Generalized Solution Techniques for Preference-Based Constrained Optimization with CP-nets

James C. Boerkoel Jr. Edmund H. Durfee Keith Purrington
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109 USA
{boerkoel,durfee,purringk}@umich.edu

ABSTRACT
Computational agents can assist people by guiding their decisions in ways that achieve their goals while also adhering to constraints on their actions. Because some domains are more naturally modeled by representing preferences and constraints separately, we seek to develop efficient techniques for solving such decoupled constrained optimization problems. This paper describes a parameterized formulation for decoupled constrained optimization problems that subsumes the state-of-the-art algorithm of Boutilier *et al.*, representing a wider family of alternative algorithms. We empirically examine notable members of this family to highlight the spaces of decoupled constrained optimization problems for which each excels, highlight fundamental relationships between different algorithmic variations, and use these insights to create and evaluate novel hybrids of these algorithms that a cognitive assistant agent can use to flexibly trade off solution quality with computational time.

Categories and Subject Descriptors
G.1.6 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms
Theory, Algorithms, Performance

Keywords
Decoupled Constrained Optimization Problems, CP-nets

1. INTRODUCTION
Semi-autonomous agents can assist users with cognitive tasks like configuring products for an online shopper, scheduling meetings for a busy executive, or helping cognitively impaired individuals enjoy a greater degree of independence [6]. Such agents solve constrained optimization problems, balancing their users' desires with hard, externally-imposed constraints. In particular, a large number of outcomes (complete assignments to the problem variables) might be feasible within a problem's hard constraints, so the task of the cognitive-support agent is also to help find an outcome that is (among) the best.

Such constrained optimization problems can be broadly divided into two categories—those where the preference and constraint representation are coupled and those where the representations remain decoupled. Coupled approaches with quantitative preferences generally model preferences as ―soft" constraints, and have included approaches such as weighted CSPs [1], fuzzy CSPs [8] and temporal CSPs [5]. Similarly, Max-CSP (maximizing the number of satisfied constraints) is a common qualitative, coupled representation.

Decoupling has the advantage of saving the user the burden of expressing what is possible, allowing her instead to describe only what she wants. Product configuration offers one example of this: a vendor knows what products can be built, while the shopper knows what he or she wants to buy. Similarly, people with cognitive deficits might have clear preferences about their daily activities, but rely on others (such as caregivers) to keep track of constraints (such as medication regimens or television schedules).

Previous work by Boutilier *et al.* [3] has described a decoupled approach that models users' preferences compactly and qualitatively using CP-nets. Introduced more formally in Section 2, CP-nets are attractive because of their relative ease of elicitation [4]. In addition to describing the representation, Boutilier *et al.* specified a particular algorithm for solving the constrained optimization problem, proved its correctness, and demonstrated that, in some sense, the optimization problem is no harder than finding any solution to the underlying CSP.

This paper's contributions are (1) to generalize those prior insights into a versatile, parameterized family of algorithms for solving decoupled constrained optimization problems, (2) to evaluate the strengths and limitations of notable members of that family, and (3) to utilize those results to create novel hybrid techniques that can flexibly trade off solution quality and time. We begin (Section 2) by summarizing the separate qualitative preference optimization and constraint satisfaction problems and their solution techniques, and illustrate an example application that involves solving both problems simultaneously. In Section 3, we describe our parameterization of a basic CSP search framework to encompass a family of algorithms that includes Boutilier *et al.*'s specific case, and we evaluate, both empirically and analytically, the solution methods that arise at the parameter boundaries, as well as Boutilier *et al.*'s approach and other variants that the parameterization supports. We then, in Section 4, show how our more generalized framework can easily support a hybridization of techniques to achieve boundedly-approximate optimization faster, as well as ―anytime" behavior. We summarize our results and describe future directions in Section 5.

Cite as: Generalized Solution Techniques for Prefence-Based Constrained Optimization with CP-nets, Boerkoel, Durfee and Purrington, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 291-298

2. BACKGROUND

We briefly introduce a simple constrained optimization problem that we will use to illustrate key concepts, and then summarize the CP-net formalism and the basic CSP search formulation that comprise the components of the decoupled constrained optimization problem.

2.1 A Simple Example

Consider the example problem in Figure 1 (left). Ann has three different objectives to complete between 8am and noon: exercise, get an errand done, and recreate with friends. Ann may have one or more choices for how to meet each objective. For example, she could play cards or scrap-book for recreation, swim or bike for exercise, and go to the bank or the store for her errand. She cannot perform two activities at the same time, and each activity has its own duration and may need to occur during restricted intervals of time. For example, going to the bank is a short errand but must happen during ~~b~~anker's hours" while going to the store takes longer but the store never closes. Similarly, swimming tires her out faster than biking, but the pool hours are more restrictive. Thus, only some combinations of activities to meet her objectives will be schedulable. Finding a feasible combination involves solving a constraint satisfaction problem.

Ann also has preferences about how she accomplishes each of her objectives. For example, all else being equal, Ann prefers biking to swimming. If Ann bikes, she prefers to be on her feet less time by going to the bank, but if she swims then she prefers going to the store. Finally, Ann prefers to scrap-book if she visited the store (for supplies). Notice that Ann's preferences over some choices are conditional on what other choices have been made. Capturing such conditional preferences is what CP-nets do well.

