Constraint Programming for Component-Level Robot Design

Andrew Wilhelm[1]* and Nils Napp[1]

Abstract— **Effective design automation for building robots would make development faster and easier while also less prone to design errors. However, complex multi-domain constraints make creating such tools difficult. One persistent challenge in achieving this goal of design automation is the fundamental problem of component selection, an optimization problem where, given a general robot model, components must be selected from a possibly large set of catalogs to minimize design objectives while meeting target specifications. Different approaches to this problem have used Monotone Co-Design Problems (MCDPs) or linear and quadratic programming, but these require judicious system approximations that affect the accuracy of the solution. We take an alternative approach formulating the component selection problem as a combinatorial optimization problem, which does not require any system approximations, and using constraint programming (CP) to solve this problem with a depth-first branch-and-bound algorithm. As the efficacy of CP critically depends upon the orderings of variables and their domain values, we present two heuristics specific to the problem of component selection that significantly improve solve time compared to traditional constraint satisfaction programming heuristics. We also add redundant constraints to the optimization problem to further improve run time by evaluating certain global constraints before all relevant variables are assigned. We demonstrate that our CP approach can find optimal solutions from over 20 trillion candidate solutions in only seconds, up to 48 times faster than an MCDP approach solving the same problem. Finally, for three different robot designs we build the corresponding robots to physically validate that the selected components meet the target design specifications.**

I. INTRODUCTION

Designing robots is difficult because it involves nested, multi-domain, multi-objective constraints. Given a target performance specification such as payload, endurance, or speed, designers need expertise in several domains to translate it into a final robot design. Errors in one part of the system may result in subtle, difficult to diagnose failures in other subsystems. For example, a designer proficient in systems programming may forget to account for the combined transient power draw of components resulting in a design that produces intermittent, difficult to replicate faults even though it passes static and component-level testing. By decreasing the chance of such design errors, automating component selection for robot design can reduce both the development time and likelihood of encountering failures during testing.

Component selection is a particularly difficult optimization problem since the discrete solution space increases combinatorially as the number of available components increase, making brute force search methods intractable except for the smallest of problems. Furthermore, the categorical nature of component choices makes it difficult to directly map the problem onto well-developed industrial integer, or mixed-integer solvers, such as IBM CPLEX [1], or Gurobi [2]. One approach is to build simplified robot models that map onto such solvers, but this requires judicious approximations typically by an expert, see Sec. II.

[1]Department of Electrical and Computer Engineering, Cornell University, Ithaca, NY, 14850, United States
*Correspondence to ajw343@cornell.edu

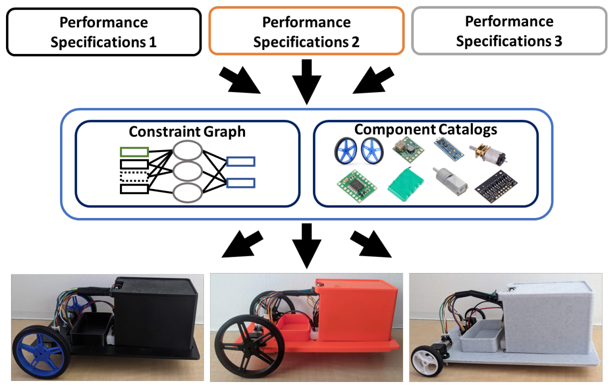
Performance Specifications 1
Performance Specifications 2
Performance Specifications 3
Constraint Graph
Component Catalogs

Fig. 1: Overview of the component selection problem for robot design. Given high-level performance specifications, our bounded constraint optimization algorithm selects components from an available parts list. We designed and physically validated three different differential drive robot designs using our algorithm. (Component images from pololu.com)

Another approach that has been explored in the context of design problems is to formulate robots as a monotone co-design problem (MCDP) [3]. Given one or more objectives, these can be solved iteratively using a fixed-point algorithm to compute the Pareto front of minimal solutions that meet the target specification. This framework has a convenient modeling language and unit-checking of designs and can easily express non-linear (albeit monotone) relationships between components. This makes it somewhat easier to express the component selection problems without approximations, yet the current solver does not scale well with larger catalogs and many circular dependencies between components.

We take an alternative approach of solving a useful subclass of problems and focus on a combinatorial optimization problem without having to approximate our system as linear or monotone. Our approach uses constraint programming (CP) and a depth-first branch-and-bound algorithm, see Sec. III. Compared to the MCDP formulation our approach cannot handle continuous decision variables or determine a multi-objective Pareto front of solutions, but it can compute multiple minima if they exist.

