and ρ a large penalty constant. The first term pertains to the number of variables with an empty domain. The greater the number of such variables in an assignment, the more this assignment is penalized. The second term is a tie-breaker and pertains to the product of domain sizes. This product reflects the number of complete assignments which are extensions of α and the greater this number is, the less *constrained* is the assignment. Recall from Sect. 5.1 that this function is applicable because we use a modified version of arc-consistency.

In practice, the neighborhood can be prohibitively large to compute. Its size can be reduced if a window of w variables is selected and the repair operator is only applied to them, disallowing any modifications to variables outside the window. In addition to reducing the size of the neighborhood, using a window also allows for consistency to be enforced in a more efficient manner. The selection of the window size w is critical: computational performance is improved as w decreases but the repair operator becomes less effective since a smaller neighborhood is available. Except for the size of the window, there is also the issue of which variables to include in it. The simplest method is to order the variables lexicographically, include w consecutive variables in the window and slide the window so that it contains a different set of variables in every execution of the reparative stage. This produces an effect similar to that of tabu search: a

recently modified variable leaves the window and will not be modified again for a number of iterations. As a result, it is highly unlikely that the same assignment is reached more than once or that an infinite loop is entered.

6 Experimental Results on a Scheduling Problem

In this section, we present experimental results from the application of CNR to *open-shop*, a scheduling problem. In open-shop, a set of n jobs, each consisting of m tasks, must be processed on a set of m machines. Task i of job j requires processing time t_{ij} and must be executed by machine i. Each machine can execute one task at a time and no two tasks of the same job can be executed simultaneously. Our goal is to find a non-preemptive schedule that minimizes the *makespan*, that is the finish time of the latest task. For $m > 2$ non-preemptive open-shop is NP-hard [8] and problems of very small size still remain unsolved. This is a problem particularly appropriate for methods utilizing repair operators.

6.1 Application of the Search Method

In the experiments performed with CNR, open-shop is handled as a general CSP and no problem-specific techniques are employed:

Constructive Stage. In this stage, the variable ordering heuristic selects the variable with the smallest domain, whereas the value ordering heuristic selects the smallest value in the domain (which is only reasonable when minimizing the makespan). Both ordering heuristics are elementary.

Repair Stage. In this stage, the ALTER neighborhood with first improvement selection is used. Experiments with alternative neighborhoods have been performed, although not extensively, since it immediately became apparent that the ALTER neighborhood is particularly well-suited for the problem at hand. Due to the large number of values in each domain, only a small number of randomly selected values is examined for each variable and, in addition, a sliding variable window of size $w = 1$ is employed. Experiments with larger windows showed that the increase in window size hampers computational performance without improving solution quality.

To minimize the makespan, CNR is coupled with *branch-and-bound*. After each run of CNR, a constraint is added which enforces new solutions to be of better quality, rendering the current assignment infeasible. Search continues by repairing this assignment, which is of good quality and likely to be repaired easily.

Since CNR is not complete, a stopping criterion must be used: in our experiments, the search was interrupted after 120 minutes of computation and only if the time elapsed after the last improvement exceeded 30 minutes. In some cases where the best known solution is available, search was interrupted when this solution was obtained.

6.2 Experiments

We applied CNR on three sets of benchmark problems from the literature:

- 40 instances by Taillard, contained in [9], with sizes ranging from 4×4 to 10×10 and known optimal solutions.
- 52 instances by Brucker, contained in [10], with sizes ranging from 3×3 to 8×8. The optimal solution is not always known but all instances have a classical lower bound of 1000.
- 80 instances by Guéret & Prins, presented in [11] along with an improved lower bound for open-shop. These instances are especially designed so that the proposed bound differs significantly from the classical one.

For these benchmark instances, the quality of the final solutions obtained with CNR is compared against that of a diverse range of algorithms for open-shop:

- the genetic algorithm GA of Prins [12],
- the decision-repair algorithm TDR of Jussien and Lhomme [6],
- (only for the Taillard instances) two specialized tabu search methods TS-A and TS-L, of Alcaide et al. [13] and Liaw [14] and
- (only for the Guéret & Prins instances) the improved branch-and-bound algorithm BB of Guéret & Prins [15].

Experimental results for these algorithms have been published collectively in [6] (available at `http://njussien.e-constraints.net/palm-os.html`). Note that in open-shop scheduling, the quality of the final solution obtained is of primary interest, regardless of the time required to obtain it. Open-shop is a hard problem where quality is considered more important than efficiency. Besides, the execution time of some algorithms strongly depends on the particular instance, while in some cases it is not reported at all. The two hour limit on the execution time of CNR is strict, compared to that of most other algorithms.

Results for the Taillard benchmarks are displayed in Table 1. Except for the 10×10 instances, the performance of CNR is excellent. Results for the Brucker benchmarks are displayed in Table 1. Once again CNR exhibits prime performance for all sets of instances up to size 7×7, whereas for the 8×8 instances, GA is only slightly better.

Results for the Guéret & Prins instances are presented in Table 2 and are outstanding, since CNR prevails over all other methods in every problem size. In fact, CNR yields the best solution quality for all instances in the set, except for a single 9×9 instance. Moreover, the solution quality obtained for all instances of size 10×10, is strictly better than that of any other algorithm. Together with TDR, CNR solves the greatest number of instances to optimality. It is clear that CNR is particularly well-suited for this series of problems.