2.2 CP-Nets

Briefly, CP-nets [2] are a compact, graphical representation of a user's qualitative *ceteris paribus* preferences, very similar in structure to Bayes-nets. Variables are arranged in a directed graph, and each variable is accompanied by a conditional preference table (CPT) that specifies its associated preference function, conditioned on the values of its parent variables. Ann's preferences from Section 2.1 are captured in the CP-net shown in Figure 1 (left). Further, (acyclic) CP-nets induce partial ordering over all outcomes; the induced partial order for the CP-net shown in Figure 1 (left) is shown in Figure 1 (right), where an arrow from outcome *O1* to *O2* indicates *O1* is strictly preferred to (dominates) *O2*. Since preference is transitive, the outcomes in Figure 1 (right) go from most preferred (at the top) to least preferred (at the bottom). The pair of outcomes <Bike, Store, Cards> and <Swim, Store, SBook> are incomparable since neither dominates the other.

Note that the number of outcomes is combinatorial in the number of CP-net nodes. Thus, while queries about outcomes can easily be carried out in the induced outcome graph, some outcome queries can be answered much more efficiently using the exponentially smaller CP-net. For example, preferential optimization is simple and efficient: the optimal outcome is found by assigning each CP-net variable (in topological order) to its most preferred value given its parents, with no need to backtrack. (This is called the forward-sweep algorithm [2].) In Figure 1 (left), the optimal outcome is thus Bike (unconditionally), Bank (given Bike), and Cards (given Bank), as confirmed at the top of Figure 1 (right). If a variable has an exogenously assigned value, then that value is assigned when the variable is reached. For example, if Ann's bike has a flat tire, her most preferred outcome under this constraint is <Swim, Store, SBook>, as again can be confirmed in Figure 1 (right).

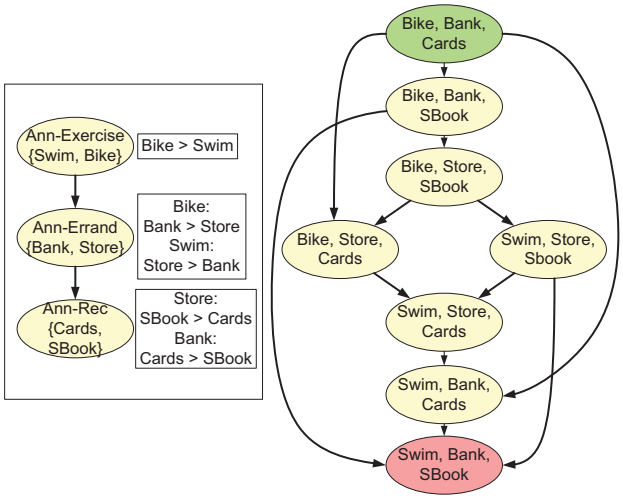

Figure 1. An example CP-net (left) along with its induced preference graph (right).

This paper also exploits the efficiency of an ordering query: if *O1* is orderable over *O2*, it means that *O2* cannot be strictly preferred to *O1* based on the information in the CP-net. We equivalently say that *O1* is non-dominated by *O2*. The algorithm for deciding ordering queries compares outcomes *O1* and *O2* on the basis of the first variable *V* (in some topological order that respects the CP-net partial order) that has a different value in *O1* and *O2*. The outcome in which the value of *V* is more preferred is orderable over the other. For some pairs of outcomes, it is possible to determine that *O1* is orderable over *O2* for some topological order and *O2* over *O1* for a different order. More generally, orderability is only determinable in one direction. In the running example, if *O1* is <Bike, Store, Cards> and *O2* is <Swim, Store, SBook>, then the ordering query only confirms that *O1* is orderable over *O2* and not the other way around.

As we shall see, if all we care about is finding some feasible outcome that we are sure is not dominated by any other feasible outcome, then *ordering queries suffice.* That is, for a particular constrained optimization problem, there might be a sizable set of outcomes *O** that are non-dominated by any other feasible outcomes. For this paper, as in the earlier work of Boutilier *et al.*, we consider any outcome *O* in *O** to be a valid solution to the constrained optimization problem.

2.3 Constraint Satisfaction Problems

Like a CP-net, a constraint satisfaction problem (CSP) is defined in terms of variables and their possible values. Unlike CP-nets where any combination of assignments (outcome) is possible though not necessarily preferable, CSPs impose constraints on the assignments of values to variables such that assignments (outcomes) that violate constraints are impermissible (infeasible) as solutions. We have already seen a simple example of this, where the CSP imposes a constraint that Ann-Exercise cannot be assigned ~~b~~ike" (because of the flat tire), essentially removing from Figure 1 (right) all outcomes that involve ~~B~~ike", and leading to a different preferred outcome than if the constraint did not exist.

Constraints can arise for a variety of reasons. In our running example, for instance, the constraints can involve timing (e.g., the bank's hours, the amount of time needed to shop) or other kinds of dependencies (e.g., scrapbooking depends on having gone to the store). While there are some interesting challenges in solving problems with temporal and non-temporal constraints [7], for this paper we will simply assume that there are some combinations of variables' values that are forbidden. Thus, solving a CSP involves searching through possible variable assignments, pruning assignments that violate constraints, to find one or more assignments satisfying all constraints.

More generally, the canonical technique for solving CSPs alternates between 1) propagating the effects of constraints to prune the search space and 2) making tentative assignments to one or more variables once all such propagation has been exhausted. If it reaches a state where all variables have been assigned, then it returns that solution. On the other hand, if the search space becomes empty, the process backtracks to some previous tentative assignment(s), and tries something else. If it ever gets to the point where it has backtracked to and exhaustively tried all of the possible tentative assignments, it terminates and indicates that the problem is unsatisfiable.

Of course, how efficiently the search finds a solution can depend on making good tentative assignments. Typical CSP heuristics traditionally attempt to assign variables in ways that will have the best chance of leading to a complete satisfying assignment and, when unsuccessful, will prune the greatest amount of the search space. As we shall see, if the problem is not only to find a satisfying solution, but in fact to find a *most-preferred* satisfying solution, then different heuristic decisions might prove more suitable.