Our key contributions are as follows:

- We formalize the component selection problem as a combinatorial optimization problem which does not

require an approximated model.
- The effectiveness of CP depends on heuristics for ordering variables and domains. We solve the optimization problem using CP and present two heuristics that significantly speed up solving this specific problem type.

We also describe a number of techniques necessary to achieve practical run times, including adding redundant constraints to more efficiently trim the search space. We demonstrate our CP approach can determine optimal solutions up to 48 times quicker than an MCDP approach solving an identical problem and have validated the effectiveness of our proposed heuristics when using an industrial constraint programmer. We have also physically built three different robot designs and validated that they meet target specifications.

II. RELATED WORK

Ideally the robot design process can be automated, provide formal guarantees, generate minimal solutions, and be fully integrated in an end-to-end system [4]. Methods developed for single-domain problems cannot achieve these goals so deliberate progress must be made towards automating multi-domain design and synthesis tools.

One approach to automate the design process is to use expert-designed modular components that facilitate composition. When carefully chosen, even a small set of components allows these approaches to meet a wide variety of performance specifications. Modular components can abstract and simplify the design process (and are even used in some children toys [5], [6]), which is essential for non-expert users.

For instance, [7] developed a system to allow novice users to design printable and foldable robots using a library of basic and pre-designed components. This system was later extended in [8] to allow users to describe robot behavior using structured English, but users still needed to associate components with grammatical definitions and specify structural parameters during the design process. [9] also developed a tool to generate foldable robot designs that simulated and evaluated proposed designs to provide feedback and guide users via an iterative design process, and [10] developed a visual design environment that used a heuristic guided tree search algorithm to autocomplete user designs. Similar to [8], these approaches used a limited number of expert defined or hand-picked modules and did not provide guarantees on the electronics system working.

In contrast to approaches that use modular components, model-based approaches constrain or parameterize the robot geometry but can accommodate a wider range of generic components. Often these approaches receive limited or no user feedback during the design process which makes them suitable for optimization based synthesis.

If choosing components from a finite set, the problem becomes a combinatorial optimization problem in which the objective is to find an optimal object from a finite set of objects [11]. Without approximations and/or highly constrained scenarios these optimization problems are frequently difficult to solve since the problem tends to be a non-convex, non-linear combinatorial optimization problem. There are several approaches to solving constrained combinatorial optimization problems including deep learning [12] or particle swarm optimization [13], but here we present techniques specifically used in system and robot design.

The most common model-based approaches are linear and quadratic programming approaches. By linearizing their system, [14] and [15] used more traditional linear optimization techniques (mixed integer and binary programming, respectively) to determine component selection for quadcopters. Quadratic programming has also been extensively used in [16], [17], and [18], but all three of these approaches used parametrized components and without the ability to select pre-defined components from a catalog.

MCDPs are a different model-based approach that can be used to solve combinatorial optimization problems. Kleene's algorithm is used to compute an anti-chain of solutions and determine the Pareto front of minimal solutions. For the computational approach to work, the mappings between functionalities and required resources of each component need to be monotone, but can otherwise be non-linear, non-convex, or non-continuous [3].

Many of these prior works primarily optimize the robot's morphology and use relatively small catalogs of (parameterized) components operating under the assumption that a designer has a set of pre-selected suitable components. In contrast our approach prioritizes component selection from a large catalog (hundreds of distinct components instead of dozens) which is a typical situation a designer might encounter using an online parts vendor. This creates a much larger search space containing trillions of candidate solutions.

Our approach utilizes a branch-and-bound algorithm to efficiently trim and exhaustively search the entire combinatorial space. Branch-and-bound uses a branch step that partitions the search space into subsets, and a bound step that provides a lower bound on the cost for a given subset [19]. This bound is continually refined and used to prune regions of the search space. The branch-and-bound technique is the best known framework for many NP-hard combinatorial optimization problems [19] including design problems related to robotics and automation, such as designing soft actuator systems [20], spatial layout problems for manufacturing equipment [21], and wiring inside logic circuits [22].

III. METHODS

This section first describes the notation and solution techniques of the easier constraint satisfaction problem (CSP) (Sec. III-A) as it forms the basis for the constrained optimization problem (COP) we use to solve for optimal components (Sec. III-B). Sec. III-C describes techniques for speeding up the COP for robot component selection that are necessary to achieve practical solve times.

