Generalizing Backdoors*

Roberto Rossi,[1] Steven Prestwich,[1] S. Armagan Tarim,[2] and Brahim Hnich[3]

Cork Constraint Computation Centre - CTVR, University College, Cork, Ireland[1]
{r.rossi ,s.prestwich}@4c.ucc.ie
Department of Management, Hacettepe University, Ankara, Turkey[2]
armagan.tarim@hacettepe.edu.tr
Faculty of Computer Science, Izmir University of Economics, Turkey[3]
brahim.hnich@ieu.edu.tr

Abstract. A powerful intuition in the design of search methods is that one wants to proactively select variables that simplify the problem instance as much as possible when these variables are assigned values. The notion of "Backdoor" variables follows this intuition. In this work we generalize Backdoors in such a way to allow more general classes of sub-solvers, both complete and heuristic. In order to do so, Pseudo-Backdoors and Heuristic-Backdoors are formally introduced and then applied firstly to a simple Multiple Knapsack Problem and secondly to a complex combinatorial optimization problem in the area of stochastic inventory control. Our preliminary computational experience shows the effectiveness of these approaches that are able to produce very low run times and — in the case of Heuristic-Backdoors — high quality solutions by employing very simple heuristic rules such as greedy local search strategies.

1 Introduction

A powerful intuition in the design of search methods is that one wants to proactively select variables that simplify the problem instance as much as possible when these variables are assigned values. The work presented in [22] follows this intuition. One of the main contributions in this work is the notion of "Backdoor" variables. This is a set of variables for which there is a value assignment such that the simplified problem can be solved by a poly-time algorithm called the "sub-solver". This works aims to generalize the notion of "Backdoor" variables by relaxing two assumptions that the sub-solver, as defined in [22], has to satisfy.

Firstly, we will relax the assumption stating that a sub-solver must be able to "determine" the solution of a problem in polynomial time. Nevertheless we will keep assuming that the sub-solver must reject in polynomial time an input for which it cannot determine feasibility or infeasibility. A simple example on a Multiple Knapsack Problem shows the effectiveness of this approach.

* Roberto Rossi is supported by Science Foundation Ireland under Grant No. 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain-Driven Research (CTVR) and Grant No. 05/IN/I886. S. Armagan Tarim and Brahim Hnich are supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. SOBAG-108K027.

Secondly, we will relax the assumption of completeness for the sub-solver. In this case the proposed framework will produce effective heuristic strategies for solving complex problems in a structured way inspired by a *divide et impera* logic. These strategies work as an aggregation of simple heuristic rules — such as greedy local search procedures — coordinated by a search algorithm such as depth-first search. Again we will apply this approach to the Multiple Knapsack Problem already discussed and we will show that in practice run times are improved without sacrificing too much the optimality of the solution obtained. This fact will also be supported by our computational experience on a complex combinatorial optimization problem in the area of stochastic inventory control.

The work is structured as follows. In Section 2 we provide the required formal background. In Section 3 we introduce the new concept of Backdoor-Condition. In Section 4 we introduce Pseudo-Backdoors and Heuristic-Backdoors. In Section 5 we discuss an application to a complex combinatorial optimization problem in the area of stochastic inventory control. In Section 6 we present related works from the literature. Finally, in Section 7 we draw conclusions.

2 Formal Background

A *Constraint Satisfaction Problem* (CSP) is a triple $\langle V, C, D\rangle$, where V is a set of decision variables, D is a function mapping each element of V to a domain of potential values, and C is a set of constraints stating allowed combinations of values for subsets of variables in V. A *solution* to a CSP is simply a set of values of the variables such that the values are in the domains of the variables and all of the constraints are satisfied. Often we want to find a solution to a CSP that is optimal with respect to certain criteria. Let $\mathcal{S}$ be the *solution set*, that is the set of all the tuples $(d_1, \ldots, d_k) \in D_1 \times D_2 \times \ldots \times D_k$ that are solutions to the CSP. A *Constraint Optimization Problem* (COP) is a CSP on the solution set of which an objective function, $f : \mathcal{S} \rightarrow \mathbb{R}$, has to be optimized. An *optimal solution* to a COP is a solution to the CSP that is optimal with respect to f.

2.1 Constraint Programming

We now recall some key concepts in *Constraint Programming* (CP): constraint filtering algorithm, constraint propagation and arc-consistency [17].

In CP a *filtering algorithm* is typically associated with every constraint. This algorithm removes values from the domains of the variables participating in the constraint that cannot belong to any solution of the CSP. These filtering algorithms are repeatedly called until no new deduction can be made. This process is called *propagation* mechanism. When no more domain reduction can be achieved by iterating the process, we say that each constraint, and the CSP, are *locally consistent* and that we have achieved a notion of *local consistency* on the constraints and the CSP. The term "local consistency" reflects the fact that the CSP obtained through the discussed process is not globally consistent. It is instead a CSP in which all the constraints are "locally", i.e. individually, consistent. A

comprehensive discussion on the process of constraint propagation is given by Apt [1]. If we demand that every domain value of every variable in the constraint belongs to a solution to the constraint then what we achieve is *hyper-arc consistency*, that is the strongest local consistency notion for a constraint. This still does not guarantee a solution to the whole CSP because other constraints in it are not considered in such a process.

In conjunction with this process CP uses a search procedure (like a backtracking algorithm) where filtering algorithms are systematically applied when the domain of a variable is modified.