3. Decoupled Constrained Optimization

Constrained optimization focuses on the problem of finding the ~~"b~~best" satisfying solution to a CSP, for CSPs that have multiple satisfying solutions. In our running example, the constraint that Ann-Exercise cannot be ~~"B~~ike" still leaves 4 possible outcomes; we want an algorithm that (efficiently) finds the best of them. While in this simple case we could do so with the CP-net preferential optimization algorithm, more generally this will not suffice because constraints can apply to combinations of variables.

As mentioned in Section 1, one way of solving constrained optimization problems is to treat preferences as ~~"s~~oft" or ~~"w~~eighted" constraints. As a simple example, given the preferential ordering shown in Figure 1 (right), we could add to the CSP a set of weighted constraints, such as a constraint that the complete assignment be <Bike, Bank, Cards> with weight 7, that it be <Bike, Bank, SBook> with weight 6, and so on down the graph. Then, using any of several algorithms (for example [1]), this weighted CSP (where the hard constraints have infinite weights) could be solved for the assignment that minimizes the summed weights of violated constraints. Since in our example case only one of these weighted constraints can be satisfied, the solution returned must be the most preferred complete assignment.

This simple example involves introducing exponentially (in the number of variables) many weighted constraints. Typically, coupled approaches exploit factored preferences, where the strength of preference over the assignment of some subset of variables is not conditioned on the assignments to other subsets. However, as this example illustrates, when preferences are strongly conditional (as in the example CP-net), a coupled approach to constraint optimization can become unwieldy.

For this reason, as well as our previously-stated motivations (e.g., ease of preference elicitation, separation of knowledge that might stem from different places), we pursue a decoupled paradigm combining CP-nets and CSPs. In their groundbreaking work on this paradigm, Boutilier *et al.* showed that performing a CSP search, but assigning variables with respect to the CP-net structure instead of using standard CSP heuristics, can find all non-dominated (hence optimal with respect to the CP-net) variable assignments [3]. The kinds of problems motivating our work are similar, but we will be satisfied with finding just one non-dominated feasible solution, since the CP-net provides no basis for preferring any such solution over another.

Intuitively, CP-net-centric variable/value assignments will guide search toward preferred solutions, but not necessarily toward feasible solutions, while using CSP-centric heuristics will do the opposite. Which of these will most quickly lead to finding a non-dominated feasible solution (and knowing that it is non-dominated) depends on the sizes of the spaces of feasible and of non-dominated assignments. Therefore, we now turn to introducing and evaluating a more flexible, parameterized formulation that supports a family of possible heuristic methods for solving the decoupled optimization problem. Our key focus in solving general decoupled constrained optimization problems is on how to blend reasoning about optimization with reasoning about feasibility.

3.1 A Parameterized Formulation

Our more general formulation for decoupled constrained optimization involves three parameters; we introduce a fourth, *t*, (for satisficing) later. The first is a step-size parameter, *s*, which specifies how many variables the algorithm assigns between propagation steps. The second parameter, *h*, determines which variable assignment strategy is used to make those *s* assignments. As we have already indicated, we consider assigning variables in one of two ways: 1) using the CP-net to determine variable ordering and the most preferred values (*h*=CP-net), and 2) using the domain-independent CSP-based heuristic dom/deg, a minimum-remaining-values-style heuristic (*h*=MRV), which selects the variable with fewest remaining values in its domain, normalized by the degree of the variable. In this second method, the chosen variable's value is subsequently selected to minimize conflicts with other variables' domains. To break any remaining ties, the MRV heuristic uses the CP-net heuristic, and vice versa. Finally, the third parameter, *all*, is a Boolean indicating whether all solutions (or just the first) should be returned.

The workhorse algorithm, SolveCSP, is summarized in simplified form in Figure 2. We will shortly describe possibilities for the subroutine PropagateConstraints. If the CSP is unsolvable (some variables have no values remaining in their domains), then the procedure returns (triggering a backtrack). If the CSP has a single consistent value assignment for every variable, then this assignment is returned. Otherwise, the assignment heuristic *h* builds a new assignment of *s* variables. The SolveCSP routine is then called recursively on a CSP containing this assignment of variables (CSP'). If CSP' returns an assignment, this solution is returned unless *all* solutions are requested. Otherwise (if no valid assignment was found or if *all* solutions were requested), SolveCSP is then called on a CSP preventing this assignment of

variables (CSP''), and the result of SolveCSP is unioned with the current (possibly empty) set of results, and returned.

```
SolveCSP(CSP,s,h,all)
Inputs: CSP - a CSP instance, s - num vars to assign,
h - assignment heuristic, and all - a Boolean
indicating whether every or only first solution is
returned
Outputs:a (possibly empty) set of variable assignments

PropagateConstraints(CSP);
if (∃ v∈Variables(CSP) s.t. domain(v) = {})
   return {},
else if (∀ v∈Variables(CSP), |domain(v)|=1)
   return {Assignment(CSP)};
else
   newAssign ← {};
   for i=1 .. min(s,NumUnassignedVariables(CSP)
      Variable V ← h.chooseNextVariable(CSP);
      Value v ← h.chooseNextValue(CSP,V);
      newAssign ← newAssign U {V=v}
   CSP' ← CSP + newAssign
   result ← SolveCSP(CSP',s,h,all);
   if (result ≠ {} & !all)
      return result;
   CSP'' ← CSP + createNoGood(newAssign);
   return  solveCSP(CSP'',s,h,all) ∪ result;
```

Figure 2. Parameterized Formulation.

