Nonconvex Optimization Is Combinatorial Optimization

Abram L. Friesen **Pedro Domingos**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195 USA
{afriesen,pedrod}@cs.washington.edu

Abstract

Difficult nonconvex optimization problems contain a combinatorial number of local optima, making them extremely challenging for modern solvers. We present a novel nonconvex optimization algorithm that explicitly finds and exploits local structure in the objective function in order to decompose it into subproblems, exponentially reducing the size of the search space. Our algorithm's use of decomposition, branch & bound, and caching solidifies the connection between nonconvex optimization and combinatorial optimization. We discuss preliminary experimental results on protein folding, Gaussian mixture models, and bundle adjustment.

1 Introduction

Difficulty in nonconvex optimization arises from a combinatorial explosion of modes in the objective function. Current methods resort to using convex optimizers in conjunction with randomness [1, 2, 3] or local linearization [4, 5], leaving them unable to explore the exponential number of local optima present in these problems. While these techniques yield impressive results in certain domains, their inability to deal with more complicated underlying structure hinders them in many important problems (e.g., protein folding [6, 7], MAP inference [8]).

Fortunately, techniques from combinatorial optimization provide methods for effectively performing exponentially more search in the same amount of time – namely, decomposition, branch & bound, and caching. Building on similarities with combinatorial optimization, our insight is that optimizing a nonconvex function consisting of a sum of terms, each of which is over a subset of the variables, can be performed exponentially faster if the objective function decomposes into independent subgroups of terms. Our ideas and techniques are based on model counting (#SAT) (see [9]), but are also related to variable elimination [10], recursive condition [11], and, even more closely, sum-product networks [12], since we exploit local structure and are not restricted to conditional independencies.

Our contribution is the first nonconvex optimization algorithm that finds and exploits local structure in the objective function in order to decompose the problem, reducing the size of the search space by an exponential factor. In the following section, we explain the algorithm, first at a high level and then in more detail, providing both pseudocode and a theoretical guarantee of optimality. Section 3 presents a brief description of experiments on protein folding, Gaussian mixture models, and bundle adjustment [13]. Finally, Section 4 concludes with a summary and discussion of future work.

2 A combinatorial algorithm for nonconvex optimization

Decomposition in the objective function occurs for reasons both structural and approximate. The simplest form of decomposition exists when separate terms contain non-overlapping subsets of variables, e.g., $\min_{x,y} t_0(x) + t_1(y) = \min_x t_0(x) + \min_y t_1(y)$. These functions are identical, but the minimization on the right is significantly cheaper than that on the left. Conditional independence

(as exploited in variable elimination and recursive conditioning) allows more general decomposition. For example, we can decompose $\min_{x,y,z} t_0(x,z) + t_1(y,z)$ by conditioning on z, to give $\min_z \{\min_x t_0(x,z) + \min_y t_1(y,z)\}$. Variable elimination and recursive conditioning are inherently discrete techniques, but the same principle applies in continuous domains as well. Independence due to local structure, which is strictly more general than conditional independence, can occur if, for example, we are optimizing $\min_{x,y,z} t_0(x) + t_1(x,y,z) + t_2(y)$ and, by setting $z = c$, term t_1 becomes approximately constant, in the sense that its variation is dominated by that of t_0 and t_1, resulting in $k + \min_x t_0(x) + \min_y t_2(y)$. Additional local decomposition is possible if terms have specific internal structure that can be exploited (e.g., by setting part of the term to a constant). We refer to this process of setting terms or subterms to constants as simplification.

Our algorithm begins by heuristically choosing a variable to condition on. It then explores this variable's domain, assigning a finite set of values from the domain, chosen to guarantee that we do not incur more than ϵ error (see Section 2.2). For each assignment, our algorithm simplifies the objective function, decomposes it into independent sub-functions consisting of independent sets of terms and variables, and then recurses on these sub-functions, summing the results of these recursive calls to get the evaluation of the function. Pseudocode of our algorithm is shown in Algorithm 1. The recursive structure allows us to choose variables and their values based on local information, maximizing simplification and decomposition. In the implementation of our algorithm, we employ branch & bound and caching to provide additional exponential computational gains. These are omitted from Algorithm 1 for clarity, but are discussed briefly in Section 2.3.