In CP constraints that capture a relation among a non-fixed number of variables are called *global constraints* [17]. These constraints can be associated with powerful filtering algorithms. *Optimization-oriented global constraints* embed an optimization component, representing a proper relaxation of the constraint itself, into a global constraint [11]. This component provides three pieces of information: (a) the optimal solution of the relaxed problem; (b) the optimal value of this solution representing an upper bound on the original problem objective function; (c) a *gradient function* $\mathbf{grad}(V,v)$, which returns for each variable-value pair (V,v) an optimistic evaluation of the profit obtained if v is assigned to V. These pieces of information are exploited to perform *cost-based filtering* of suboptimal values in the domains and for guiding the search.

2.2 Local Search

A *neighborhood structure* is a function $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s. Without loss of generality, we here restrict the discussion to minimization problems. A *locally minimal solution* (or local minimum) with respect to a neighborhood structure $\mathcal{N}$ is a solution $\widehat{s}$ such that $\forall s \in \mathcal{N}(\widehat{s}) : f(\widehat{s}) \leq f(s)$. We call $\widehat{s}$ a strict locally minimal solution if $\forall s \in \mathcal{N}(\widehat{s}) : f(\widehat{s}) < f(s)$. Local search (LS) algorithms for COPs start from some initial solution and iteratively try to replace the current solution by a better solution in an appropriately defined neighborhood of the current solution. In this process it is extremely important to achieve a proper balance between *diversification* and *intensification* of the search. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. Among the most popular local search strategies we recall the *Iterative Improvement*, or *Hill Climbing*, in which each move is only performed if the resulting solution is better than the current solution and the algorithm stops as soon as it finds a local minimum. *Tabu Search* is a more advanced strategy, in fact it is among the most cited and used. Tabu search explicitly uses the history of the search, both to escape from local minima and to implement an explorative strategy. *Iterated Local Search* and *Variable Neighborhood Search* constitute other examples of local search strategies. For a comprehensive survey on local search and metaheuristic strategies the reader may refer to [5].

2.3 Hidden Structures: Backdoors

A powerful intuition in the design of search methods is that one wants to proactively select variables that simplify the problem instance as much as possible when these variables are assigned values. This intuition leads to the common heuristic of branching on the most constrained variable first.

In [22] the authors discuss a formal framework inspired by these techniques and present rigorous complexity results that support their effectiveness. One of the main contributions in this work is the notion of "Backdoor" variables. This is a set of variables for which there is a value assignment such that the simplified problem can be solved by a poly-time algorithm called the "sub-solver". In addition, the notion of "Strong Backdoor" is introduced, this is a set of variables for which any assignment leads to a poly-time solvable subproblem. Let us summarize which properties this algorithm should satisfy. A sub-solver A given as input a CSP, C: (i) either rejects the input C, or "determines" C correctly (as unsatisfiable or satisfiable), returning a solution if satisfiable; (ii) runs in polynomial time; (iii) can determine if C is trivially true (has no constraints) or trivially false (has a contradictory constraint); (iv) if A determines C, then for any variable x and value v, then A determines the simplified CSP where x is assigned to v.

We use the notation $C[v/x]$ to denote the simplified CSP obtained from a CSP, C, by setting the value of variable x to value v. Let $a_S : S \subseteq V \rightarrow D$ be a partial assignment. We use $C[a_S]$ to denote the simplified CSP obtained by setting the variables defined in a_S.

Definition 1 (Backdoor [22]). *A nonempty subset S of the variables is a Backdoor in C for A if for some $a_s : S \rightarrow D$, A returns a satisfying assignment of $C[a_S]$.*

Intuitively, the Backdoor corresponds to a set of variables, such that when set correctly, the sub-solver can solve the remaining problem. In a sense, the Backdoor is a "witness" to the satisfiability of the instance, given a sub-solver algorithm. The authors also introduced a stronger notion of the Backdoor to deal with both satisfiable and unsatisfiable (inconsistent) problem instances.

Definition 2 (Strong Backdoor [22]). *A nonempty subset S of the variables is a strong Backdoor in C for A if for all $a_s : S \rightarrow D$, A returns a satisfying assignment or concludes unsatisfiability of $C[a_S]$.*

A Strong Backdoor S is sufficient for solving a problem. This means if S is relatively small, one obtains a large improvement over searching the full space of variable/value assignments. Furthermore Backdoors can be exploited to dynamically switch the propagation logic and achieve a higher level of consistency during the search.

Example 1. Let us consider the following CSP=$\langle V, C, D\rangle$:
$V \equiv \{X_1, X_2, ..., X_m, N\}$,
$D \equiv \{X_1, X_2, ..., X_m, N \in \{1, \ldots, m\}\}$,

$C \equiv \{\texttt{NValue}([X_1, X_2, ..., X_m], N), N = m\}$.
We recall that the `NValue` constraint requires that variables $X_1, X_2, ..., X_m$ are assigned N different values. Propagating the `NValue` constraint is NP-hard [4] and thus its propagator, which we shall call P, does not achieve hyper-arc consistency since this would be computationally too expensive. Nevertheless it is clear that in the given CSP, once constraint $N = m$ is propagated, constraint $\texttt{NValue}([X_1, X_2, ..., X_m], N)$ becomes equivalent to $\texttt{allDiff}([X_1, X_2, ..., X_m])$, which requires the m decision variables to be all assigned different values from their domains and for which an efficient way of achieving hyper-arc consistency exists. Therefore, let A be the poly time algorithm that achieves hyper-arc consistency for `allDiff`, then $N \rightarrow m$ is a Backdoor with respect to A. In practice, this implies that we can create a new constraint `DNValue` (Dynamic NValue) able to switch the propagation logic from P to A as soon as the assignment $N \rightarrow m$ (Backdoor) is detected.