3.2 Boundary Cases

There are two distinct parameterizations that result in familiar algorithmic variations that serve to bound the space covered by the general framework. The first is to solve the underlying constraint satisfaction problem entirely, enumerating all feasible assignments (this corresponds to parameters of *s*=1, *h*=MRV, and *all*=True), and performing ordering queries (Section 2.2) over them to find a non-dominated assignment. Thus, with this heuristic search the recursion in SolveCSP continues to find all assignments. The other is to solve the problem from the other direction (parameterized by *s*=|*V*|, *h*=CP-net, and *all*=False), where assignments are generated in descending order of preference until the first feasible assignment is found. The framework's underlying (depth-limited, backtracking) search procedure coupled with the CP-net heuristic ensures that complete assignments are generated in non-increasing preference order (that is, an order consistent with the partial order over induced outcomes described in Section 2.2). Hence, the first feasible one must be non-dominated, so with this heuristic the recursion in SolveCSP ends with the first feasible assignment found. For ease of description we refer to these parameterizations as **CSP-first** and **Pref-first**, respectively.

We expect Pref-first to perform very well when problems are relatively under-constrained. For example, in the simple running problem from Section 2.1, if there are no constraints, then Pref-first amounts to simply running the forward-sweep algorithm (Section 2.2) and then confirming the assignment is feasible. Pref-first's performance will, however, worsen rapidly as problems become more difficult. Examining the outcomes in preference order is, from the perspective of the CSP, equivalent to blind chronological backtracking, and thus is more prone to degradation when infeasible assignments involve variables occurring near the root(s) of the CP-net. For instance, in the simple problem of Section 2.1, if Ann-Exercise is constrained to not be assigned ~~"b~~ike" (due to the flat tire), the Pref-first search would generate 5 complete assignments, the fifth (<Swim, Store, SBook>) being the first feasible assignment.

In contrast, we would expect the performance curve for CSP-first to mirror that of Pref-first, performing well on hard problems, but worse on easy ones. In the running example, if Ann-Exercise cannot be bike, then the MRV heuristic would immediately prune out the infeasible half of the assignments and only consider the 4 remaining ones. If there were even more constraints, they would whittle the candidates down even further, making CSP-first even faster. But if there were no constraints, CSP-first would generate all 8 assignments: CSP-first generates feasible assignments in an order uncorrelated with preferences, and so the non-dominated solution(s) cannot be identified until all feasible assignments are enumerated and compared.

The discussion so far has ignored the subroutine PropagateConstraints (in SolveCSP, Figure 2). Modern CSP search algorithms actually tend to incorporate powerful forward-checking that enables them to prune away portions of the search space when constraints dictate [9]. Such techniques can benefit both CSP-first and Pref-first approaches, in both cases removing values from unassigned variables' domains that are inconsistent with constraints associated with values assigned to the variables so far. In our running example, for instance, forward-checking could immediately remove ~~"B~~ike" from the Ann-Exercise variable's domain (given the unary constraint that rules it out), such that Pref-first could immediately skip over the infeasible outcomes, more efficiently finding the most-preferred feasible solution.

We perform our empirical validation using an off-the-shelf, forward-checking, constraint satisfaction solver called ~~"C~~SP for Java" (CSP4J) [9]. We also take advantage of an existing random problem generator [10]. One of the generator's parameters, *p*, specifies the probability that each tuple of values is excluded as being invalid for each n-ary constraint, and is a rough measure of how constrained individual problem instances are. We generate random CP-networks by adapting a Bayes-net generator available from the University of Sao Paolo Decision Making Lab (http://www.pmr.poli.usp.br/ltd/).

We generated performance curves over 100 randomly generated problems, each problem containing 35 constraints and 10 variables with 3 values in each variable's domain, for each value of the difficulty parameter *p* in increments of 0.05. Throughout this work, to enhance the clarity of our results, we present timing results as a sum rather than an average. The results are closely in accord with our expectations. Figure 1 shows the mirror-image exponential curves, with a distinct cross-over point as we move from less-constrained to more-constrained problems, along with an ~~"I~~nterleaved" curve, which we explain shortly. We observe that these naïve approaches partition the problem space into two regions—a lightly constrained region in which Pref-first is the superior choice, and a highly-constrained region in which the CSP forward checking allows CSP-first to perform better.

3.3 The Interleaved Approach

The mirror-image performance curves for our two boundary approaches suggest that we are likely to benefit from hybridizing the two algorithms by interleaving constraint propagation and preferential reasoning. In terms of our framework, we set *s*=1, *h*=CP-net, and *all*=False, so that once constraint propagation ends and tentative assignments must be made, SolveCSP (Figure 2) invokes the CP-net to select and assign only one variable. The CP-net heuristic chooses a variable with no unassigned parents, and picks the most preferred value from among those that remain in

the variable's domain, given the assignments that have already been made to its parents. The resulting algorithm is essentially isomorphic to that of Boutilier *et al*. [3], except that with *all*=False, this instance of the algorithm terminates after finding just one non-dominated solution rather than searching for all of them. Boutilier *et al*. [3] show that in such an algorithm, outcomes are considered in a preferentially non-increasing order. Thus, if we care only to find any non-dominated solution, the interleaved algorithm can safely terminate after finding the first feasible solution. It is this fact that accounts for much of the algorithm's performance advantage.

As expected, Figure 3 shows that the interleaved algorithm combines the features of the boundary approaches in such a way as to gain the benefits of each. Since the CP-net is used to order variables, the search can stop after finding the first feasible outcome, whereas a more CSP-centric search (CSP-First) must find every feasible outcome. On the other end of the spectrum, the interleaved algorithm compares favorably to Pref-first because allowing constraint propagation between CP-net-based assignments to remove infeasible values from variables' domains enables the algorithm to prune away large numbers of outcomes that Pref-first would otherwise consider.