For variable selection (line 1 of Algorithm 1), we use a MOMS-type (maximum occurrences of minimum size [14]) structural heuristic that chooses the variable that shares terms with the largest number of unique variables. This prioritizes variables with more dependencies and aids decomposition by assigning those first. Note that this heuristic makes local decisions because the current sets of variables and terms are determined by the earlier assignments, simplifications, and decompositions that have occurred, which are determined locally. For choosing values of a variable (line 3 of Algorithm 1), we begin with the value that resulted in the minimum in a neighboring region and then iteratively step away from that value, preferring to move downhill since that will take us to the minimum in fewer steps. When choosing the step size, we assume Lipschitz continuity and compute the largest step in the variable's domain that guarantees that we will never incur an error larger than a specified amount, relative to the global optimum (see Section 2.2 for more details).

Algorithm 1: Approximate minimization of a nonconvex function by recursive decomposition into locally independent subspaces.

Function Minimize(f, ϵ)**:**

Input: The function $f(X) = \sum t_j(X_j)$ over the variables, $X = \{x_i\}$, contained in the terms $T = \{t_j\}$, each of which is over a subset of the variables X_j.
The allowed approximation error, ϵ.

Output: Value within ϵ of the global minimum of $f(X)$ in the hyperrectangle defined by the domains of X, $\{dom(x_i)\}$.

1 $x \leftarrow$ chooseVar(X) // *pick x to maximize decomposition*
2 **while** x's domain could contain a new minimum
3 $x = (v \leftarrow$ chooseVal$(dom(x), \epsilon)$ // *pick value v and assign to x*
4 simplify(T_x, ϵ) // *simplify terms containing x*
5 $\{f^k(X^k)\} \leftarrow$ decompose(f, T, X) // *split into indep. functions*
6 $f_{x=v} \leftarrow \sum_k$ Minimize$\left(f^k, \epsilon\right)$ // *recurse over simplified functions*
7 $f^{min} \leftarrow \min\{f^{min}, f_{x=v}\}$
return f^{min}

2.1 An example

A domain to which our algorithm seems particularly well-suited is protein folding [6, 7]. Protein folding is the process by which a protein, consisting of a long chain of amino acids, assumes its functional shape. The computational problem is to predict this final conformation from the known sequence of amino acids. This requires minimizing an energy function consisting mainly of a sum of pairwise distance-based terms representing chemical bonds, hydrophobic interactions, electrostatic forces, etc., where the variables are the relative angles between the atoms. A good analogy for

protein folding is solving a jigsaw puzzle, since atoms cannot pass through each other and amino acids bond to one another based on their 3-D shape and specific chemical properties, much like puzzle pieces only fit in one specific location. While a jigsaw puzzle is inherently continuous, solving it involves independently grouping similar pieces and building a solution by fitting specific pieces together, instead of trying all pieces against all other pieces. Similarly, our algorithm reinterprets continuous optimization as combinatorial optimization by breaking the objective function into groups, solving those groups independently, and constructing the overall solution from those pieces.

In addition, protein folding is near impossible for optimization techniques that only exploit conditional independencies, since almost every pairwise interaction is meaningful at some point in the space and thus there are no conditional independencies. However, at any point in the conformation space, the majority of these energy terms are approximately zero, since they are distance-based terms with large negative exponents. Thus, by taking advantage of local structure, we can achieve decompositions that algorithms like variable elimination cannot. Protein folding provides a good example of why we selected our particular heuristic for choosing variables. In protein folding, there typically exists a backbone of atoms or amino acids that, once set, causes the remainder of the variables to decompose into independent subsets. Many other problems seem to exhibit this same backbone structure (e.g., SAT [15]) and, thus, we use a MOMS-type heuristic to prioritize backbone variables, which tend to be the most highly-connected variables.

2.2 Continuous aspects

Considerations of continuity appear when selecting the values of variables. The parameter ϵ, the absolute error allowed in the returned minimum relative to the (unknown) global minimum, controls the tradeoff between accuracy of the solution and the complexity of the computation – as ϵ is increased, the error increases, but the number of values the algorithm needs to consider decreases. The following theoretical results provide bounds on the error in the minimum returned by our algorithm. Briefly, our assumption of Lipschitz continuity allows us to explore each variable in a finite number of steps, while only incurring a fixed amount of error, ϵ_v, for each variable. In addition, our algorithm computes the bounds of each term and simplifies those with bounds of width less than ϵ_s. With these two mechanisms, we can bound the error in the minimum returned by our algorithm to $\epsilon = n\epsilon_v + m\epsilon_s$, for an objective function with n variables and m terms.