In this paper, we use a differential drive, transport robot as an example design problem since this type of robot contains common components (such as motors, microcontroller, etc.) and is widely used in industrial applications. We assume the designer knows the robot model and, given a set of target specifications such as speed, endurance, and maximum

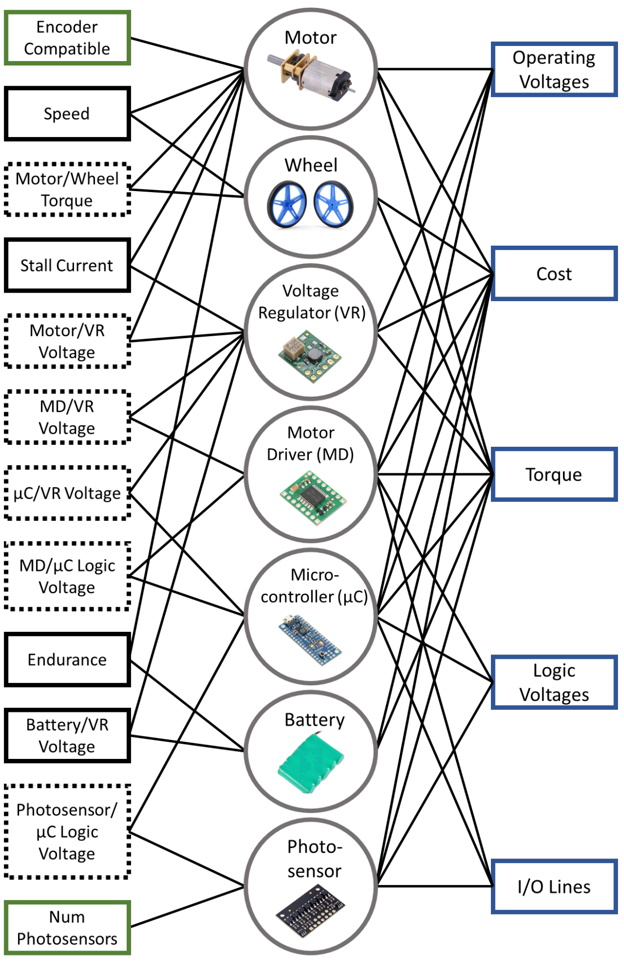

Fig. 2: The bipartite constraints graph for our differential drive robot model. The combinatorial optimization problem's variables are the components (center) represented as gray ovals. Unary, binary, and global constraints are green, black, and blue rectangles, respectively. Redundant binary constraints added to improve run time (Sec. III-C.3) are indicated by a dashed outline. Each constraint node implies a mathematical constraint that expresses component compatibility in terms of variable properties. An edge between a constraint and a variable indicates that those variable properties are present in that constraint.

navigable incline, would like to select components to meet these high-level specifications while minimizing cost.

A. *Constraint Satisfaction Problems*

A CSP describes problems using constraints on variables and is commonly applied to problems such as job scheduling, map coloring, sudoku, and the 8-queens problem [23]. While our goal for robot design is to optimize for cost, the goal of solving a CSP is to find any feasible solution such that no constraints are violated. The typical solution technique is to use a back-tracking search that uses the constraints to skip over infeasible regions of the search space, and our proposed solution uses the same core ideas but simply keeps searching and refining constraints, Sec. III-B.

A CSP has a finite set of variables $V = \{v_1, v_2, \dots, v_n\}$, a set of finite domains (one for each variable) $D_i = \{d_{i,1}, d_{i,2}, \dots, d_{i,m}\}$, and a finite set of constraints $C = \{C_1, C_2, \dots, C_p\}$ that specify valid combinations of values. This formulation lends itself to the component selection problem where one knows a general robot model (e.g. a differential drive robot) and needs to select specific items from possibly large yet finite sets of potential components. Each variable v_i describes one component type that needs to be selected, and the associated domain D_i is the set of all available components of that type. Here, the domains are populated from parts that can be easily ordered from online vendors. Each variable or component type has associated properties, which are required for determining whether or not component combinations are viable. For instance, in our implementation the variable v_{motor} represents the motor class of components, and this variable has properties such as ω_{max} or I_{stall} representing the maximum angular speed and stall current of the motor, respectively, which are populated from the motor documentation.

Constraints C_i are relational constraints between the properties of components that can be expressed as mathematical expressions, e.g.

$$V_{\text{max}} \leq v_{\text{motor}}.\omega_{max} * v_{\text{wheel}}.radius$$

where V_{max} is the maximum speed of the robot, and the terms of the right hand side refer to the associated properties of the motor and wheel variables. These constraints encode not only viable combinations of components but also the performance specification of the robot, which can be expressed using mathematical relations on variable properties.

For our example problem selecting components for a differential drive robot, we use 7 different component types which have between 3 and 9 associated properties. There are 17 total constraints in our model, but 6 of these are redundant binary constraints added to improve run time (see Fig. 2 and Sec. III-C.3). These constraints ensure the target performance specifications are met (e.g. the motor can provide enough torque and angular speed to move the maximum payload up the maximum incline) and also account for system considerations (e.g. compatible operating and logic voltages or sufficient number of I/O lines). While these constraints may involve several variables, the expressions themselves are not complicated and have been omitted due to space limitations.