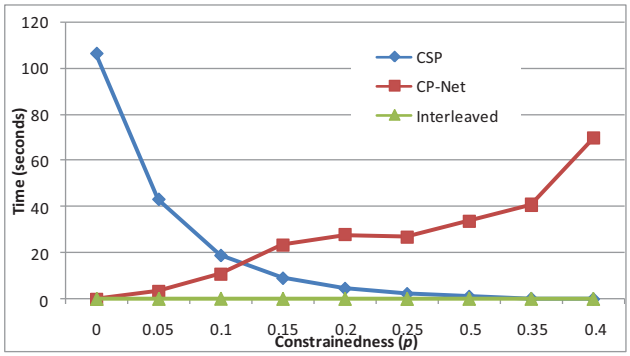

Figure 3. Performance curves for canonical approaches.

Figure 3 shows data generated from problems that were relatively small in order to allow the exponential algorithms to run to completion. These problems are not big enough for the interleaved algorithm to take any appreciable time. Figure 4 shows the performance of the interleaved algorithm on much larger problems (50 variables, 5 values each, 180 constraints). It also shows the performance of a traditional CSP search that uses CSP-centric heuristics and halts when finding the first (not necessarily optimal) feasible assignment. Again, to enhance the clarity of the graph, each data point is the sum of the time required for 100 randomly-generated problems at each value of *p*.

Figure 4 shows that, in general, traditional MRV-based heuristics outperform the CP-net interleaving heuristic in finding a *feasible* assignment. However, unlike the interleaved algorithm which can return the first assignment it finds (since it is, by definition, non-dominated, and thus a member of the set of most preferred), the MRV-based heuristic cannot provide any such guarantees about the preferability of the first solution it finds. So preference optimization must run MRV-based search to find and compare *all* feasible solutions before terminating. Figure 3 shows this, where finding the most preferred feasible assignment is always faster using the interleaved algorithm. Figure 4 motivates our satisficing approach (Section 4), since blending these two approaches can speed solving when only approximate preferability suffices.

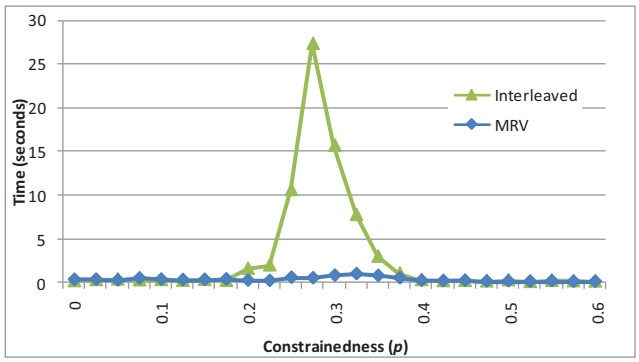

Figure 4. Interleaved vs. Traditional CSP algorithm.

Our empirical evaluation of the interleaved algorithm demonstrates that it is extremely fast at the ends of the problem space (< .01 seconds on average), while requiring more time in the region of the space that corresponds to a "phase transition", where the problems are most difficult. In addition, it is clear that for a large region in the middle of the space, including the "phase transition", the interleaved algorithm substantially improves performance over either of the boundary algorithms. The remaining question, then, is about the relative performance near the extremes of the problem space.

As we see in Figure 5 (generated on problems with 50 variables, 5 values each, and 180 constraints) on the hardest problems there is never a case where CSP-first outperforms the interleaved algorithm. At some point, problems become sufficiently constrained that a single feasible assignment is implied by the constraints. In such cases, regardless of the algorithm used, the initial propagation step ends up solving the entire problem, without ever needing to guess a tentative assignment, so the algorithm never performs "search" proper. Thus, in the limit, the performance curves for all algorithms converge to the same point. Short of that point, however, the interleaved approach is faster.

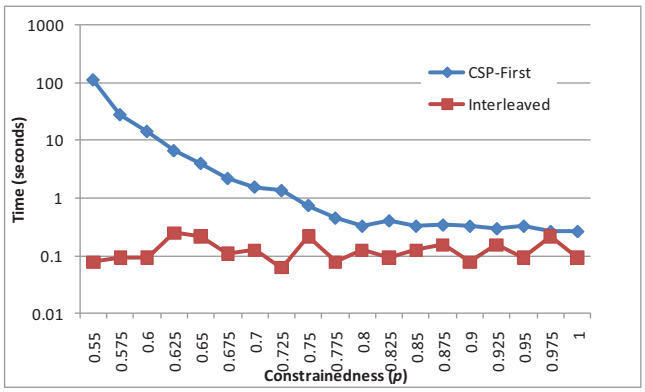

Figure 5. Most-constrained region of the search space.

The interleaved algorithm can be thought of as CSP-first that replaces commonly-used variable ordering heuristics with an ordering that is effectively random, in return for being able to stop after finding the first solution. Although on very hard problems using the CP-net variable ordering makes the interleaved algorithm approximately 25-times slower than CSP-first at finding *some* feasible assignment (Figure 4), the fact that CP-net ordering

guarantees that this *first* assignment is *optimal*, whereas CSP-first must explore its *entire* search space, more than compensates for CSP-first's efficient pruning of its search space. In Section 4, we discuss ways of realizing the performance benefits of heuristic variable ordering while preserving the early-termination properties of the CP-net ordering.