In this regard an interesting discussion is carried on in [3], where the parameterized complexity of global constraints is discussed.

3 Backdoor Conditions

We now define an important concept that will be employed throughout all the rest of this work. In the previous section we provided the formal definition for Backdoors and Strong Backdoors. We have also seen that a given sub-solver A must run in polynomial time and must reject (in polynomial time) the input if it is not able to either conclude satisfiability or unsatisfiability. In other words, the sub-solver must be able to check in polynomial time a "condition" that states if it can or it cannot solve the problem of assigning the remaining decision variables. We shall call this condition Backdoor Condition, formally defined as follows:

Definition 3 (Backdoor Condition). *Given a CSP, C, a* Backdoor Condition *with respect to a sub-solver A is a (global) constraint P on the subset $S \subseteq V$ of the decision variables in C that are currently instantiated, such that if the partial assignment $a_S : S \subseteq V \rightarrow D$ satisfies P, then a_S is a Backdoor in C for A. Determining if a_S satisfies P must be performed in polynomial time.*

Obviously we do not differentiate between Backdoor Condition and Strong Backdoor Condition, in fact it is easy to note that, if the set of variables S that are currently instantiated is a Strong Backdoor, P will be simply satisfied by any assignment. In contrast, if a specific partial assignment a_S is a Backdoor, then P will be satisfied by a_S but not by other partial assignments that are not Backdoors. Once the structure of the remaining subproblem has been revealed, dedicated strategies can be adopted to cope with it. The following section describes two possible directions.

4 Hidden Structures: Pseudo-Backdoors and Heuristic-Backdoors

Having an efficient (polynomial) algorithm for handling a subproblem that arises when some of the decision variables are fixed is indeed desirable. Nevertheless, often it may be the case that, after some decision variables have been fixed, the remaining subproblem is still NP-hard, but it has some additional structure that the original problem does not have. If this is the case, it is possible that specialized algorithms, such as dedicated propagators or heuristic procedures, may be able to exploit this additional structure in order to either achieve a stronger filtering or quickly produce promising or optimal assignments for all or some of the remaining decision variables.

4.1 Pseudo-Backdoors

If we relax assumption (ii) for A, we may accept sub-solvers having an exponential worst-case run time required to "determine" a solution for the CSP. Nevertheless the sub-solver should still be able to reject the input in polynomial time if satisfiability or unsatisfiability cannot be inferred. The key idea then is that, although a given sub-solver is not guaranteed to produce a solution in polynomial time, it should be able to produce competitive run times in practice. This intuition directly leads to the notion of Pseudo-Backdoor. We consider a sub-solver $\widehat{A}$ that is able to reject an input in polynomial time, but that may require exponential time to "determine" a solution for the CSP or to conclude unsatisfiability.

Definition 4 (Pseudo-Backdoor). *A nonempty subset S of the variables is a Pseudo-Backdoor in C for $\widehat{A}$ if for some $a_s : S \to D$, $\widehat{A}$ returns a satisfying assignment of $C[a_S]$ or concludes unsatisfiability of $C[a_S]$.*

Definition 5 (Strong Pseudo-Backdoor). *A nonempty subset S of the variables is a Strong Pseudo-Backdoors in C for $\widehat{A}$ if for all $a_s : S \to D$, $\widehat{A}$ returns a satisfying assignment or concludes unsatisfiability of $C[a_S]$.*

Particularly important is now the concept of Pseudo-Backdoor Condition:

Definition 6 (Pseudo-Backdoor Condition). *Given a CSP, C, a* Pseudo-Backdoor Condition *with respect to a sub-solver $\widehat{A}$ is a (global) constraint P on the subset $S \subseteq V$ of the decision variables in C that are currently instantiated, such that if the partial assignment $a_S : S \subseteq V \to D$ satisfies P, then a_S is a Pseudo-Backdoor in C for $\widehat{A}$. Determining if a_S satisfies P must be performed in polynomial time.*

Example 2. We consider a multiple knapsack problem with two bins into which objects can be fitted. A set of objects is given, for each object a profit and a weight are also given. Each bin is assigned a certain capacity. We want to fit as many objects as possible in the bins in such a way to maximize profit and

Items	KP-DFS	KP-DFS-DP
10	0.02	0.03
15	0.45	0.04
20	14	0.100
25	210	0.270

Table 1. Multiple Knapsack Problem. Comparison between the run times (in seconds) of a pure depth-first search strategy (KP-DFS) and of the hybrid depth-first/dynamic programming search strategy based on the Pseudo-Backdoor discussed (KP-DFS-DP).