A satisfying *assignment* for a CSP is a list of specific values for each variable that is chosen from the associated domains such that all constraints are met. A partial assignment is where only a subset of the variables have an assigned domain value while other variables remain unassigned.

A key idea is that even a partial assignment can restrict the possible values of the unassigned variables. The search of satisfying solutions proceeds by assigning variables and then ruling out possible values of the unassigned variables from their domains based on the constraints. If any of the domains of unassigned variables become empty, there is no satisfying solution that contains the assigned variables and that portion of the search space can be eliminated without having to enumerate all possible combinations.

Constraints can be classified based on the number of variables they are connected to, which determines how they are used to rule out values during the search. Unary constraints are constraints on the domain values of a single variable, and can be treated in a pre-processing step to eliminate elements of the domain. Binary constraints are connected to two variables and can be efficiently propagated via a method called Maintaining Arc Consistency (MAC) [23], which uses the AC-3 algorithm to propagate binary constraints. Constraints on three or more variables are called global constraints. These constraints are checked as soon as possible but are not directly propagated by MAC, see Sec. III-C. We express the structure of the constraints as a bipartite graph between constraints and variables, where an edge between a variable and a constraint indicates its mathematical expression uses capabilities from that particular variable (see Fig. 2).

One way to improve efficiency is the order in which to assign variables and their domain values using domain-independent heuristics. In determining which order to select variables, a common approach is the Minimum-remaining-values (MRV) heuristic which chooses the variable with the fewest "legal" values. The intuition behind this is to pick a variable that is more likely to fail soon, resulting in fewer combinations to check since the search space is pruned more efficiently. In contrast, to determine the order to select domain values, the least-constraining-value (LCV) heuristic chooses domain values that eliminate the fewest domain values of neighboring variables. Since the objective of solving a CSP is to find any feasible solution, selecting domain values by LCV helps to find a solution quicker since it chooses the most likely values to be in a solution.

B. Component Selection as a COP

For the component selection problem, we assume the robot model and performance specification have been expressed as constraints between components, and among all satisfying solutions for the associated CSP the designer is looking for an optimal component selection to minimize system cost.

While many of the same CSP techniques can be used in solving a COP, the approach to solving standard CSPs is good at finding a feasible solution quickly but does not provide an optimal solution. A CSP solver stops as soon as it reaches a valid assignment for all variables, whereas a COP solver must continue iterating to exhaustively check the entire search space. Since the search space could contain a large number of valid solutions, simply removing the previously found combination and running the CSP solver again would quickly become intractable. Instead, to find an optimal solution we use a depth-first branch-and-bound algorithm that uses the current minimal cost to further prune the search tree as minimal solutions are found.

1) Depth-first Branch-and-Bound Using a Cost Bound: The goal of branch-and-bound is to efficiently consider all possible solutions while minimizing the number of solutions that have to be evaluated. To do this, the search space is recursively split into smaller sets, called branching. After branching, a lower bound of the smaller search space is determined and compared to a global upper bound. If the lower bound is larger than the global upper bound, the algorithm does not need to continue searching that solution subset since the minimal solution cannot be in that subset.

In our example design problem, we are attempting to minimize the cost of a differential drive, transport robot. In this case the global upper bound is the cost of the cheapest solution found so far. To determine a lower bound on cost, we evaluate the total cost of the current partial assignment. Our approach maintains a list of all solutions with the same cost (*solns* in Algorithm 1) and clears the list if a cheaper solution is found. If the algorithm is terminated early during solving, it can still provide locally minimal (but not necessarily globally minimal) solutions.

Algorithm 1 gives the pseudocode for a recursive bounded constraint optimization algorithm for component-level robot design. Using the techniques discussed above, the algorithm attempts to efficiently prune the search space and provide the set of all minimal solutions for a given set of specifications. If no solution exists, the algorithm returns an empty set.