3.3.1 Correlated Problems

In many settings, such as in a setting with many competing agents, preferable resources may be scarce in practice. While in general, this paper focuses on the reasoning from the point of view of a single, independent, self-interested agent, one way that we have tried to make the random CSP instances better reflect real, multiagent scenarios is to include a parameter *corr* that represents the positive or negative correlations that constraints tend to have with preferability. The way we incorporate *corr* is simple: currently, the generator guarantees that the CSP instance is feasible by first generating a *guaranteed solution*, or a full assignment of values to variables, and then assuring that subsequent constraints do not exclude any tuples that correspond to this guaranteed assignment. When *corr*=0.0, there is no correlation, and this guaranteed solution is chosen completely randomly. When *corr* is positive, with probability *corr*, a variable assignment is chosen that corresponds to the next assignment suggested by the CP-net while building this guaranteed solution (roughly analogous to an agent acting in the presence of other cooperative, altruistic agents). When *corr* is negative, variables are assigned to their least preferred value during the guaranteed solution construction, with probability 1-*corr* (roughly analogous to an agent competing with other agents).

Figure 6 presents a high-level view of how the CP-net curves shift with different correlation values. When there is positive correlation, the CP-net curve grows much less quickly than before. When correlation is 1.0, the first solution is guaranteed to be feasible, and thus returns immediately. Since the correlation does not affect the CSP heuristic (which generates *all* solutions, and sorts based on preferability), we use the correlation only for analyzing the CP-Net vs. Interleaved crossover.

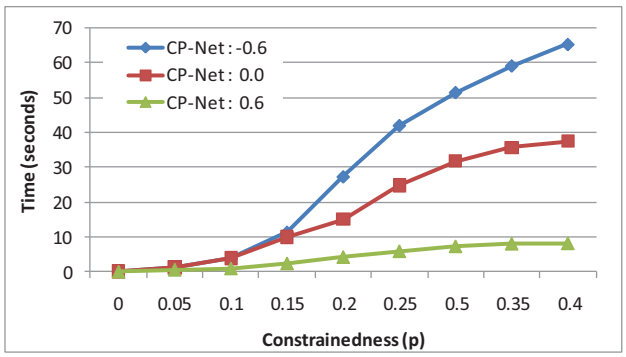

Figure 6. CP-net curves, shift as constraints are correlated with preference.

We observe results similar to Figure 5 at the other extreme of problem difficulty (where problems are extremely under-constrained) in Figure 7 (again generated on problems with 50 variables, 5 values each, and 180 constraints). On the easiest problems, it turns out that there is never a case where Pref-first outperforms the interleaved algorithm, even with constraints biased towards the CP-net (*corr* = 0.6). On problems without meaningful constraints (when *p*=0, all tuples are included as valid in all constraints), the two algorithms must exert identical effort; visiting each constraint exactly twice (once for each variable assignment of the two variables involved in the binary constraint). However, as soon as the problem is constrained enough that Pref-first might need to backtrack, the exponential cost of backtracking outweighs the low likelihood of needing to backtrack. In contrast, the interleaved algorithm detects constraint violations as soon as they occur, and so when backtracking is necessary, in expectation, an exponentially smaller number of variable-value combinations must be tried.

We were surprised to see how quickly the interleaved approach outperformed the CP-net on CSPs with positive *corr* settings. A positive value means that a portion (in this case roughly 60%) of variables in the guaranteed solution will be set in exact accordance with the CP-net. Therefore, the worst-case backtrack of the Pref-first approach on problems with positive *corr*=0.6 will contain an expected 60% fewer variables, (which is amplified by the exponential nature of backtracks). More realistically, problems are more likely to have a negative *corr* value. For example, resource constraints may dictate that preferable resources are scarcer, and hence constraints will often bias against preferable outcomes. While this clearly impacts Pref-first negatively, as Figure 7 shows, the interleaved approach is fairly robust to changes in *corr* values. This is important, since it demonstrates that, even in more realistic problems where structural correlations between preferences and constraints exists, the interleaved approach continues to dominate the Pref-first approach.

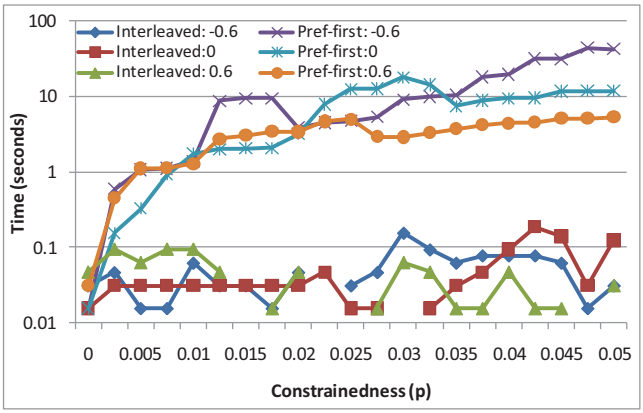

Figure 7. Least-constrained region of the search space.

3.3.2 Impact of Step Size

From experimentation, the interleaved approach also dominates the Pref-first approach even if the high backtrack cost of Pref-first is mitigated with a more conservative step size ($1 < s < |V|$) and problems are generated with a Pref-first bias (positive *corr* value). Analytically, the worst-case complexity of the arc-consistency algorithm used by the CSP4J solver (an AC-3 variant) is known to be $O(ed^2)$ [9], where e is the number of constraints and d is the size of the largest variable domain, while the expected-case complexity is much less than this. Loosely speaking, AC-3 is applied every time search changes the domain of a variable, or $O(a + b)$ times where a is the number of variable assignments and b is the number of backtracks. Assigning multiple variables at a time helps mitigate the number applications of the AC-3 algorithm during search, $O(a/s)$; however, it does so at the cost of increasing the number of backtracks necessary when a poor assignment is identified during search, $O(bds)$. Thus, although

backtracking occurs less frequently on even the most under-constrained problems, in expectation, the exponential (in d) increase in AC-3 applications outweighs the saved AC-3 applications during the assignment phase of search. In fact, the AC-3 algorithm exploits the sparseness of the constraint graph, avoiding its worst-case complexity, and in turn, negating most of the benefit, O(a/s) vs. O(a), of processing multiple variable assignments at once. Additionally, we found that even a dynamic adjustment strategy, which immediately regressed to a step size of 1 when encountering infeasibility, failed to outperform the original, interleaved algorithm, in the space of problems we explored.