to not exceed the capacity available for each bin. A simple observation directly leads to an effective Pseudo-Backdoor Condition. As soon as *the objects fitted in one of the two containers occupy enough capacity so that none of the remaining objects can be fitted in it*, the remaining problem is then to fit the unassigned objects to a "virtual bin" having a capacity equal to the residual capacity of the other bin. Once a given partial assignment a_S satisfies the Pseudo-Backdoor Condition described, the remaining problem is obviously a simple 0-1 Knapsack. If we employ a generic depth-first search strategy to solve the original problem, once reached the node corresponding to a_S, we can either continue exploring the search space using this strategy, or switch to a more efficient Dynamic Programming approach [15] for solving the remaining 0-1 Knapsack problem and returning the optimal solution with respect to the partial assignment a_S. Both the approaches have a worst case exponential run time, but in practice the dynamic programming approach will outperform a simple depth-first search by orders-of-magnitude. In Table 1 we compare the performance of a pure depth-first search strategy versus that of a hybrid depth-first/dynamic programming strategy based on the Pseudo-Backdoor described. All the experiments presented in this work have been performed on an Intel(R) Centrino(TM) CPU 1.50GHz with 2Gb RAM. The problem parameters are: bin capacities 500 and 1500; item profits and weights generated randomly and uniformly distributed respectively in [0,200] and [0,500]. We considered four instances comprising 10, 15, 20, and 25 items respectively. As the computational times witness, the hybrid approach scales particularly well.

4.2 Heuristic-Backdoors

Another requirement we could relax for a given sub-solver A is *completeness*. This means that the sub-solver may adopt a heuristic strategy.

In **CSPs** the former observation leads to the following approach:

- A solution method in which the sub-solver is used for heuristically produce a feasible assignment for some or all the remaining decision variables.

In **COPs** the former observation can lead to two different approaches:

– A complete solution method in which the heuristic sub-solver is used to generate a near-optimal solution that provides a good bound during the search. This approach is typically used in branch and bound algorithms [13].
– A heuristic solution method in which the heuristic sub-solver is used for assigning "promising" values to some or all the remaining decision variables.

More formally, a heuristic sub-solver $\tilde{A}$ given as input a CSP, C, either rejects the input C in polynomial time, or "may induce" a (partial) assignment on it; if $\tilde{A}$ "may induce" a (partial) assignment on C, then for any variable x and value v, then $\tilde{A}$ "may induce" a (partial) assignment on the simplified CSP where x is assigned to v. In order to clarify, "may induce" means that the sub-solver will actually induce an assignment if the heuristic strategy employed is able to produce such an assignment within the given time/runs limit, otherwise the sub-solver will simply reject the input. In fact, in contrast to a sub-solver A associated with a (Strong) Backdoor, $\tilde{A}$ is not required to run in polynomial time and does not necessarily have to return a feasible assignment.

Definition 7 (Heuristic-Backdoor). *A nonempty subset S of the variables is a Heuristic-Backdoor in C for $\tilde{A}$ if for some $a_s : S \rightarrow D$, $\tilde{A}$ may return a feasible assignment for $C[a_S]$.*

Definition 8 (Strong Heuristic-Backdoor). *A nonempty subset S of the variables is a Strong Heuristic-Backdoor in C for $\tilde{A}$ if for all $a_s : S \rightarrow D$, A may return a feasible assignment for $C[a_S]$.*

A (Strong) Heuristic-Backdoor S is not always sufficient for heuristically solving a problem. Nevertheless if S is relatively small and if the heuristic strategy is successful, one may obtain a large improvement over searching the full space of variable/value assignments.

Definition 9 (Heuristic-Backdoor Condition). *Given a CSP, C, a* Heuristic-Backdoor Condition *with respect to a heuristic sub-solver $\tilde{A}$ is a (global) constraint P on the subset $S \subseteq V$ of the decision variables in C that are currently instantiated, such that if the partial assignment $a_S : S \subseteq V \rightarrow D$ satisfies P, then a_S is a Heuristic-Backdoor in C for $\tilde{A}$. Determining if a_S satisfies P must be performed in polynomial time.*

(Strong) Heuristic-Backdoors are particularly suitable for developing structured ways of heuristically solving complex problems. In what follows we will show that using this novel concept it is possible to develop effective heuristic approaches to complex combinatorial optimization problems by employing very simple heuristic strategies, such as Hill Climbing procedures. The main reason for this is that, by using tree search, the original problem is split into much smaller problems. On these smaller problems simple heuristic rules such as iterative improvement often produce high quality assignments in almost no time.

Example 3. Let $\tilde{A}$ be a simple Greedy algorithm for solving 0-1 Knapsack problems [15]. In this algorithm objects are ordered by decreasing profit over weight.

Items	KP-DFS	KP-DFS-DP	KP-DFS-LS	% of real optimum
10	0.02	0.03	<0.001	100
15	0.45	0.04	<0.001	97.9
20	14	0.100	0.01	100
25	210	0.270	0.02	99.2

Table 2. Multiple Knapsack Problem. Comparison between the run times (in seconds) of a pure depth-first search strategy (KP-DFS), of the hybrid depth-first/dynamic programming search strategy based on the Pseudo-Backdoor discussed (KP-DFS-DP), and of the hybrid depth-first/local search strategy based on the Heuristic-Backdoor discussed (KP-DFS-LS). % of real optimum denotes the fraction (in percentage) of the optimum profit achieved by the heuristic approach.