For any f, let $f^*(X)$ denote the global minimum.

Lemma 1. *For a chosen $\epsilon_v > 0$, if $f(x) \in \mathbb{R}$ is Lipschitz continuous and any evaluation $f(x = a)$ has error at most ϵ_e, then our algorithm computes $f^{min}(x)$, with $\left|f^{min}(x) - f^*(x)\right| \leq (\epsilon_v + \epsilon_e)$.*

Lemma 2. *If $f(x_0, \ldots, x_n) = \sum_{j=1}^{m} t_j(X_j)$ and $t_k(X_k) \in [L, U]$ then the maximum error incurred in $f(x_0, \ldots, x_n)$ from setting $t_k(X_k) = \frac{U+L}{2}$ is $\frac{U-L}{2}$.*

Our algorithm computes $[L_k, U_k]$ for all t_k, simplifying terms that incur error less than ϵ_s. Let ρ be an assignment to a subset of variables such that $f_\rho(x_0)$=$f(x_0, x_1$=$\rho_1, ..., x_n$=$\rho_n)$. Let m_i be the number of terms containing only x_i (ignoring variables set to a constant) at a specific recursion level.

Theorem 1. *Leaf nodes of the recursion return $f^{min}_{\rho_c}(x_c)$, such that $\left|f^{min}_{\rho_c}(x_c) - f^*_{\rho_c}(x_c)\right| \leq \epsilon_v + m_c\epsilon_s$. Similarly, the parents of leaf nodes return $f^{min}_{\rho_p}(x_p, \{x_c\})$, such that $\left|f^{min}_{\rho_p}(x_p, \{x_c\}) - f^*_{\rho_p}(x_p, \{x_c\})\right| \leq \epsilon_v + m_p\epsilon_s + \sum_{\{x_c\}} \epsilon_v + m_c\epsilon_s$.*

Theorem 2. *When minimizing $f(x_0, \ldots, x_n) = \sum_{j=1}^{m} t_j(X_j)$, the output of our algorithm, $f^{min}(x_0, \ldots, x_n)$, has error $\epsilon = n\epsilon_v + m\epsilon_s$, such that $\left|f^{min}(x_0, \ldots, x_n) - f^*(x_0, \ldots, x_n)\right| \leq \epsilon$.*

We omit all proofs here due to lack of space.

2.3 Additional details

Decomposition produces significant improvements in performance but does not address scenarios where entire regions of the space cannot contain a better minimum. We employ branch & bound [16] to stop exploring a region of space, potentially avoiding an exponential number of computations, if the lower bound for that region exceeds the upper bound on the minimum of the objective function. However, decomposition can still result in an exponential number of repeated computations. To prevent this, we cache the computed f^{min} of each sub-function and check the cache at each level

of recursion before performing the computation. A major distinction between our cache and that of #SAT [17] or recursive conditioning, is that our cache must be continuous and approximate. Finally, despite these considerable gains, most problems remain exponential, making joint optimization over large numbers of variables impossible. Instead, we trade locality for computation time by iteratively performing a series of optimizations over subdomains of the variables (not shown in Algorithm 1). This process resembles gradient descent, except that at each step our algorithm computes the optimum over a small hyperrectangle, instead of taking a step in the gradient direction. As such, we are able to jump over nearby local optima that can trap gradient descent.

3 Experiments

We applied our algorithm to a restricted protein folding [6, 7] problem: sidechain optimization. Amino acids consist of a backbone and a sidechain. In this problem, the backbone atoms are fixed and only the sidechain atoms can move. The state of the art [18, 19] is for a discretized version of the problem, and, simply by using our algorithm as a combinatorial optimizer, we achieve comparable results on many of the proteins in their dataset. However, our algorithm's strength lies in its ability to solve continuous optimization problems, which the existing solvers are unable to do, so we anticipate even stronger results on continuous versions. The proteins that we find difficult are those with many dependencies that do not exhibit much decomposition. We are working on exploiting more structure in this domain both to handle these challenging proteins and to solve the full protein folding problem.