C. Improving Constrained Optimization for Component Selection

In the remainder of this section we present methods to improve the solve time of our CP approach. Even with depth-first branch-and-bound, application-specific heuristic strategies are important for efficiently solving a COP [24]. In

Algorithm 1 Constraint Programming Algorithm: Our CP algorithm uses a recursive function *backtrack* to solve a COP. Variables *a, cb, v,* and *d* represent the current assignment, global cost bound, COP variable, and domain value, respectively. The *get_next_var* and *order_domain_vals* functions use the variable and value ordering heuristics presented in Section III-C.1 and III-C.2 to select the next variable or domain value. The function *eval_global_constr* evaluates the global constraints on partial variable assignments.

```
Input: ConstraintOptimizationProblem $cop$
Output: $solns$
 1: $solns \leftarrow \{\}, cb \leftarrow 0$
 2: assert_unary_constraints($cop$)
 3: $cb, solns \leftarrow$ backtrack($\{\}, solns, cb$)
 4: return $solns$
 5:
 6: function backtrack($a, solns, cb$)
 7:     if is_complete_assignment($a$) then
 8:         $total\_cost \leftarrow$get_total_cost($a$)
 9:         if $total\_cost < cb$ then
10:             $cb \leftarrow total\_cost$
11:             $solns \leftarrow \{\}$
12:         end if
13:         $solns$.append($a$)
14:         return $cb, solns$
15:     end if
16:     $v \leftarrow$ get_next_var($cop, a$)
17:     for $d$ in order_domain_vals($cop, v, a$) do
18:         if $cop$.num_constr_conflicts($v, d, a$) is 0 then
19:             $cop$.assign($v, d, a$)
20:             if forward_check($cop, v, d, a$) then
21:                 if eval_global_constr($cop, v, d, a, cb$) then
22:                     $cb, solns \leftarrow$ backtrack($a, solns, cb$)
23:                 end if
24:             end if
25:         end if
26:     end for
27:     $cop$.unassign($v, a$)
28:     return $cb, solns$
29: end function
```

this context, we present heuristics for selecting variables and domain values that are specific to the component selection problem. We also discuss how placing redundant binary constraints on the optimization problem can further improve solve time and identify which types of constraints benefit from this technique.

1) Variable Ordering – Least Cost Difference (LCDiff): In determining which variable to select next, we use the least difference in cost between the cheapest and most expensive domain values of each variable:

$$\text{least\_cost\_diff} = \underset{var}{\text{argmin}} \max_{d_i, d_j} |d_i.\text{cost} - d_j.\text{cost}| \quad \text{for } d_i, d_j \in D_{var}$$

The rationale behind this variable ordering is that the solver can get caught in a regime where it is evaluating many possible solutions that are providing only minor reductions to the global cost (since backtracking changes the last variables in the assignment). By ordering the variables such that the largest difference variables are selected last, any changes to complete assignments after backtracking have the potential to provide a large reduction in the global cost bound, helping to eliminate large portions of the search tree. Variable ordering is performed on line 16 of Algorithm 1.

2) Domain Value Ordering – Least Cost Lexicographic Order (LCLO): Our approach to domain value ordering uses a lexicographic order to sort the components by their general "utility" based on their properties (line 17 of Algorithm 1). First, components are ordered by increasing cost since the problem objective is to minimize system cost. In the event where two domain values have the same associated cost, we create a lexicographic order based on their other properties where components that can provide greater functionality with fewer resources are ordered first.

The intuition behind this approach is to first and foremost pick the cheapest component as the problem objective is cost minimization. Otherwise, one should pick the "best" domain value available. For instance, if two motors are the same cost but one motor could provide greater torque and angular speed, then choosing this motor would be advantageous as it is more likely to be part of an optimal solution.

3) Redundant Constraints: When generating partial assignments, the sooner a global constraint can be evaluated, the quicker the solve time. With this in mind, we add redundant binary constraints that allow certain global constraints to be evaluated before all of their associated variables have been assigned. Placing redundant constraints on COPs is a common technique that improves run time through better constraint propagation and faster domain reduction without affecting the solution space [25], [26], [27].

Under certain conditions a global constraint can be evaluated before all of its associated variables have been assigned. The key requirement is that after certain "critical" variables have been assigned, assigning more variables will not cause a constraint that is violated to become unviolated. If this property were not to hold, then sections of the search space could be trimmed even though viable solutions might still exist in that space. In context of our target application of robot design, we have found this is a common property for global constraints to have since many global constraints have summation terms that increase monotonically.

For example, the global constraint on torque is a function of the system weight, wheel diameter, and motor torque: $v_{\text{motor}}.\tau \geq v_{\text{wheel}}.radius * system\_weight * max\_incline + friction\_torque$. Since $system\_weight$ is the summation of each component's weight, this global constraint is connected to all variables. However, only the motor and wheel are "critical" variables since, after these variables are assigned, the $system\_weight$ can only increase monotonically as more components are added and, once this constraint is violated, it cannot be unviolated after assigning more variables.

Although these redundant constraints provide a looser bound than the global constraint (since only a subset of variables are being evaluated instead of all variables in the constraint), in particular binary redundant constraints are advantageous since MAC can then be used to propagate binary constraints and reduce the size of the search space before proceeding to check candidate partial assignments. Note that the original global constraint is still in the model so answer accuracy is not affected. The constraints with dashed, black outlines in Fig. 2 are redundant binary constraints derived from global constraints that were added to our robot model to improve run time. In Algorithm 1 we evaluate the global constraints on line 21.