Once ordered, objects are scanned sequentially and put into the knapsack if the residual capacity allows the insertion. This can be seen as a simple Hill Climbing strategy in which at each step we perform an "improving" move (insertion of an object in the bin) until a local maximum is achieved (no more objects can be fit in the bin). In the former Example 2 the Pseudo-Backdoor Condition described incidentally is also a Heuristic-Backdoor Codition with respect to this Greedy algorithm $\tilde{A}$. Thus as soon as this condition is met by a given partial assignment a_S the remaining subproblem can be solved in a heuristic way by using $\tilde{A}$. In Table 2 we refer to the same test bed discussed in Example 2, but now we complete the overall picture by showing the performance of the new heuristic approach based on the greedy algorithm discussed (KP-DFS-LS). The run times and the discrepancies from the real optima clearly show that the heuristic approach is able to speed up the search without sacrificing too much in terms of optimality.

5 An Application to Stochastic Inventory Control

In this section we will provide a high level description of an application for Heuristic-Backdoors to a complex combinatorial optimization problem in inventory control. We firstly provide a description of the problem; secondly we propose a possible heuristic strategy $\tilde{A}$; thirdly we identify a valid Heuristic-Backdoor Condition, that is a valid constraint that a given partial assignment for decision variables has to satisfy in order to be a Heuristic-Backdoor with respect to the given heuristic strategy $\tilde{A}$; and finally the overall architecture combining the CP and the LS algorithms is presented and computational results are provided for a set of instances.

Notation and terminology We shall now introduce some important terminology concerning inventory control policies. In inventory control, when demand is probabilistic, it is useful to conceptually categorize inventories as follows:

- *On-hand stock:* This is stock that is physically on the shelf; it can never be negative. This quantity is relevant in determining whether a particular customer demand is satisfied directly from the shelf
- *Backorders:* These denote an existing demand that cannot be fulfilled since no stock is available on the shelf
- *On order:* These are stocks which have been ordered, but that for some reason have not reached the shelf yet. Reasons for this may comprise: stock inspection, transportation etc.
- *Net inventory* = (On hand) - (Backorders)

  This quantity can become negative (namely, if there are backorders). It is used in some mathematical derivations and is also a component of the following important definition
- *Inventory position:* The inventory position is defined by the relation

  Inventory position = (On hand) + (On order) - (Backorders).

5.1 Problem Description

We consider a finite planning horizon of N periods and a demand d_t for each period $t \in \{1, \ldots, N\}$, which is a random variable with probability density function $g_t(d_t)$. We assume that the demand occurs instantaneously at the beginning of each time period. The demand we consider is non-stationary, that is it can vary from period to period, and we also assume that demands in different periods are independent. A stochastic lead time l_t with probability mass function $f_t(l_t)$ for an order placed in period $t \in \{1, ..., N\}$ is given. Note that $\{l_t\}$ are mutually independent and each of them is also independent of the respective order quantity. A fixed delivery cost a is incurred for each order. A linear holding cost h is incurred for each unit of product carried in stock from one period to the next. We assume that it is not possible to sell back excess items to the vendor at the end of a period and that negative orders are not allowed, so that if the actual stock exceeds the order-up-to-level for that review, this excess stock is carried forward and not returned to the supply source. However, such occurrences are regarded as rare events and accordingly the cost of carrying excess stocks and the positive effect on the service level of subsequent periods is ignored. As a service level constraint we require the probability that at the end of each and every period the net inventory will not be negative set to be at least a given value α. Our aim is to minimize the expected total cost, which is composed of ordering costs, unit costs and holding costs, over the N-period planning horizon, satisfying the service level constraints.

A chance-constrained programming formulation for this problem is given in [18]. This formulation can be tailored to incorporate the inventory control policy adopted. For instance we may adopt the "replenishment cycle policy", which is equivalent to Bookbinder-Tan's "static-dynamic uncertainty strategy" [6]. The replenishment cycle policy is *static* in the sense that the replenishment periods are determined once and for all at the beginning of the planning horizon, and

dynamic as the order quantities are decided only after observing the realized demand in the periods before a given replenishment.

Under the replenishment cycle policy the problem is then to decide the optimal order periods, which we may denote using binary decision variables $\delta_1, \ldots, \delta_N$, where $\delta_i = 1$ iif an order is scheduled in period $i \in 1, \ldots, N$, and the respective optimal "order-up-to-positions" that is the level to which the inventory position has to be raised by each order.

As shown in [18] a complex expression involving a number of ordering decisions and order-up-to-positions must be evaluated[1] in order to compute the service level α provided at a given period i. Specifically, let a *replenishment cycle* be the time span between two consecutive replenishments, the relation to compute the service level at a given period i will involve the minimum set of former consecutive replenishment cycles spanning over $L+1$ periods, where L is the maximum possible lead time length (Fig. 1). We will not discuss this expression in details and we will assume that this is given as a black box.

5.2 A Simple Hill Climbing Procedure

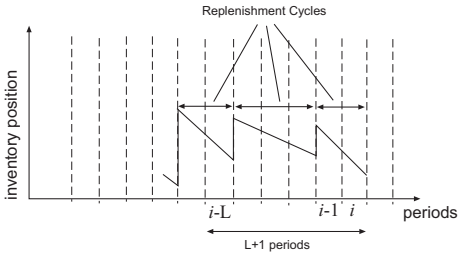

Fig. 1. Replenishment Cycles corresponding to the following partial assignment for replenishment decisions: $\delta_{i-L-1} = 1$, $\delta_{i-L} = 0$, $\delta_{i-L+1} = 1$, $\delta_{i-L+2} = 0$, $\delta_{i-L+3} = 0$, $\delta_{i-1} = 1$, $\delta_i = 0$. Since at least L periods before period i are covered by this set of consecutive cycles it is possible to determine the service level at period i.