We've also experimented with fitting Gaussian mixture models (GMMs) with diagonal covariance matrices to data. Here, dependencies between components form the underlying structure, where dependencies exists if a data point is similarly likely to be from multiple components. Our algorithm performs well here when independencies between the components exist and decomposition occurs. These independencies occur even if components are not well-separated in space; instead, components must be separated by each other (e.g., a line of components). Exploiting within-term simplifications significantly increases the amount of decomposition. The remaining challenge is that components grow their covariances to move. Thus, if the initialization is poor then most components become large, resulting in many dependencies and eliminating decomposition. We are investigating incorporating gradient descent and sampling methods to assist in these highly inter-dependent regions.

Finally, bundle adjustment [13] is a promising domain for our algorithm. The goal is to minimize the error between a dataset of 2-D images and a projection of learned 3-D points representing a scene's geometry onto learned camera models. This problem is highly-structured: cameras are explicitly dependent on the points and only implicitly on other cameras, creating a bipartite dependency graph that is simpler to decompose than a general graph. Bundle adjustment has the same the jigsaw-puzzle-like nature as protein folding: the cameras and points must fit together, but there are thousands to millions of squared-error terms that create strong constraints on the variables. This implies that exploring the underlying modes approximately is more important than finding the exact minimum of one local optimum, making this problem well-suited to our algorithm. As in the previous experiments, our algorithm performs well when decomposition is possible. We are currently improving our implementation to handle the large numbers of variables present in this task.

4 Discussion

Nonconvex optimization problems are prevalent in both the scientific and the machine learning communities. While convexity-based solvers have made remarkable progress on many of these problems, they are simply unable to cope with the exponential number of local optima present in many of these domains. To this end, we have introduced a novel, fixed-error, nonconvex optimization algorithm that uses techniques from combinatorial optimization to exploit local structure and decomposition in the objective function, exponentially reducing the size of the search space and bridging the divide between nonconvex optimization and combinatorial optimization. Our algorithm's combinatorial roots allow it to handle both discrete and continuous variables and it can trivially support constrained optimization by rewriting the constraints as terms in the objective function. We are currently working on finishing the experiments outlined above and extending our algorithm to incorporate a combination of gradient descent and random sampling. For future work, our algorithm's use of the min-sum semiring to achieve decomposition should make it straightforward to extend our algorithm to the sum-product semiring, creating an algorithm for continuous integration.

References

[1] Scott Kirkpatrick, D. Gelatt Jr., and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[2] Harold H. Szu. Non-convex optimization. In *Proceedings of the SPIE*, volume 698, pages 59–67, 1986.

[3] Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[4] Jorge Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In G.A. Watson, editor, *Numerical Analysis*, volume 630 of *Lecture Notes in Mathematics*, pages 105–116. Springer Berlin Heidelberg, 1978.

[5] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 2006.

[6] Christian B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(4096):223–230, 1973.

[7] David Baker. A surprising simplicity to protein folding. *Nature*, 405:39–42, 2000.

[8] James Park. MAP complexity results and approximation methods. In *Proceedings of the Eighteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 388–396, San Francisco, CA, 2002. Morgan Kaufmann.

[9] Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI/IAAI*, pages 157–162, 2000.

[10] Nevin Zhang and David Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994.

[11] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

[12] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

[13] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment – a modern synthesis. In *Vision algorithms: theory and practice*, pages 298–372. Springer, 2000.

[14] Jon William Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, University of Pennsylvania, 1995.

[15] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400(6740):133–137, 1999.

[16] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):pp. 699–719, 1966.

[17] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. DPLL with caching: A new algorithm for #SAT and Bayesian inference. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 10, 2003.

[18] Chen Yanover, Talya Meltzer, and Yair Weiss. Linear programming relaxations and belief propagation – an empirical study. *The Journal of Machine Learning Research*, 7:1887–1907, 2006.

[19] David Sontag and Tommi S. Jaakkola. New outer bounds on the marginal polytope. In *Advances in Neural Information Processing Systems*, pages 1393–1400, 2007.