IV. Results

We have used our CP algorithm to design and physically implement three different robots, and have also analyzed the algorithm's run time for different ordering heuristics and in comparison to the MCDP approach. We have chosen to compare to MCDP since our robot model is already monotone and it is straightforward to conduct a direct comparison between methods as these are already solving identical problems. To compare to LP or QP methods we would otherwise need to linearize the model. All provided run times in this section were collected over 10 trials on a standard office desktop computer (2.10 GHz CPU with 12 cores, 16 GB RAM).

While the backtracking, depth-first search algorithm in Algorithm 1 is a common technique to solve COPs, we have written our own implementation as we are interested in eventually extending our work to multi-objective problems. Currently available industrial constraint programmers are highly optimized but only perform single objective optimization and do not keep track of a Pareto front of solutions. However, later in this section we provide experimental data where we have implemented our proposed heuristics with an industrial constraint programmer to not only provide example solve times on a standard office computer when using a highly optimized solver but also demonstrate that our proposed heuristics are effective with these solvers (Table IV).

The high-level design specifications for three robots were selected to generate differential drive, transport robot designs that each prioritize different possible functionalities of the final design: navigating a steep incline (Steep Incline specifications), operating for an extended period of time on

	Steep Incline	**Endurance**	**Heavy Payload**
Motors (1/1/3 Chosen)	6V, HP, 300:1 **$16.95 x 2**	6V, HP, 100:1 **$16.95 x 2**	6V, MP, 250:1 **$16.95 x 2**
Battery (1/2/1 Chosen)	NiMH, 350 mAh **$7.55**	NiMH, 2200 mAh **$15.15**	NiMH, 350 mAh **$7.55**
Wheel (3/1/3 Chosen)	⌀60mm **$4.25**	⌀32 mm **$2.95**	⌀70 mm **$4.75**
Motor Driver (1/1/1 Chosen)	Dual 1 Amp **$3.33**	Dual 1 Amp **$3.33**	Dual 1 Amp **$3.33**
Microcontroller (4/4/4 Chosen)	16 MHz A-Star, 5V **$4.95**	16 MHz A-Star, 5V **$4.95**	16 MHz A-Star, 5V **$4.95**
Photosensor (2/2/2 Chosen)	7×, analog **$5.40**	7×, analog **$5.40**	7×, analog **$5.40**
Total Cost	**$59.38**	**$65.68**	**$59.88**

TABLE I: **Design System Results – Component Selection:** The components selected by our CP algorithm for each of the three different sets of performance specifications. Each design uses the same motor driver and voltage regulator. For each of these designs, the tool determined 16 or more different combinations of components could be used interchangeably and still meet the performance specifications. To simplify physical testing, we chose to test with only one of these combinations and have included those component specifications in the table.

	First Solution	**Optimal Solution**	**Solve Time**
MRV, LCV	**0.44 sec** (0.44, 0.46)	**3556 sec** (3530, 3567)	**4000 sec** (3973, 4013)
MRV, LCLO	**105 sec** (105, 106)	**183 sec** (181, 184)	**2523 sec** (2508, 2541)
LCDiff, LCV	**0.080 sec** (0.072, 0.092)	**90.8 sec** (90.6, 90.9)	**91.3 sec** (91.2, 91.5)
LCDiff, LCLO	**0.079 sec** (0.072, 0.089)	**0.079 sec** (0.072, 0.089)	**58.8 sec** (58.6, 59.1)

TABLE II: **Run Time Results – Variable and Domain Value Ordering:** Run times of our CP algorithm with and without using the variable and domain value ordering heuristic presented in Sections III-C.1 and III-C.2. Each row uses a different combination of variable and value orderings. Times shown are for the median (bolded), 10th, and 90th percentiles over 10 trials when solving the component selection problem using the Endurance robot specification. Our proposed heuristics (bottom row) significantly improve the algorithm's performance compared to traditional CSP ordering heuristics (top row).

one battery charge (Endurance specifications), and carrying a heavy payload (Heavy Payload specifications).

Table I shows the designs generated by the system using 651 distinct off-the-shelf components from Pololu's online catalog, a common robot parts supplier. These components produce 21.8 trillion ($2.18 \cdot 10^{13}$) candidate solutions our solver must search for an optimal solution. For each of the three different sets of performance specifications, our algorithm provided 16 or more optimal designs. These designs all had the same cost, but used different wheels, motors, microcontrollers, and photosensors. To simplify physical testing, only the designs using a 5V @ 16 MHz microcontroller and an analog output photosensor were tested.