Assume that a set of consecutive replenishment cycles is given, covering $L + 1$ or more periods. This can be seen as a partial assignment for decision variables δ_i. If no order-up-to-position is associated with each of the replenishment cycles identified, in order to compute "good" order-up-to-positions such that the service level provided at the end of the last replenishment cycle is at least α we may adopt the following greedy approach.

We initially associate with each replenishment cycle an order-up-to-position of 0. The service level provided will be in this case extremely low. We now start increasing of one unit the *most promising* order-up-to-position, that is the one that, once increased, produces the maximum increase with respect to the service level provided at the end of the last replenishment cycle. We proceed increasing at each step the most promising order-up-to-position for the replenishment cycles of interest until the service level provided at the end of the last replenishment cycle is at least α.

Example 4. Consider again the example in Fig. 1. Assume that x is the current order-up-to-level for the cycle covering periods $i - L - 1$ and $i - L$, y is the current order-up-to-level for the cycle covering periods $i - L + 1$, $i - L + 2$ and $i - L + 3$, finally z is the order-up-to-level for the cycle covering periods $i - 1$

[1] Note that the evaluation requires polynomial time

and i. Initially $x = 0, y = 0, z = 0$. The complex expression discussed above to compute the service level at period i can be used since at least $L + 1$ periods are covered. Assume that, after a number of steps performed by the greedy approach discussed, this expression states that the current set of order-up-to-levels provides a service level of $\alpha - 0.05$. Using the same expression we also state that increasing x by one unit provides a service level of $\alpha + 0.01$, increasing y by one unit provides a service level of $\alpha - 0.01$ and increasing z by one unit provides a service level of $\alpha - 0.02$. Thus we choose the first option, we increase x by one unit and we obtain an assignment for order-up-to-levels that satisfies the required service level constraint at period i.

5.3 A Heuristic-Backdoor Condition

In our problem it is easy to see that a Heuristic-Backdoor with respect to the greedy procedure just presented is a partial assignment for replenishment decisions, δ_t, $t = 1, \ldots, N$, such that for a given period i a sufficient number of former replenishment cycles are identified — that is the replenishment cycles identified cover at least periods from $i - L$ to i. This provides a simple Heuristic-Backdoor Condition that can be checked by the heuristic sub-solver in polynomial time.

Once a Heuristic-Backdoor has been identified by the sub-solver through the described Heuristic-Backdoor Condition, it is possible to apply the greedy approach in order to quickly find a good assignment for the respective order-up-to-positions without resorting to a complete depth-first search procedure.

5.4 A Hybrid CP/LS Algorithm

We developed a CP model, following the structure of the CP approach in [18], and we incorporated the heuristic strategy discussed in the form of a global constraint. The model is quite simple. Let $\tilde{x}$ denote the expectation of a random variable x.

$$\min \quad E\{TC\} = \sum_{t=1}^{N} \left(a \cdot \delta_t + h \cdot \tilde{P}_t \right) \tag{1}$$

subject to,

$$\tilde{P}_t + \tilde{d}_t - \tilde{P}_{t-1} > 0 \Rightarrow \delta_t = 1 \qquad t = 1, \ldots, N \tag{2}$$

$$\delta_t = 0 \Rightarrow \tilde{P}_t + \tilde{d}_t - \tilde{P}_{t-1} = 0 \qquad t = 1, \ldots, N \tag{3}$$

$$\tilde{P}_t + \tilde{d}_t - \tilde{P}_{t-1} \geq 0 \qquad t = 1, \ldots, N \tag{4}$$

$$\begin{aligned} &\texttt{serviceLevel}(\delta_1, \ldots, \delta_N, \\ &\qquad \tilde{P}_1, \ldots, \tilde{P}_N, \\ &\qquad g_1(d_1), \ldots, g_N(d_N), \\ &\qquad f(\cdot), \alpha) \end{aligned} \tag{5}$$

$$\tilde{P}_t \geq 0, \quad \delta_t \in \{0, 1\} \qquad t = 1, \ldots, N. \tag{6}$$

Period (t)	1	2	3	4	5	6	7	8
$\tilde{d}_t$	15	18	13	33	30	18	23	15

Table 3. Forecasts of period demands.

The objective function minimizes ordering (a) and holding (h) cost with respect to the expected inventory position ($\tilde{P}_t$) at the end of each period. The first three constraints states the replenishment condition, the non-replenishment condition and the inventory conservation rule. Constraint `serviceLevel`(...) dynamically decides, in polynomial time, if a given partial assignment is a Heuristic-Backdoor — i.e. if sufficient consecutive periods are covered so that order-up-to-levels can be generated — and in such a case it fixes lower bounds for the corresponding expected inventory positions by using the greedy approach described above.[2]

5.5 Cost-Based Filtering

In order to speed-up the search process we adopted the cost-based filtering strategy for stochastic COPs discussed in [19]. In this work the authors propose an expectation-based relaxation for part or all the random variables in the problem. This relaxation produces a valid bound under some assumptions. Specifically, we considered the simplified problem where the delivery lead-time is replaced by its expected value. It can be proved that the model considered satisfies the assumptions in [19]. As discussed in [18, 20] the resulting relaxed problem can be solved very efficiently using the CP approach in [21]. Therefore we adopted this relaxation in order to produce good bounds during the search process that are exploited for filtering parts of the search tree that cannot lead to a better solution.