The qualitative performance of CNR is also underlined by the fact that it managed to improve the best known solution for many instances, in both series with existing open problems. Table 3 shows the number of open problems for each series as well as the number of improved ones.

Table 1. Results for the Taillard and Brucker instances. Each column corresponds to a particular method and contains the average and maximum percent deviation from the optimum, as well as the number of instances solved to optimality (in parentheses). In the cases where the optimum is not known, the classical lower bound is used. An asterisk marks the method(s) exhibiting the best performance across a row.

size	TS-A	TS-L	GA	TDR	CNR
4×4	–	0/0(10)*	0.31/1.84(8)	0/0(10)*	0/0(10)*
5×5	–	0.09/0.93(9)	1.26/3.72(1)	0/0(10)*	0/0(10)*
7×7	0.75/1.71(2)	0.56/1.77(6)	0.41/0.95(4)	0.44/1.92(6)	0.25/0.95(6)*
10×10	0.73/1.67(1)	0.29/1.41(6)	0/0(10)*	2.02/3.19(0)	0.69/2.04(2)
3×3	–	–	0/0(8)*	0/0(8)*	0/0(8)*
4×4	–	–	0/0(9)*	0/0(9)*	0/0(9)*
5×5	–	–	0.36/2.07(6)	0/0(9)*	0/0(9)*
6×6	–	–	0.92/2.27(3)	0.71/3.5(6)	0.08/0.76(8)*
7×7	–	–	3.82/8.2(4)	4.4/11.5(3)	3.22/8.2(5)*
8×8	–	–	3.58/7.5(5)*	4.95/11.8(1)	3.64/8.2(4)

Table 2. Results for the Guéret & Prins instances. Each column corresponds to a particular method and contains the number of instances in which the best known results were obtained (or even improved, in the case of CNR), as well as the number of instances solved to optimality.

size	BB	GA	TDR	CNR
3×3	10/10*	10/10*	10/10*	10/10*
4×4	10/10*	10/10*	10/10*	10/10*
5×5	10/10*	8/8	10/10*	10/10*
6×6	9/7	2/1	10/8*	10/8*
7×7	3/1	6/3	10/4*	10/4*
8×8	2/1	2/1	10/4*	10/4*
9×9	1/1	0/0	8/2	9/2*
10×10	0/0	0/0	0/0	10/0*

Table 3. For a number of *open* instances (unknown optimal solution), CNR managed to improve the best known solution.

Series	**Open Inst.**	**Improved Inst.**
Taillard	0	–
Brucker	8	3
Guéret & Prins	32	12

7 Conclusions

In this work, we describe the general CR framework which allows for the combination of constructive and reparative features into a hybrid abstract search

method. The most characteristic feature of the CR framework is that a repair operator is applied on *partial* (rather than complete) inconsistent assignments, obtained by construction. Such an approach retains many of the advantages of both constructive and reparative methods. By specifying some of the abstract parameters of the CR framework, we obtain a search method called CNR-search. The main difference between CNR-search and the relevant approaches in [5, 6] is the nature of the repair operator, which does not perform backtracking. The extensive experimental results presented Sect. 6 clearly exhibit that CNR-search can be effective in a hard combinatorial problem such as open-shop, prevailing even over closely related methods such as TDR.

References

1. Nareyek, A., Smith, S.F., Ohler, C.M.: Integrating local-search advice into refinement search (or not). In: Proceedings of the CP 2003 Third International Workshop on Cooperative Solvers in Constraint Programming. (2003) 29–43
2. Zhang, J., Zhang, H.: Combining local search and backtracking techniques for constraint satisfaction. In: AAAI-96. (1996) 369–374
3. Schaerf, A.: Combining local search and look-ahead for scheduling and constraint satisfaction problems. In: IJCAI-97. (1997) 1254–1259
4. N. Roos, Y.P. Ran, H.v.d.H.: Combining local search and constraint propagation to find a minimal change solution for a dynamic csp. In: AIMSA. (2000)
5. Prestwich, S.: Combining the scalability of local search with the pruning techniques of systematic search. Annals of Operations Research **115** (2002) 51–72
6. Jussien, N., Lhomme, O.: Local search with constraint propagation and conflict-based heuristics. Artificial Intelligence **139** (2002) 21–45
7. Minton, S., Johnston, M., Philips, A., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence **58** (1992) 161–205
8. Gonzalez, T., Sahni, S.: Open shop scheduling to minimize finish time. Journal of the ACM **23** (1976) 665–679
9. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64** (1993) 278–285
10. Brucker, P., Hurink, J., Jurich, B., Wostmann, B.: A branch-and-bound algorithm for the open-shop problem. Discrete Applied Mathematics **76** (1997) 43–59
11. Guéret, C., Prins, C.: A new lower bound for the open-shop problem. Annals of Operations Research **92** (1999) 165–183
12. Prins, C.: Competitive genetic algorithms for the open shop scheduling problem. Mathematical Methods of Operations Research **52** (2000) 389–411
13. Alcaide, D., Sicilia, J., Vigo, D.: A tabu search algorithm for the open shop problem. Trobajos de Investigación Operativa **5** (1997) 283–297
14. Liaw, C.F.: A tabu search algorithm for the open shop scheduling problem. Computers and Operations Research **26** (1999) 109–126
15. Guéret, C., Jussien, N., Prins, C.: Using intelligent backtracking to improve branch-and-bound methods: An application to open-shop problems. European Journal of Operational Research **127** (2000) 344–354