To demonstrate the efficacy of our proposed heuristics, we ran trials to compare the run times using different combinations of our proposed heuristics (LCDiff and LCLO) with heuristics traditionally used in solving CSPs (MRV and LCV), see Table II. As our CP approach can provide intermediate solutions, we have included the times to determine the first and optimal solutions. Each of our proposed heuristics individually reduces the solve time, and using these heuristics in combination solves the optimization problem almost 70 times quicker than using MRV and LCV together.

	Constraint Programming Algorithm			**MCDP**
	First Solution	**Optimal Solution**	**Solve Time**	**Solve Time**
Steep Incline	**0.031 sec** (0.031, 0.050)	**14.3 sec** (14.3, 22.8)	**118 sec** (113, 159)	**52 min** (47, 125)
Endurance	**0.079 sec** (0.072, 0.089)	**0.079 sec** (0.072, 0.089)	**58.8 sec** (58.6, 59.1)	**48 min** (46, 170)
Heavy Payload	**0.040 sec** (0.040, 0.063)	**115 sec** (114, 146)	**245 sec** (245, 301)	**64 min** (46, 354)

TABLE III: **Run Time Results – Algorithm Comparison:** Run times for our CP approach and the MCDP framework. Times shown are for the median (bolded), 10th, and 90th percentiles over 10 trials. As our algorithm provides intermediate answers while solving, we have also included the time to find the first feasible solution and the time to find the first optimal solution.

	Steep Incline	**Endurance**	**Heavy Payload**
Industrial Solver Heuristics	**4.85 sec** (4.83, 4.95)	**4.88 sec** (4.79, 4.95)	**4.96 sec** (4.90, 5.01)
Our Heuristics	**4.72 sec** (4.66, 4.76)	**4.75 sec** (4.70, 4.94)	**4.74 sec** (4.66, 4.77)

TABLE IV: **Run Time Results – Heuristic Comparison with Industrial Solver:** Run times for the default variable and value ordering heuristics of an industrial solver (IBM CP Optimizer) compared to our proposed heuristics (presented in Sections III-C.1 and III-C.2) implemented in the same solver. Times shown are for the median (bolded), 10th, and 90th percentiles over 10 trials. For the problem of component selection, there is a small but significant ($p < .05$) reduction in run time when using our proposed heuristics (bottom row) over the default heuristics of IBM CP Optimizer when performing a depth-first search.

We directly compared the run time of our CP approach to an MCDP approach (Table III). Across all three robot designs, our CP approach provides an optimal solution at least 15 times and up to 48 times faster than the MCDP approach and determines a feasible solution within one tenth of a second.

We also used our heuristics within IBM CP Optimizer, an industrial constraint programming solver, and compared the performance using the industrial solver's default variable and value ordering heuristics to our proposed heuristics (Table IV). When solving the component selection problem using a depth-first search, there is a small but significant ($p < .05$) reduction in run time when using our proposed heuristics over the default heuristics.

Finally, each of the three robots were tested over three trials to empirically verify that they met the target specifications. At the maximum payload and speed, the average endurance over three trials was 56.3 min, 207.2 min, and 69.5 min for the Steep Incline, Endurance, and Heavy Payload robots, respectively. All three robots were able to operate for longer than their minimum endurance and thus all three surpassed their respective target performance specifications.

V. CONCLUSION

In this paper we presented an alternative approach to the component selection problem for robot design using a COP formulation and developed a depth-first branch-and-bound algorithm to solve this problem. To improve the run time of our algorithm, we proposed two heuristics for ordering variable and domain values specific to the component selection optimization problem, and also applied redundant binary constraints for better constraint propagation and faster domain reduction. We demonstrate that the run time of our approach is significantly quicker than the MCDP approach, and physically built robots using the solutions generated from our approach and validated they met the target specifications.

Our approach is well-suited for the discrete combinatorial optimization problem of component selection in which even small catalogs can produce trillions of combinations. However, our CP algorithm cannot solve for solutions that have continuous or mixed domains, which the MCDP framework and linear and quadratic programming can handle.

Additionally, when using our CP algorithm the model does not need to be made linear or monotone. For the other approaches, these approximations provide trade-offs. Linear and quadratic programming can have quicker solve times since the linearization allows for computationally quick matrix operations to be used to solve the problem, while a monotone model allows MCDPS to solve multi-objective optimization problems.

In future work we plan to conduct a further analysis of run time, determining how the algorithm scales as component catalogs increase in size and how the run time complexity compares to other approaches. Since one would expect CP to work better when the problem is highly constrained and there are fewer satisfying solutions, it would be useful to compare run times between such cases. Finally, we plan to extend our current work to a larger library of robot models and eventually develop a more complete, automated design system that accounts for more aspects of robot design such as the robot morphology and design of a suitable controller.