4. A SATISFICING VARIANT

As shown in Figure 4, there is a computational cost to assigning variables in CP-net order instead of using a CSP-centric variable-ordering heuristic. To combat this cost, we add an additional parameter to our formulation that allows users of our decoupled constrained optimization techniques to gracefully trade optimality for efficiency. This parameterization also leads naturally to an anytime algorithm.

4.1 A Satisficing Parameterization

So far, the parameter h has statically dictated which variable assignment heuristic to use. However, h could be determined dynamically during search by introducing a satisficing parameter t that corresponds to a user-specified satisfaction threshold. Values of t are given between 0 and 1, and describe how close the user wants the returned solution to be to a non-dominated feasible assignment. Specifically, t describes the proportion of all outcomes (whether feasible or not) that 1) may dominate a returned outcome O while 2) not dominating an optimal solution O^*, such that O is accepted as a satisfactory solution. So, a value of t=.25, for example, means that for any outcome O returned by the algorithm, less 25% of all outcomes represent potential improvements over O.

For any partial assignment pa, Purrington and Durfee [6] showed how the CP-net semantics may be used to quickly determine a lower and upper bound, t_L and t_U respectively, on the rank, expressed as a proportion of the entire assignment space, achieved by any extension of pa to a full assignment. When search decisions are made optimally, t_U doubles as an upper-bound on the rank of O*, and thus the difference ($t_U - t_L$) is an upper-bound on the difference in rank between optimal solution O^* and any extension of pa to a complete assignment.

From this, the satisficing algorithm follows in a straightforward way. We assume that the user specifies the satisfaction threshold t as an algorithmic parameter. Then, after any variable domain change is propagated, the current lower and upper bound on satisfaction, t_L and t_U, are updated. If (t_U - t_L) $\geq t$, the next variable is chosen in the order specified by the CP-net (h=CP-Net, as with the simple interleaved algorithm). Since all variables are assigned optimally with respect to the CP-net until the (t_U - t_L) $\geq t$ condition is violated, t_U represents an upper-bound on the rank of an optimal assignment O^*. And since t_L represents the lower-bound ranking of any completion of the current partial assignment, when (t_U - t_L) $< t$, any feasible extension of the current partial assignment is guaranteed to be separated from an optimal assignment by less than 100t% of possible assignments, so the remaining variables can be assigned in MRV order (h=MRV), ignoring the user's preferences.

To illustrate, consider again the simple running example (Section 2.1). Let us say that the user (Ann) is easily satisfied, only caring that the outcome assignment is within 50% of the best feasible assignment. The CP-net will be used for the first assignment, assigning "Bike" to "Ann-Exercise." The lower bound of any complete assignment building from this partial assignment is 5/8 (to confirm this, in Figure 1 (right) the worst outcome with "Bike" is no worse than 5 out of the 8 outcomes). Since (t_U - t_L) = (1-.625) < .5, the satisficing variant shifts to the CSP-centric (MRV) heuristic to return whichever feasible completion to the assignment is easiest to find. Similarly, had "Bike" been unavailable, the CP-net heuristic would have chosen "Swim" and then the CSP-based heuristic would have taken over since the best possible outcome would now be better than 3/8 of the outcomes, which is less than 0.5.

This satisficing algorithm should be no slower than the interleaved algorithm. When t=0, (t_U - t_L) can never fall beneath t, so the algorithm is just the interleaved algorithm. When t=1.0, any partial assignment achieves the threshold, so the algorithm just returns the first outcome found using MRV, which we showed in 5 to be up to about 25 times faster than the interleaved algorithm for this space of random problems. Finally, for values of t between 0 and 1, the algorithm assigns some, but not all, variables in MRV order, which we expect to prune more of the search space than using the CP-net order, though the degree of savings available will depend on the specifics of the problem.

We verified our expectation by running 100 random problems using 50 variables with domain size of 5, using linear CP-nets for ease of implementation. For these experiments, we fixed p = .275, which corresponds to a portion of the problem space where problems were empirically determined to be difficult. We decreased satisfaction threshold t exponentially from 1.0 until the solutions were indistinguishable from those of the interleaved algorithm. As we have observed, the extreme points of t=1.0 corresponds to pure MRV (stopping after one solution), and approaches pure CP-net-order interleaving as t decreases. We show the results in Figure 8, summing the running time for the 100 problems for the sake of clarity. As a reference, the interleaved algorithm takes a sum of 24.8 seconds when run on these same 100 problems.

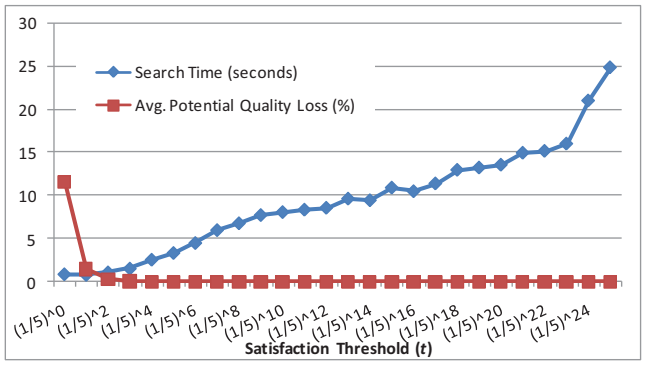

Figure 8. Search time and average potential quality loss vs. satisfaction threshold.