5.6 Computational Experience

In this section we consider the same set of instances analyzed in [18]. Specifically, we focus on the last two instances there discussed, in which a stochastic lead time of 2 periods over an 8-period planning horizon is considered.

We assume an initial null inventory level and a normally distributed demand with a coefficient of variation $\sigma_t/\tilde{d}_t = 0.3$ for each period $t \in \{1, \ldots, 8\}$. The expected values $\{\tilde{d}_t\}$ for the demand in each period are listed in Table 3. The other parameters are $a = 30$, $h = 1$, $\alpha = 0.95$.

Firstly, we analyze a stochastic lead time with probability mass function $f_i(t) = \{0.2(0), 0.6(1), 0.2(2)\}$. That is an order is received immediately with probability 0.2, after one period with probability 0.6, and after two periods with

[2] Note that the minimum expected inventory position required to provide a service level α at the end of a given period can be directly computed from the order-up-to-position at the beginning of the cycle covering the period by subtracting the expected demand from the beginning of the cycle and up to that period.

$E\{TC\}$: 532								
Period (t)	1	2	3	4	5	6	7	8
R_t	50	72	101	88	79	72	54	31
δ_t	1	1	1	0	1	1	0	0
Shortage probability	–	–	5%	5%	3%	5%	5%	5%

Table 4. Optimal policy under stochastic lead time, $f_i(t) = \{0.2(0), 0.6(1), 0.2(2)\}$, in periods $\{1, 2\}$ the inventory cannot be controlled. R_t is the opening-inventory-position at period t, that is $R_t = \tilde{P}_t + \tilde{d}_t$.

probability 0.2. The optimal solution — obtained using the approach discussed in [18] — is presented in Fig. 2, details about the optimal policy are reported in Table 4. The number of replenishment cycles is 5. The policy parameters are:

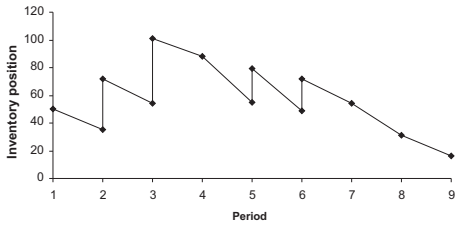

Fig. 2. Optimal policy under stochastic lead time, $f_i(t) = \{0.2(0), 0.6(1), 0.2(2)\}$.

cycle lengths= $[1, 1, 2, 1, 3]$ and order-up-to-positions= $[50, 72, 101, 79, 72]$.

Secondly, we consider a different probability mass function for the lead time: $f_i(t) = \{0.5(0), 0.0(1), 0.5(2)\}$, which means that we maintain the same average lead time of one period, but we increase its variance. The optimal solution is presented in Fig. 3, details about the optimal policy are reported in Table 5. The number of replenishment cycles is still 5, policy parameters are: cycle

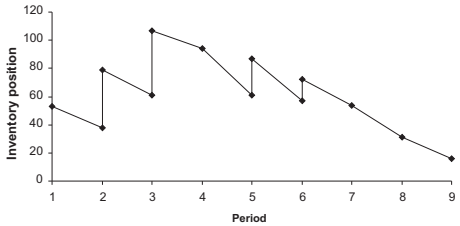

Fig. 3. Optimal policy under stochastic lead time, $f_i(t) = \{0.5(0), 0.0(1), 0.5(2)\}$.

lengths= $[1, 1, 2, 1, 3]$ and order-up-to-positions= $[50, 72, 101, 79, 72]$. Although

$E\{TC\}$: 562								
Period (t)	1	2	3	4	5	6	7	8
R_t	53	79	107	94	87	72	54	31
δ_t	1	1	1	0	1	1	0	0
Shortage probability	–	–	5%	5%	0%	5%	5%	5%

Table 5. Optimal policy under stochastic lead time, $f_i(t) = \{0.5(0), 0.0(1), 0.5(2)\}$. R_t is the opening-inventory-position at period t, that is $R_t = \tilde{P}_t + \tilde{d}_t$.

the average lead time is still one period, order-up-to-positions are slightly higher than in the former case where the variance of the lead time was lower. Also the cost reflects this, in fact it is 5.6% higher than in the former case.

For both the instances discussed our heuristic approach achieves a considerable improvement in terms or run time without sacrificing the quality of the solution obtained.

The heuristic used for the selection of the variable is the usual min-domain/max-degree heuristic. Decision variables have different priorities in the heuristic: the δ_k have higher priority than the $\tilde{P}_k$. The value selection heuristic chooses values from the smallest to the largest.

For both the two instances presented the exact and the heuristic approaches, in conjunction with the discussed cost-based filtering strategy, produce the same solution, but while the exact approach in [18] requires more than one hour to prove optimality for both the first and the second instance, our heuristic approach only requires respectively 5 and 6 seconds to complete the search process. Also in this case the experiments have been performed on an Intel(R) Centrino(TM) CPU 1.50GHz with 2Gb RAM.

6 Related Works

The concept of Backdoors has been originally introduced in [22]. Since then, much of the work on Backdoors has been focused on SAT problems, see for instance [14]. In [7] the authors propose an explanation-based approach exploiting Backdoors for dynamically identifying and exploiting structures in CSPs. Nevertheless, to the best of our knowledge, in the literature Bakdoors have not been used so far for switching the search strategy either to a complete or incomplete different strategy not necessarily polynomial (such as Dynamic Programming).