ACKNOWLEDGMENT

This material is based on work supported by the National Science Foundation grants NSF#1846340, NSF#2054744, and the Graduate Research Fellowship Program DGE#2139899. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] International Business Machines Corporation (IBM), "User's Manual for CPLEX," 2023. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio

[2] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[3] A. Censi, "A mathematical theory of co-design," *arXiv preprint arXiv:1512.08055*, 2015.

[4] A. Q. Nilles, D. A. Shell, and J. M. O'Kane, "Robot design: Formalisms, representations, and the role of the designer," *arXiv preprint arXiv:1806.05157*, 2018.

[5] KINEMATICS GMBH, "Tinkerbots," Accessed Mar. 1, 2022 [Online]. [Online]. Available: https://www.tinkerbots.de/?lang=en

[6] Mod Robotics, "Cubelets Robot Blocks," Accessed Mar. 1, 2022 [Online]. [Online]. Available: https://modrobotics.com/

[7] A. M. Mehta, J. DelPreto, B. Shaya, and D. Rus, "Cogeneration of mechanical, electrical, and software designs for printable robots from structural specifications," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2892–2897.

[8] A. M. Mehta, J. DelPreto, K. W. Wong, S. Hamill, H. Kress-Gazit, and D. Rus, "Robot creation from functional specifications," in *Robotics Research*. Springer, 2018, pp. 631–648.

[9] A. Schulz, C. Sung, A. Spielberg, W. Zhao, R. Cheng, E. Grinspun, D. Rus, and W. Matusik, "Interactive robogami: An end-to-end system for design of robots with ground locomotion," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1131–1147, 2017.

[10] R. Desai, Y. Yuan, and S. Coros, "Computational abstractions for interactive design of robotic devices," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1196–1203.

[11] A. Schrijver *et al.*, *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003, vol. 24.

[12] J. Whitman, M. Travers, and H. Choset, "Modular mobile robot design selection with deep reinforcement learning," in *NeurIPS Workshop on ML for engineering modeling, simulation and design*, 2020.

[13] Q. Gu, Q. Wang, X. Li, and X. Li, "A surrogate-assisted multi-objective particle swarm optimization of expensive constrained combinatorial optimization problems," *Knowledge-Based Systems*, vol. 223, p. 107049, 2021.

[14] Ø. Magnussen, M. Ottestad, and G. Hovland, "Multicopter design optimization and validation," 2015.

[15] L. Carlone and C. Pinciroli, "Robot co-design: beyond the monotone case," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3024–3030.

[16] A. Spielberg, B. Araki, C. Sung, R. Tedrake, and D. Rus, "Functional co-optimization of articulated robots," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5035–5042.

[17] S. Ha, S. Coros, A. Alspach, J. Kim, and K. Yamane, "Computational co-optimization of design parameters and motion trajectories for robotic systems," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1521–1536, 2018.

[18] T. Du, A. Schulz, B. Zhu, B. Bickel, and W. Matusik, "Computational multicopter design," 2016.

[19] B. H. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*. Springer, 2011, vol. 1.

[20] N. Ebrahimi, T. Guda, M. Alamaniotis, D. Miserlis, and A. Jafari, "Design optimization of a novel networked electromagnetic soft actuators system based on branch and bound algorithm," *IEEE Access*, vol. 8, pp. 119 324–119 335, 2020.

[21] M. Solimanpur and A. Jafari, "Optimal solution for the two-dimensional facility layout problem using a branch-and-bound algorithm," *Computers & Industrial Engineering*, vol. 55, no. 3, pp. 606–619, 2008.

[22] B. Taylor and L. Pileggi, "Exact combinatorial optimization methods for physical design of regular logic bricks," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 344–349.

[23] S. J. Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[24] B. Liu and Y.-W. Ku, "Constraintlisp: an object-oriented constraint programming language," *ACM SIGPLAN Notices*, vol. 27, no. 11, pp. 17–26, 1992.

[25] B. Cheng, K. M. F. Choi, J. H.-M. Lee, and J. Wu, "Increasing constraint propagation by redundant modeling: an experience report," *Constraints*, vol. 4, pp. 167–192, 1999.

[26] P. Van Hentenryck, "Parallel constraint satisfaction in logic programming: Prelimiary results of chip with pepsys," in *6th International Conference on Logic Programming*, 1989, pp. 165–180.

[27] N. Beldiceanu and E. Contejean, "Introducing global constraints in chip," *Mathematical and computer Modelling*, vol. 20, no. 12, pp. 97–123, 1994.