These results are in keeping with our expectations. For these problems, we observe a performance improvement of approximately 25-fold by using MRV instead of CP-net ordering (comparing t=1 to t=0). Furthermore we see algorithmic speed degrade relatively smoothly as t decreases, and the algorithm

behaves more and more like CP-net. It is important to note that we allowed the CP-net heuristic to break any ties encountered by the MRV variable and value ordering heuristics, leading to assignments that are separated from the optimal solution by an average of just under 12% of the all possible assignments, even when traditional MRV is used exclusively (t=1.0). The key observation is that this margin decreases rapidly as t approaches 0, allowing users with modest concessions on quality (within 1% of optimal) to solve problems an order of magnitude more quickly than the interleaved approach. While the actual quality loss may be affected when problem constraints are correlated with preferences, the satisficing algorithm is invariant to this correlation, since constraint propagation will simply eliminate the highly preferable, but infeasible results from consideration.

4.2 An Anytime Algorithm

Figure 8 shows that optimality can be flexibly traded for run-time efficiency, naturally leading to an anytime extension to the basic satisficing algorithm. Anytime behavior can be achieved simply by recording the value $t_{NEW} = (t_U - t_L)$, as soon as the $(t_U - t_L) < t$ condition is met. Thus, an anytime algorithm might first find any solution using MRV (t=1.0). This initial solution has some corresponding t_{NEW} value, which can be used as the input parameter for the next iteration of the algorithm by restarting the algorithm with (t=t_{NEW}). Eventually, this algorithm will terminate with an iteration that degenerates to the interleaved algorithm. Figure 8 shows what amounts to a first derivative of such an approach. While this approach is clearly slower at finding the optimal solution than the interleaved algorithm, it is capable of impressive anytime performance.

We can also use the t_{NEW} value that we compute to direct a branch-and-bound-style search. Once one complete solution is found, along with a corresponding t_{NEW} value, the search can continue, pruning the search space of branches that cannot possibly meet the new t_{NEW} threshold of the best solution identified so far. Although this strategy naturally avoids duplicated search effort, immediately backtracking to the point in the search where the $(t_U - t_L) < t$ condition was first met can further improve performance. Backjumping to the point allows the CP-net heuristic to pick up where it left off, optimally guiding the search process and avoiding a potentially large amount of backtracking over variables that, due to the nature of CP-nets, are unlikely to highly impact solution quality.

5. CONCLUSION

This paper makes several novel contributions towards solving decoupled constrained optimization problems. First, we offered a simple parameterization to generalize the traditional CSP algorithm. Second, we empirically demonstrated that approaches within this parameterization, including two boundary approaches, are dominated by an interleaved approach similar to [3], which had previously not been evaluated empirically. Additionally, we showed that adjustments to the step-size parameter s offered no improvement over the interleaved approach. In order to recover some of the loss in expected search efficiency accrued by replacing CSP-centric variable and value ordering heuristics with CP-net based ones, we introduced a satisfaction threshold t that we showed flexibly trades optimality for search efficiency, and also enables the creation of an anytime algorithm.

This paper provides a parameterization that is useful for evaluating future work in decoupled constrained optimization. In fact, we have been applying this parameterization to solving CSPs involving continuous variables, called HSPs [7], for use in assisting people with cognitive difficulties to manage their schedules. To our knowledge, algorithms for such problems have only been analyzed in a coupled setting [5]. Using the HSP framework, our parameterization correctly identifies the relevant questions: how do we make search decisions (h), how frequently do we interleave constraint processing with optimizing (s), and can we dynamically trade optimality for efficiency (t).

6. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their comments and suggestions. The work reported in this paper was supported, in part, by the National Science Foundation under grant IIS-0534280 and by the Air Force Office of Scientific Research under Contract No. FA9550-07-1-0262. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or United States Air Force.

7. REFERENCES

[1] Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Shiex, T., & Verfaillie, G. 1999. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3) (1999) pages 275-316.

[2] Boutilier, C., Brafman, R., Domshlak, C., Hoos, H. and Poole, D. 2004. CP-nets: A tool for representing and reasoning about conditional ceteris paribus preference statements. *JAIR* 21 (2004), pages 135-191 .

[3] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. and Poole, D. 2004. Preference-based constrained optimization with CP-nets. *Computational Intelligence*, 20(2) (2004), pages 137-157.

[4] Koriche, F. and Zanuttini, B. 2009. Learning Conditional Preference Networks with Queries. In *IJCAI 2009*, pages 1930-1935.

[5] Peintner, B., Moffit, M. D., and Pollack, M. E. 2005. Solving Over-constrained Disjunctive Temporal Problems with Preferences. In *ICAPS 2005*, pages 202-211.

[6] Purrington, K. and Durfee, E. H. 2007. Making social choices from individuals' CP-nets. In *AAMAS 2007*, pages 1122 – 1124.

[7] Schwartz, P. 2007. Managing Complex Scheduling Problems with Dynamic and Hybrid Constraints. PhD. Diss., CSE., Univ. of Mich., Ann Arbor (2007).

[8] Schiex, T. 1992. Possibilistic constraint satisfaction, or –How to handle soft constraints.” In *UAI 1992,* pages 269-275.

[9] Vion, J. 2006. CSP4J: a Black-box CSP Solving API for Java In. *Proc. of the 2nd International CSP Solver Competition* (2006)*,* pages 75-88.

[10] Xu, K, Boussemart, F. Hemery, F. Lecoutre, C. 2007. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence* 171 (2007), pages 514-534.