The integration of Operations Research and Constraint Programming techniques for combinatorial optimization is a very active research field [12]; but operations research techniques are typically employed for generating valid relaxations used for performing domain filtering and, with the exception of Bender's Decomposition in [7], they are not employed as alternative search strategies that can take over the control of the search process when a given condition is met.

Finally, the integration between Constraint Programming and Local Search has been discussed in a variety of works — the reader may refer to the review

in [10] — and it is an active research field [9]. Several ways of blending constraints with local search procedures have been discussed in the literature [2, 16]. In all these approaches the local search engine is used to "guide" the search, while Constraint Programming is used for exploring promising neighborhood. Alternatively, local search techniques can be introduced within a constructive global search algorithm [8]. The technique we propose is of this second kind, but the notion of Heuristic-Backdoor makes our approach novel and more general compared to other specialized approaches presented in the literature.

7 Conclusions

We generalized Backdoors in such a way to allow sub-solvers that do not run in polynomial time. This led to Pseudo-Backdoors and to Heuristic-Backdoors, that let us switching the search logic (or the propagation logic of a given global constraint) as soon as a known structure in the remaining subproblem that has to be solved is revealed by a given partial assignment. While Pseudo-Backdoors guarantee completeness in the search process, this is not the case when Heuristic-Backdoors are employed. Heuristic-Backdoors in fact let the sub-solver employ a heuristic strategy to produce a solution. We applied both Pseudo-Backdoors and Heuristic-Backdoors to a simple Multiple Knapsack Problem taken as running example. We have also shown the effectiveness of Heuristic-Backdoors on a complex combinatorial optimization problem. In future works we aim to extend our computational experiments, that so far only present preliminary results, and we aim to provide a detailed description of the case study in Section 5, for which only a very high level description has been provided in this work.

References

1. K. Apt. *Principles of Constraint Programming*. Cambridge University Press, Cambridge, UK, 2003.
2. B. De Backer, V. Furnon, P. Shaw, P. Kilby, and P. Prosser. Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, 6(4):501–523, 2000.
3. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, C.-G. Quimper, and T. Walsh. The parameterized complexity of global constraints. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2008.
4. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Filtering algorithms for the nvalue constraint. *Constraints*, 11(4):271–293, 2006.
5. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.
6. J. H. Bookbinder and J. Y. Tan. Strategies for the probabilistic lot-sizing problem with service-level constraints. *Management Science*, 34:1096–1108, 1988.
7. H. Cambazard and N. Jussien. Identifying and exploiting problem structures using explanation-based constraint programming. *Constraints*, 11(4):295–313, 2006.
8. A. Cesta, G. Cortellessa, A. Oddi, N. Policella, and A. Susi. A constraint-based architecture for flexible support to activity scheduling. *Lecture Notes in Computer Science*, 2175:369+, 2001.

9. B. Crawford, C. Castro, and E. Monfroy. Integration of constraint programming and metaheuristics. In Ian Miguel and Wheeler Ruml, editors, *Abstraction, Reformulation, and Approximation, 7th International Symposium, SARA 2007, Whistler, Canada, July 18-21, 2007, Proceedings*, volume 4612 of *Lecture Notes in Computer Science*, pages 397–398. Springer, 2007.
10. F. Focacci, F. Laburthe, and A. Lodi. Local Search and Constraint Programming. *In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science. Kluwer Academic Publishers, Norwell, MA*, 2002.
11. F. Focacci, A. Lodi, and M. Milano. Optimization-oriented global constraints. *Constraints*, 7:351–365, 2002.
12. F. Focacci and M. Milano. Connections and integrations of dynamic programming and constraint programming. In *Proceedings of the International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems CP-AI-OR 2001*, 2001.
13. E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.
14. I. Lynce and J. Marques-Silva. Hidden structure in unsatisfiable random 3-sat: An empirical study. In *ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 246–251, Washington, DC, USA, 2004. IEEE Computer Society.
15. S. Martello and P. Toth. *Knapsack Problems*. John Wiley & Sons, NY, 1990.
16. G. Pesant and M. Gendreau. A view of local search in constraint programming. In Eugene C. Freuder, editor, *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Cambridge, Massachusetts, USA, August 19-22, 1996*, volume 1118 of *Lecture Notes in Computer Science*, pages 353–366. Springer, 1996.
17. J.-C Regin. *Global Constraints and Filtering Algorithms*. in Constraints and Integer Programming Combined, Kluwer, M. Milano editor, 2003.
18. R. Rossi, S. A. Tarim, B. Hnich, and S. Prestwich. Computing replenishment cycle policy under non-stationary stochastic lead time. Technical report, Cork Constraint Computation Centre, 2008.
19. R. Rossi, S. A. Tarim, B. Hnich, and S. Prestwich. Cost-based domain filtering for stochastic constraint programming. In *Proceedings of the 14th International Conference on the Principles and Practice of Constraint Programming*, pages 235–250. Springer Verlag, 2008. Lecture Notes in Computer Science No. 5202.
20. S. A. Tarim. *Dynamic Lotsizing Models for Stochastic Demand in Single and Multi-Echelon Inventory Systems*. PhD thesis, Lancaster University, 1996.
21. S. A. Tarim, B. Hnich, R. Rossi, and S. Prestwich. Cost-based filtering techniques for stochastic inventory control under service level constraints. *Constraints*, 2008 (forthcoming).
22. R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1173–1178. Morgan Kaufmann, 2003.